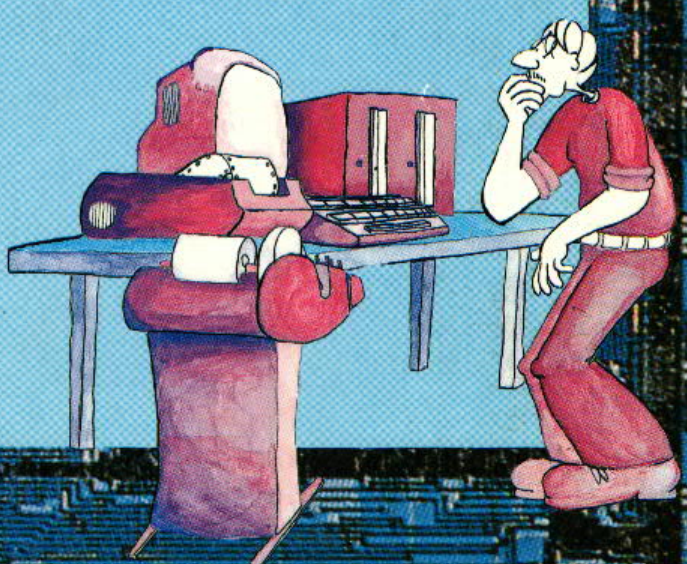
MIKROCOMPUTER-GRUNDWISSEN

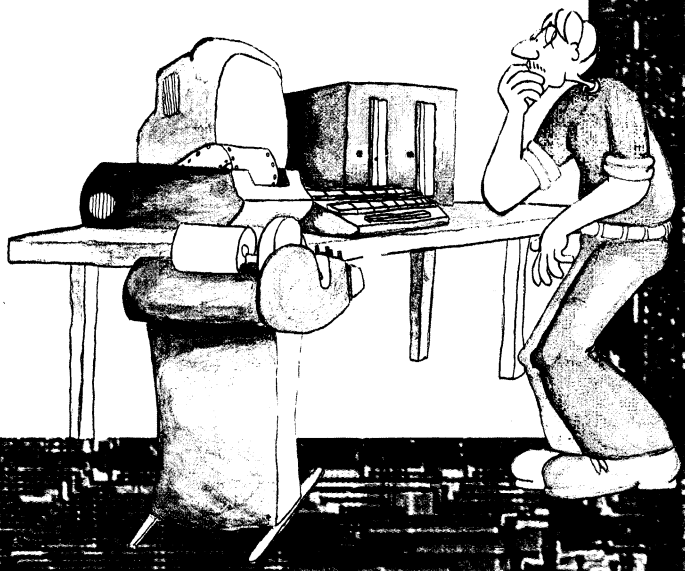
Eine allgemeinverständliche
Einführung in die
Mikrocomputer-Technik



Von Adam Osborne

MIKROCOMPUTER-GRUNDWISSEN

Eine allgemeinverständliche Einführung in die Mikrocomputer-Technik



Von Adam Osborne

MIKROCOMPUTER-GRUNDWISSEN

Eine allgemeinverständliche
Einführung in die
Mikrocomputer-Technik

Von Adam Osborne

te-wi Verlag GmbH

München

IMPRESSUM

Dieses Buch ist eine Übersetzung aus der amerikanischen Originalausgabe "AN INTRODUCTION TO MICROCOMPUTERS, Volume 0, The Beginner's Book" by Adam Osborne.

LIZENZNEHMER und HERAUSGEBER:
te-wi Verlag GmbH, Theo-Prosel-Weg 1
8000 München 40

GESAMTHERSTELLUNG:
technik marketing, München
tm 3409/384/4. Auflage
Printed in W-Germany

ISBN 3-921803-02-0

MIKROCOMPUTER-GRUNDWISSEN

Eine allgemeinverständliche
Einführung in die
Mikrocomputer-Technik

Von Adam Osborne

te-wi Verlag GmbH

München

IMPRESSUM

Dieses Buch ist eine Übersetzung aus der amerikanischen Originalausgabe "AN INTRODUCTION TO MICROCOMPUTERS, Volume 0, The Beginner's Book" by Adam Osborne.

LIZENZNEHMER und HERAUSGEBER:
te-wi Verlag GmbH, Theo-Prosel-Weg 1
8000 München 40

GESAMTHERSTELLUNG:
technik marketing, München
tm 3409/384/4. Auflage
Printed in W-Germany

ISBN 3-921803-02-0

INHALTSÜBERSICHT V

EINFÜHRUNG IX

Kapitel		Seite
1	**DIE TEILE, DIE DAS GANZE BILDEN**	1-1
	Ein Mikrocomputer-System	1-3
	Der Bildschirm oder die Videoanzeige	1-3
	Die Tastatur	1-5
	Der Drucker	1-7
	Bauteile, die eine Menge Informationen speichern	1-13
	Speicher	1-14
	Floppy-Disk-Geräte	1-15
	Geräte mit starren Platten	1-22
	Plattenzugriff	1-24
	Logische und physikalische Aufzeichnungen	1-27
	Aufzeichnungen und Dateien	1-29
	Kassettengeräte	1-30
	Lochstreifengeräte	1-43
2	**WIR VERWENDEN EINEN MIKROCOMPUTER UND BEOBACHTEN WIE ER WÄCHST**	2-1
	Ein Programm schaffen und es zum Arbeiten bringen	2-3
	Die Frontplatte eines Mikrocomputers	2-4
	Der Fernschreiber	2-6
	Verwendung eines einfachen Mikrocomputer-Systems	2-15
	Festwertspeicher	2-16
	Tastaturen	2-21
	Einige Anwendungen für Mikrocomputer	2-25
3	**MIKROCOMPUTER-SYSTEMKOMPONENTEN – WAS WIR SEHEN IST NICHT IMMER DAS WAS WIR BEKOMMEN**	3-1
	Physikalische und logische Einheiten in Mikrocomputer-Systemen	3-2
	Mikrocomputer-Hardwarekomponenten	3-6
	Erneute Zuweisungen logischer Einheiten	3-11
	Gerätetreiber	3-12

Kapitel Seite

Wahlmöglichkeiten für Mikrocomputer-Systemkomponenten 3-17
Sichtgeräte 3-17
Tastaturen 3-27
Drucker 3-36
Massenspeicher 3-45

4 WIR KOMMEN ZU DEN GRUNDLAGEN 4-1
Zahlen und Logik 4-2
Binäre Daten 4-2
Das binäre Zahlensystem 4-4
Umwandlung binär in dezimal 4-11
Umwandlung dezimal in binär 4-12
Bits, "Nibbles" und Bytes 4-16
Binäre Arithmetik 4-18
Binäre Addition 4-18
Binäre Subtraktion und negative Zahlen 4-22
Binäre Multiplikation und Division 4-33
Oktal- und Hexadezimalzahlen 4-34
Oktal-Hexadezimalumwandlungen 4-37
Umwandlungen Dezimal-Oktal und Dezimal-Hexadezimal 4-38
Zeichencodes 4-40
Computerlogik und Boolesche Operationen 4-43
Status-Flags 4-43
Logische Operatoren 4-45
Der NICHT-Operator 4-46
Der UND-Operator 4-46
Der ODER-Operator 4-48
Der XOR-Operator 4-49

5 DAS INNENLEBEN EINES MIKROCOMPUTERS 5-1
Programmiersprachen 5-1
Ein Vergleich zwischen höheren Sprachen und der Assemblersprache 5-6
Die Funktionslogik des Mikrocomputers 5-10
Informationswege 5-12
Die Zentraleinheit 5-14
Serielle Logik 5-14

Kapitel Seite

Serieller Logikschritt 5-18
Datenspeicherung in der Zentraleinheit 5-20
Programmspeicher 5-20
Speicherplätze und Adressen 5-21
Datenspeicher 5-22
Folge der Ereignisse bei einem Additionsprogramm 5-22

6 NUN FÜGEN WIR ALLES ZUSAMMEN 6-1
Wortgröße 6-1
Busse 6-4
Darstellung von Signalen auf den Bus-Leitungen . . 6-4
Register 6-5
Die arithmetische und logische Einheit 6-6
Die zusätzliche CPU-Logik 6-9
Datenregister 6-9
Verwendung von Datenregistern 6-12
Das Befehlsregister und das Steuerwerk 6-15
Logische Konzepte und zeitliche Steuerung . . . 6-18
Logik zur Bewegung binärer Daten 6-18
Das Taktsignal und die zeitliche Steuerung der Befehlsausführung 6-27
Speicherzugriff 6-31
Speicheradressier-Logik 6-35
Adressierung des Programmspeichers und der Befehlszähler 6-44
Programmlogik und der Befehlszähler 6-45
Adressier-Register für Datenspeicher 6-48
Adressierung externer Logik 6-49
Befehlsvorrat und Programmierung 6-49

ANHANG

A **Standard-Zeichencodes** A-1
B **Standardsymbole für Flußdiagramme** B-1
C **μP-Lexi** C-1

Verzeichnis der Bilder

Bild		Seite
1-1	Ein typisches Mikrocomputer-System	1-2
1-2	Die beschriebene Oberfläche einer Diskette	1-20
2-1	Ein Flußdiagramm für Joe's Programm zur Zahlung von Rechnungen	2-24
3-1	Logische Einheiten, die einen Mikrocomputer umgeben	3-4
3-2	Die logischen Einheiten für das Mikrocomputer-System von Bild 3.1	3-5
3-3	Logische Einheiten für ein Fernschreiber-Terminal	3-5
3-4	Logische und physikalische Einheiten, die mittels eines Gerätetreibers verbunden sind	3-13
3-5	Die Verwendung von Gerätetreiberprogrammen zum Ersatz physikalischer Einheiten	3-15
3-6	Flußdiagramm für ein einfaches Videoanzeigen-Treiberprogramm	3-18
4-1	Zerlegung gepackter Byte- und ASCII-Code-Bildungslogik	4-44
5-1	Mikrocomputer-Funktionslogik	5-10
5-2	Mikrocomputer-Funktionslogik, die bei der Bewegung und Speicherung von Daten beteiligt ist	5-12
5-3	Mikrocomputer-Funktionslogik, die bei der Modifikation von Daten beteiligt ist	5-12
6-1	Die arithmetische und logische Einheit	6-7
6-2	Mikrocomputer-Systembusse	6-36

Verzeichnis der Tabellen

Tabelle		Seite
4-1	Alle vierstelligen Binärzahlen und ihre Dezimaldarstellungen	4-15
4-2	Die größte Zahl, die mit Binärzahlen mit 1 bis 16 Stellen dargestellt werden kann	4-16
4-3	Zahlensysteme	4-37

EINFÜHRUNG

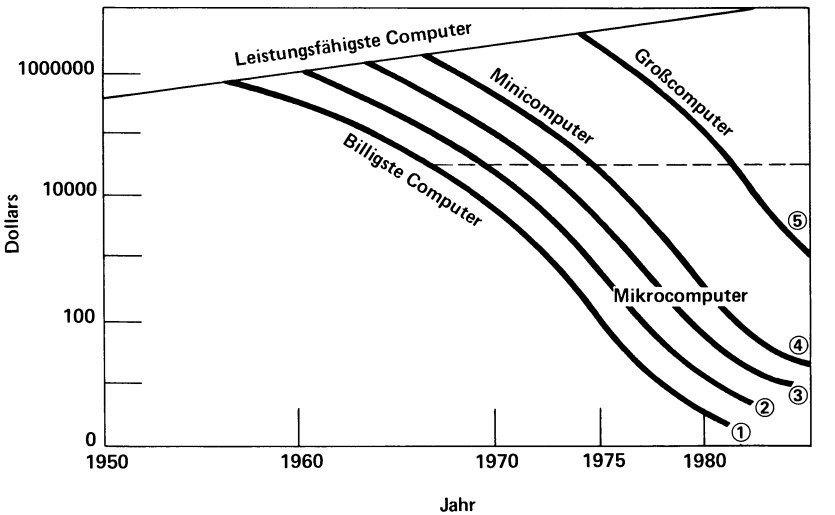
Im Mittelpunkt dieses Buches steht der Mikrocomputer. Computer sind heute in unserer Industriegesellschaft, wie das Auto und die Elektrizität, ein nicht mehr fortzudenkender Bestandteil unseres täglichen Lebens geworden. Während eines normalen Tages kommen wir mehrmals mit Computern in Berührung. Und die Zukunft wird uns noch mehr Computer bringen.

Die nationale US-Volkszählung im Jahre 1950 wurde durch die ENIACI ermöglicht, die als der erste kommerzielle Computer der Welt angesehen wird. ENIACI kostete mehr als eine halbe Million Dollar (und zwar 1950er Dollar). Heute kann man dieselbe Rechenleistung für zehn Dollar kaufen. Tatsächlich sind Computer heute derart billig geworden, daß man sie zur Steuerung von Bildschirmspielen, Kinderspielzeug, Zeitgebern für Herde, Nähmaschinen und Waschmaschinen verwenden kann. In nicht allzu ferner Zukunft wird wahrscheinlich jedes Auto zwei oder drei Computer zur Steuerung des Motors, der Anbaugeräte und der Armaturenanzeigen enthalten.

Wir wissen alle, was die Elektronik bei den Taschenrechnern und der Uhrenindustrie bewirkt hat. Elektronische Rechner und Uhren werden von Bausteinen gesteuert, die in Wirklichkeit nichts anderes als kleine Computer sind. Die Musikindustrie wird als nächstes durch die Computer und die Elektronik revolutioniert.

Während der letzten 25 Jahre wurden eine ganze Reihe neuer Erfindungen gemacht, die die Herstellungskosten von Rechenschaltungen drastisch reduziert haben. Aber überraschenderweise hat sich an den grundlegenden Computerkonzepten wenig geändert. Daher wurde bei jedem neuen technologischen Durchbruch derselbe Computer wie vorher gebaut aber weniger dafür

bezahlt. Die folgende Grafik veranschaulicht die Preisentwicklung der letzten Jahre.



Im Laufe der Jahre wurde der Unterschied zwischen dem leistungsfähigsten und dem teuersten Computer ausgeprägter. In der gezeigten Grafik bedeuten die mit ①, ②, ③, ④ und ⑤ gekennzeichneten Kurven etwa gleichwertige Computer, d.h. Computer mit derselben Leistung. Sehen wir uns einmal beispielsweise die mit ② bezeichnete Kurve an. Sie zeigt uns, daß es 1960 für 1000000 Dollar den leistungsfähigsten Computer gab. Ein etwa gleichwertiger Computer wurde 1975 für ca. 1000,– Dollar verkauft, der aber längst nicht mehr als Hochleistungs-Computer angesehen werden kann, verglichen mit all den neuen seit 1975 erhältlichen noch leistungsfähigeren Geräten.

MINICOMPUTER GROSSCOMPUTER

Durch die großen Preis- und Leistungsunterschiede der Computer in den sechziger Jahren entstand naturgemäß eine bestimmte Klassifizierung der Geräte. Um 1965 begann man die billigsten Computer als Minicomputer zu bezeichnen. Leistungsfähigere Computer wurden zur Unterscheidung in die Kategorie der

Großcomputer (mainframes) eingereiht. Die Vorsilbe "Mini" resultiert vor allem aus der Tatsache, daß die neuen, preisgünstigen Computer in den Abmessungen kleiner als ihre Vorgänger waren. Der Unterschied zwischen einem Minicomputer und einem Großcomputer ist schwer zu definieren. Die einzigen Unterschiede sind wirklich nur Preis und Größe. Minicomputer sind wesentlich billiger als Großcomputer und im allgemeinen weniger leistungsfähig, obwohl es beträchtliche Überschneidungen gibt.

MIKROCOMPUTER

Um 1972 erschienen sehr billige Computerprodukte und wurden Mikrocomputer genannt. Die Vorsilbe "Mikro" wurde wegen der extremen Kleinheit des Produkts im Vergleich zu einem "Mini" angewandt, ebenso wie die Vorsilbe "Mini" auf den kleineren Abmessungen dieses Produkts im Vergleich zu einem Großcomputer begründet war. Aber wieder gibt es Überschneidungen mit Produkten, die als Mikrocomputer bezeichnet werden, und Produkten, die man Minicomputer nennt. Die Überlappung betrifft sowohl Leistung wie Abmessungen. Die heutzutage erhältlichen leistungsfähigsten Mikrocomputer können mehr als die Minicomputer mit der geringsten Leistung. Daher ersetzen Mikrocomputer häufig Minicomputer oder werden wie Minicomputer verwendet.

Es wäre sinnlos, in diesem Buch die Unterschiede zwischen Mikrocomputern und Minicomputern exakt zu definieren. Denn die manchmal recht komplexen Zusammenhänge können logischerweise nur dann begriffen werden, nachdem man Computer im allgemeinen versteht.

WIE DIESES BUCH GEDRUCKT IST

"Mikrocomputer-Grundwissen" enthält einen sehr breiten Bereich an Informationen, und daher wird der Text in **fetten** und mageren Buchstaben gedruckt. Der Zweck des verschiedenartigen Druckes besteht darin, daß man die wesentlichen Informationen rasch herausfinden kann. Der fettgedruckte Text hebt alle wesentlichen Informationen hervor. Hat man diesen Text nicht vollkommen verstanden, kann man dem mager gedruckten Text ausführlichere Informationen entnehmen.

MIKROCOMPUTER-GRUNDWISSEN

Dieses Buch ist das erste in einer umfangreichen Reihe von Mikrocomputer-Fachbüchern und wendet sich an alle, die sich privat und beruflich für dieses hochinteressante Gebiet der Technik begeistern oder sich einfach, aus welchen Gründen auch immer, damit auseinandersetzen müssen. Denn "Mikrocomputer-Grundwissen" ist der "maßgeschneiderte Anzug" für alle, die sich die Mikrocomputer-Technik von Null an erarbeiten wollen.

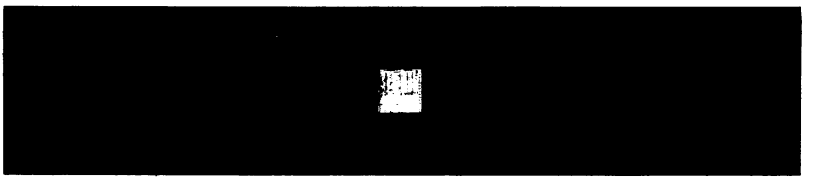
Nach sechs Lernschritten beherrschen Sie die Materie und können mitreden, wenn es um das aktuelle Thema MIKROCOMPUTER geht.

Mit den Testfragen am Ende jedes Kapitels können Sie Ihren Lernerfolg kontrollieren und bestätigt sehen.

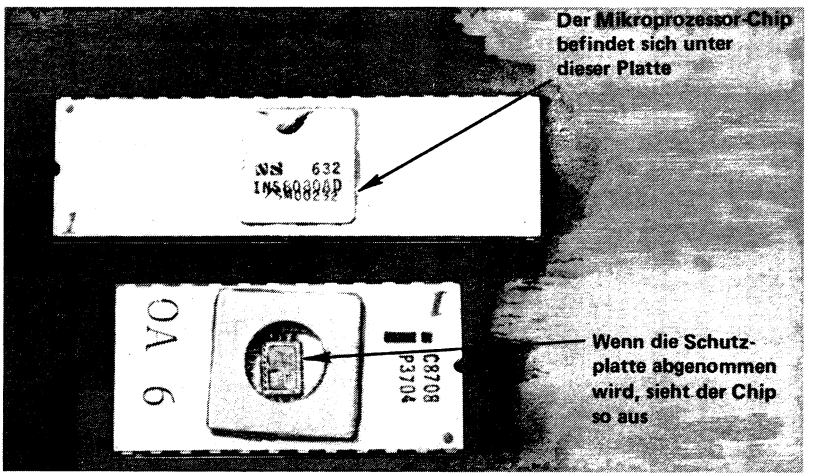
Kapitel 1
DIE TEILE, DIE DAS GANZE BILDEN

GROSS-INTEGRATION
LSI (LARGE SCALE INTEGRATION)

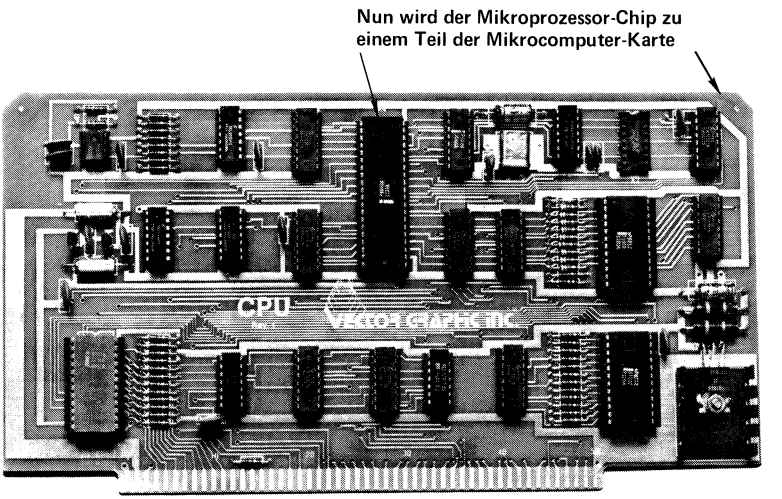
Das Zeitalter der Computerindustrie ist das Ergebnis einer neuen Technologie, mit der Zehntausende von mikroskopisch kleinen elektronischen Schaltungen auf einer Fläche von wenigen mm^2 untergebracht werden können. Diese neue Technologie wird als Groß-Integration (Large Scale Integration = LSI) bezeichnet. Wir sehen hier einen LSI-Baustein, dargestellt in seiner natürlichen Größe:



Es ist heute möglich, in einem einzigen LSI-Baustein, nicht größer als der gezeigte, alle Schaltungen unterzubringen, die man für das „Gehirn" eines einfachen Computers benötigt. Wir nennen diesen LSI-Baustein einen Mikroprozessor. Der Mikroprozessor ist der eigentliche „Computer", d. h. der „Rechner" innerhalb eines Mikrocomputer-Systems. Dies ist die tatsächliche Größe eines Mikroprozessors:



Wir wissen nun, wie ein Mikroprozessor aussieht. Zunächst können wir ihn aber vergessen, denn erst wesentlich später in diesem Buch kehren wir wieder detailliert zu den Mikroprozessoren zurück. Der „Mikroprozessor" wird zu einem kleinen Teil eines „Mikrocomputers":



Ein Mikrocomputer wiederum ist ein kleiner Teil eines „Mikrocomputer-Systems". Wenn wir zum ersten Mal ein „Mikrocomputer-System" sehen, so wird dies weitgehend wie jedes andere Computersystem aussehen. Ein typisches Mikrocomputer-System ist in Bild 1-1 dargestellt:

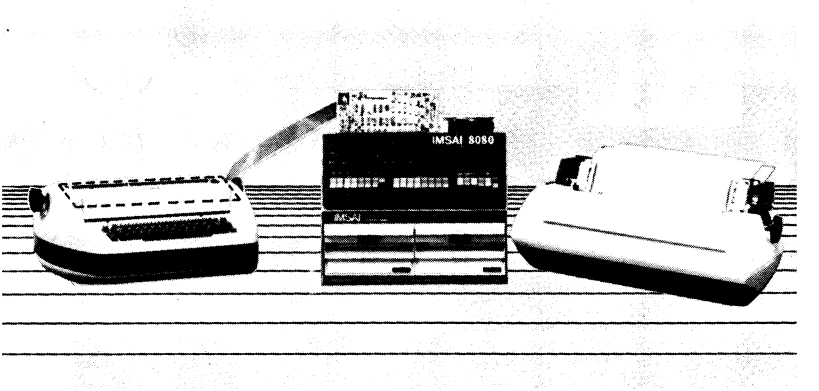


Bild 1.1 Ein typisches Mikrocomputer-System

Ein Mikrocomputer-System, dargestellt in Bild 1-1, ist ein Hilfsmittel zur Erledigung einer Aufgabe. Derart oberflächlich betrachtet, müssen wir auch gar nicht wissen, daß wir es mit einem Computer zu tun haben und dies ist in manchen Fällen zum allgemeinen Verständnis auch völlig ausreichend. **Wenn wir beispielsweise ein Mikrocomputer-System in unserem Büro zur Lohnverrechnung einsetzen, so wird dieses Ding zu einer Büromaschine.** Sie ist für uns ein Hilfsmittel für die Eingabe der Lohndaten und gibt uns hoffentlich eine exakte Lohnverrechnung. Vorausgesetzt wir erhalten die gewünschten Ergebnisse, müssen wir nicht wissen, wie der Computer arbeitet, ebensoviel wie wir von der Funktion eines Strahltriebwerkes wissen müssen, um mit einer kommerziellen Luftlinie zu fliegen.

Nun gibt es wahrscheinlich keinen Grund, warum wir jemals lernen sollten wie ein Düsenantrieb funktioniert. Wenn der Konstrukteur des Strahltriebwerkes auf dem nächsten Flug neben uns sitzt, so werden ihn seine Kenntnisse nicht schneller zu seinem Bestimmungsort bringen als uns selbst, noch wird dadurch seine Reise in irgendeiner Art effektiver als unsere. Wenn wir jedoch nicht wissen wie Computer arbeiten, so sollte uns dies traurig stimmen. Wollen wir, daß unser Buchhalter in einer Welt lebt, die über unser Begriffsvermögen hinausgeht? Werden wir sicher sein, daß der Buchhalter immer ein ehrlicher Mensch bleiben wird? Sicherlich nicht. Warum also annehmen, daß unser Programmierer immer ehrlich sein wird, wenn er niemals überprüft wird und in einer Welt arbeitet, die nur er versteht?

Und wenn wir bei unserer Arbeit nicht mit Computern zu tun haben, so sollten wir trotzdem ihren Einfluß in unserem Alltagsleben betrachten. Wie lange wollen wir die Entschuldigung „Der Computer hat sich geirrt" annehmen? Wir haben keine andere Wahl, als diese Entschuldigung zu akzeptieren, bis uns jemand sagt wie Computer arbeiten — und damit wollen wir nun beginnen.

EIN MIKROCOMPUTER-SYSTEM

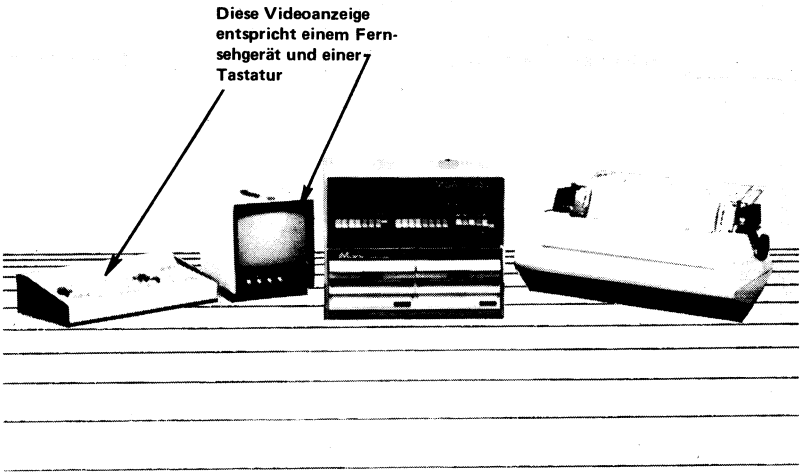
Wir beginnen mit der Betrachtung eines ganzen Mikrocomputer-Systems und prüfen, wie jeder Teil des Systems arbeitet.

DER BILDSCHIRM ODER DIE VIDEOANZEIGE

Die auffallendsten Teile des in Bild 1-1 gezeigten Mikrocomputer-Systems sind jene, mit denen die Kommunikation vom Computer zum Menschen hergestellt wird. Wir führen einen Dialog mit dem Mikrocomputer-System, um Daten einzugeben. **Das Mikrocomputer-System spricht zu uns, indem es die entsprechenden Nachrichten auf einem Bildschirm darstellt.** Wir antworten hierauf, indem wir eine Antwort über die Tastatur eintippen. Alles was wir eingeben, wird sofort auf dem Schirm dargestellt, um zu sichern, daß wir keine Fehler gemacht haben — und der Mikrocomputer unsere Eingabe korrekt angenommen hat.

Zwischen einem Fernsehgerät und einer Bildschirm- oder Videoanzeige gibt es nur geringe Unterschiede. Der wesentliche Unterschied liegt darin, daß die Videoanzeige eine bessere „Auflösung" besitzt. Das heißt, daß wir kleinere Zeichen auf dem Bildschirm haben können, ohne daß diese Zeichen unscharf oder schlecht leserlich werden. Nehmen wir aber eine gewisse Mindestgröße von Ziffern und Zahlen in Kauf, kön-

nen wir ein Fernsehgerät als Videoanzeige verwenden. Dies kann folgendermaßen gezeigt werden:



Leute mit einem kleinen Budget benützen sehr häufig ein Fernsehgerät als Videoanzeige.

Ein mit Mikrocomputern und Mikrocomputer-Systemen verbundenes Problem ist das der Terminologie und der Sprache. Während ein Teil der Terminologie aus reinem Fachjargon besteht, so ist doch eine gewisse Zahl von Fachausdrücken zum Verständnis erforderlich. Erinnern wir uns daran, daß Englisch (und natürlich auch jede andere Sprache) in nicht-technologischen Gesellschaften entstanden ist, und deshalb sind manchmal Worte oder Ausdrücke, die technische Konzepte ausreichend darstellen, schwer zu finden. In diesem Buch werden daher eine Reihe von Computerausdrücken eingeführt um auch andere Bücher verständlich zu machen.

Warum sprechen wir zum Beispiel von einer „Videoanzeige" anstatt von einem Fernsehbildschirm? Die Antwort ist, daß ein Fernsehbildschirm vor allem zur Darstellung von Bildern entwickelt wurde. Eine Videoanzeige dient jedoch vor allem zur Darstellung gedruckter Wörter. Es gibt daher wesentliche Unterschiede in der Elektronik. Unser Fernsehbildschirm macht Bilder sichtbar, abgestuft nach den einzelnen Tonwertstufen von Schwarz über Grau bis Weiß (bei Farbfernsehern selbstverständlich farbig und Farbsättigungen), benötigt dafür aber keine hohe Auflösung zur Darstellung kleiner Zeichen und Details. Anders dagegen die Videoanzeige. Sie ist zur Darstellung von Texten konzipiert und bildet daher nur Schwarz oder Weiß (bzw. Volltonfarben) auf das Terminal und benötigt dafür eine hohe Auflösung, damit selbst noch kleinste Schriftzeichen scharf und sauber zu lesen sind.

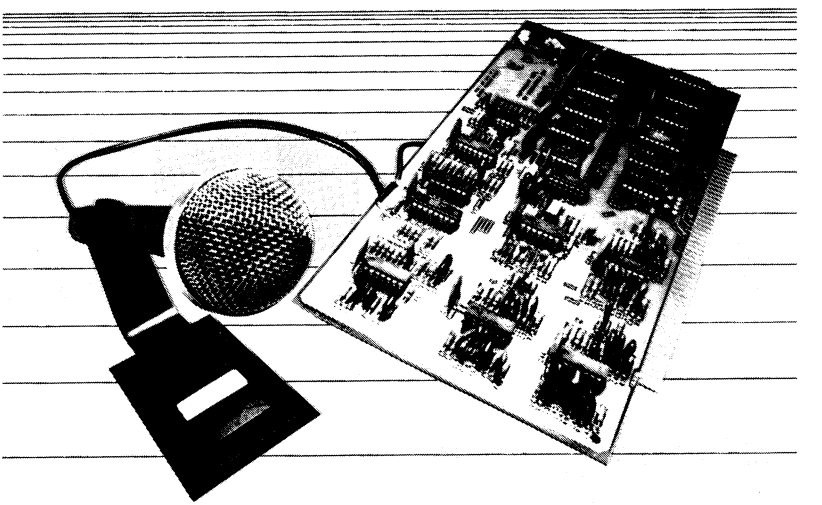
Wenn die Hersteller von Fernsehgeräten hochwertige Bildröhren mit der Auflösung, wie sie für Videoanzeigen-Terminals benötigt werden, bauen würden, dann könnte unser Fernsehgerät als Videoanzeige für einen Computer ohne Verlust an Anzeigequalität verwendet werden. Das Fernsehgerät würde jedoch wesentlich mehr kosten.

CRT **VDU**

Eine Videoanzeige wird manchmal als Kathodenstrahlröhre (Cathode Ray Tube = CRT) oder Videoanzeigeeinheit (Video Display Unit = VDU) bezeichnet. Eine Video-Anzeigeeinheit als VDU zu bezeichnen, erscheint logisch. Sie jedoch nur ein„CRT" zu nennen, sagt dem durchschnittlichen Anwender wenig. „CRT", d.h. Kathodenstrahlröhre, bezieht sich nur auf die in Fernsehgeräten und Videoanzeigen verwendeten Bildröhren. „Kathodenstrahlen" werden für das Schreiben auf dem Bildschirm verwendet. In nicht allzu ferner Zukunft werden die Kathodenstrahlröhren wahrscheinlich durch billigere und effizientere Technologien ersetzt. Aber wir dürfen nicht erwarten, daß sich die Terminologie mit der Technologie ändert. Computertechniker werden wahrscheinlich die Videoanzeigen auch in der Zukunft noch als CRTs bezeichnen, wenn Kathodenstrahlröhren nur mehr in technischen Museen zu finden sein werden.

DIE TASTATUR

Wir kehren zu unserem Computersystem zurück und sehen uns die Tastatur an. Derzeit müssen wir noch eine Tastatur verwenden, um neue Informationen in ein Computersystem einzugeben. In Zukunft werden wir die Tastatur durch ein Mikrofon ersetzen und neue Informationen einfach in den Computer sprechen.

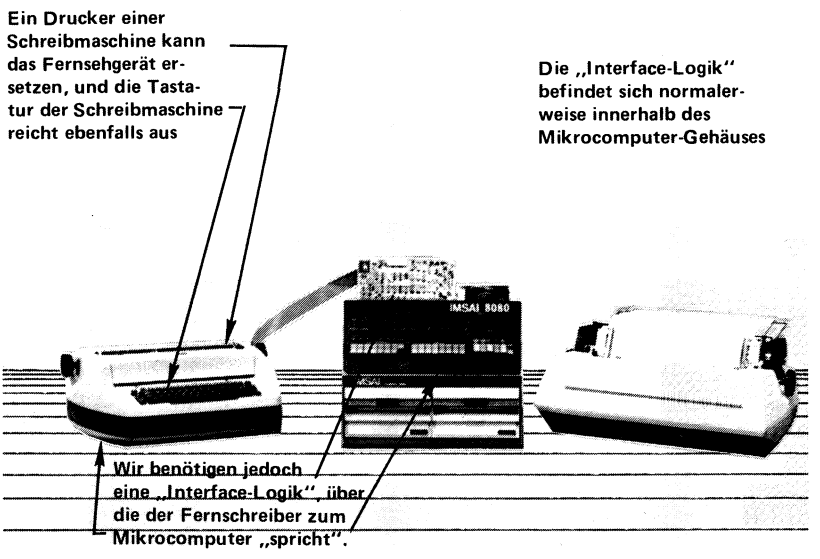


INTERFACE

Es gibt eine Unzahl von Tastaturen auf dem Markt, und jede beansprucht für sich Eigenschaften, die wir nützlich oder überflüssig finden können. **Wir können die Tastatur durch eine Schreibmaschine ersetzen, vorausgesetzt, daß uns jemand das nötige „Interface" für die Schreibmaschine baut.** Das Interface ist ein Gerät oder eine Schaltung, die zur Anpassung zweier Geräte dient, und im Deutschen manchmal als Schnittstellen-Schaltung bezeichnet wird.

Woraus besteht ein Interface?

Wenn wir unsere Schreibmaschine einfach neben den Mikrocomputer und das Fernsehgerät stellen, so wird das Drücken der Tasten der Schreibmaschine keinen Einfluß auf die Anzeige auf dem Bildschirm haben. Dazu muß die entsprechende Elektronik vorhanden sein, die die gedrückten Tasten abfragt und die entsprechenden Signale erzeugt, damit der Mikrocomputer und die Bildschirmanzeige wissen, daß eine Taste gedrückt wurde. Dies kann folgendermaßen dargestellt werden:

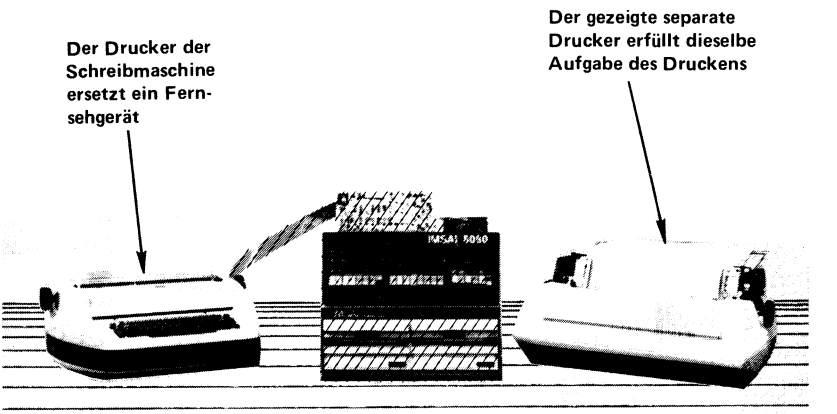


Wie erwartet, muß die Tastatur und die Videoanzeige, in der Tat jeder Teil eines Mikrocomputer-Systems, seine eigene elektronische Interface-Schaltung zum Mikrocomputer besitzen.

Wir können daher eine Tastatur und eine Videoanzeige durch ein Fernsehgerät und eine Schreibmaschine ersetzen.

DER DRUCKER

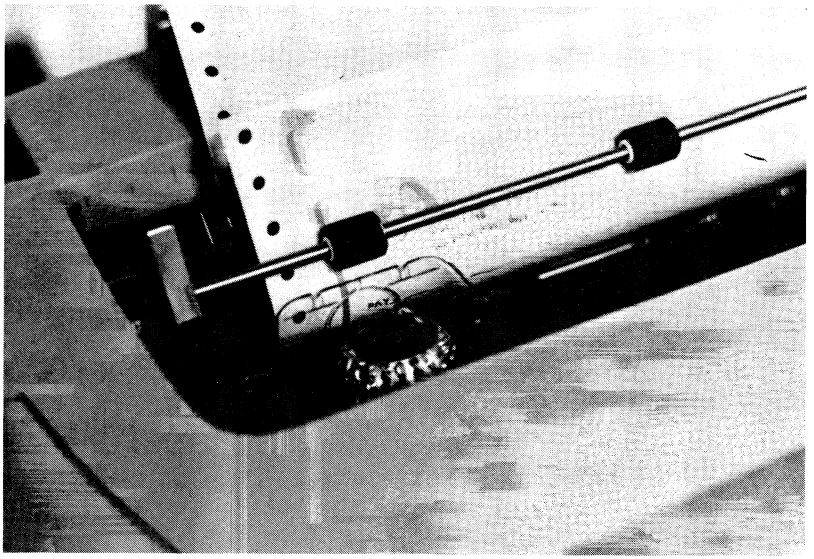
Die Schreibmaschine bietet einen Vorteil, den die Tastatur nicht besitzt. Die Schreibmaschine druckt, was eingetippt wird. In unserem gezeigten Mikrocomputer-System wurde diese Funktion von einem Drucker ausgeführt.



Wenn wir die Tastatur durch eine Schreibmaschine ersetzen, können wir auf den Drukker verzichten. Weshalb kauft dann überhaupt jemand Drucker? Die Antwort ist simpel, denn die Druckgeschwindigkeit ist wesentlich höher. Schreibmaschinen sind schwerfällige, mechanische Geräte, die höchstens 15 Zeichen pro Sekunde drucken können. Die langsamsten Drucker sind zweimal so schnell, während ein schneller Drucker 600 Zeilen pro Minute drucken kann. Ein extrem schneller Drucker verarbeitet sogar Tausende von Zeilen pro Minute.

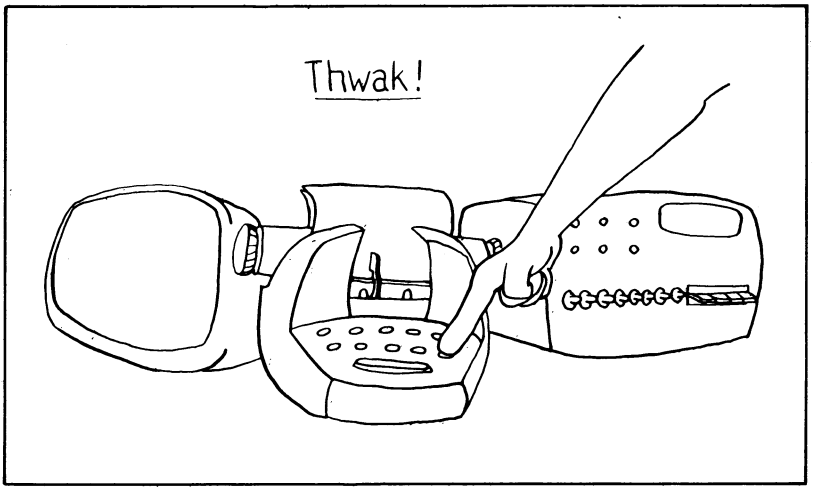
Sind 15 Zeichen pro Sekunde nicht schnell genug? Natürlich kann niemand so schnell schreiben, aber erinnern wir uns daran, der Drucker wird nicht nur das ausdrucken was wir eintippen. Betrachten wir ein Lohnverrechnungsprogramm. Der Mikrocomputer berechnet die Auszahlungsdaten und druckt dann die entsprechenden Auszahlungsschecks ohne weitere menschliche Hilfe aus.

Nach Eingabe der Auszahlungsdaten über die Tastatur müssen wir den Zahlungsscheck in die Schreibmaschine einführen und korrekt justieren:

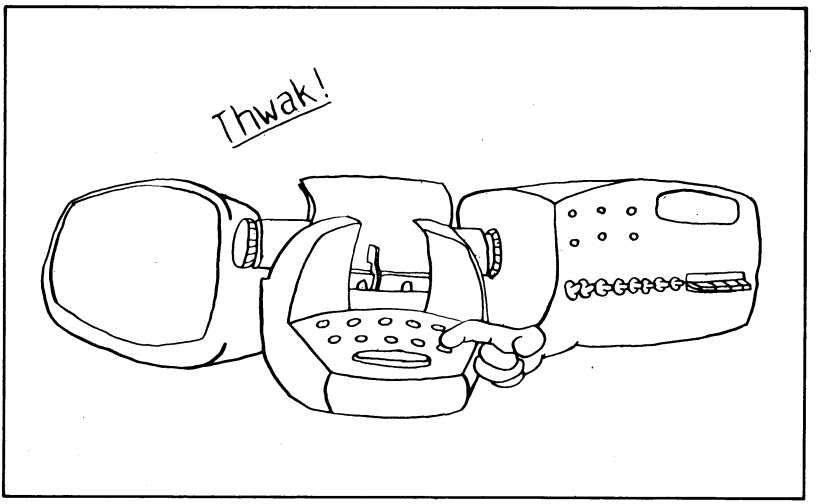


Wenn der Mikrocomputer die Berechnung beendet hat, wird er die Zahlungsschecks ausschreiben, während wir eine Kaffeepause (oder ein Nickerchen) einlegen können. Hier werden wir feststellen, daß 15 Zeichen pro Sekunde eine schrecklich langsame Druckgeschwindigkeit sind. Wenn wir eine Lohnliste für 50 Angestellte haben, ist das ganze Mikrocomputer-System 45 Minuten lang nur mit dem Drucken der Auszahlungsschecks beschäftigt. Wenn wir einfach einen schnelleren Drucker verwenden, können wir die Druckzeit wesentlich verringern und das Mikrocomputer-System früher wieder für andere Zwecke verwenden. Wenn beispielsweise ein Drucker mit 150 Zeichen pro Sekunde arbeitet, und das ist nichts Ungewöhnliches, wird er das Drucken der Auszahlungsschecks in nur 4 1/2 Minuten erledigen.

Die Druckgeschwindigkeit ist jedoch nicht der einzige Grund, weshalb man zweckmäßigerweise den Drucker von der Tastatur trennt. Wenn die beiden verbunden sind, so wie es in einer Schreibmaschine der Fall ist, wird bei jeder Betätigung einer Taste ein Zeichen gedruckt:

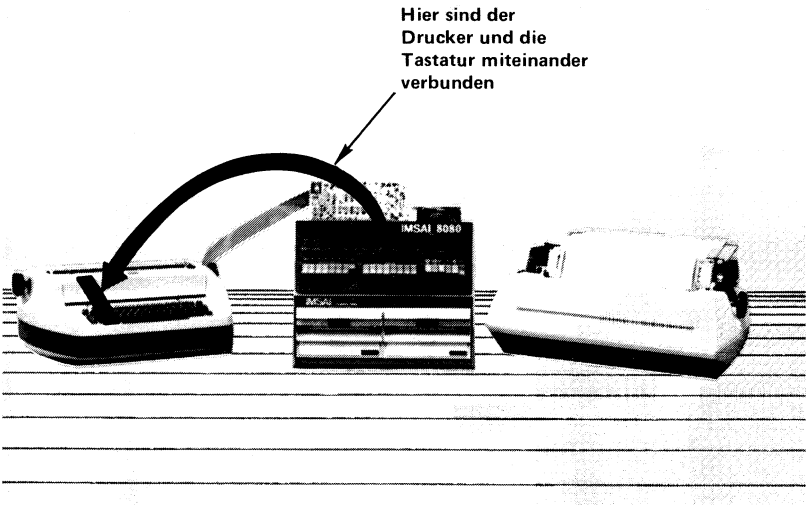


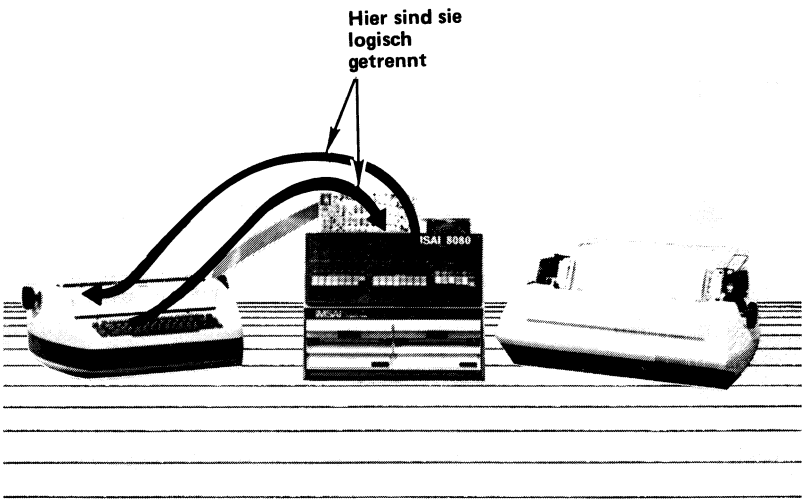
Jedesmal, wenn der Mikrocomputer ein Zeichen druckt, wird er eine Taste betätigen.



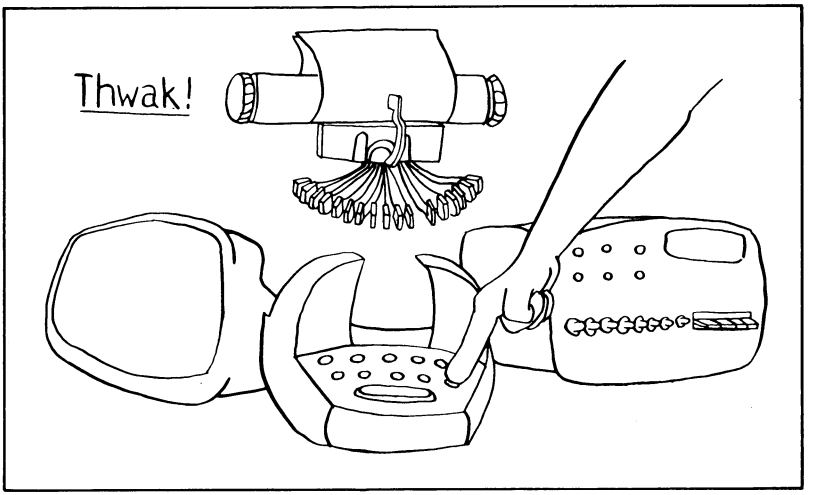
Das bedeutet, daß wir die Tastatur nicht ohne gleichzeitigem Drucken verwenden können und umgekehrt, wir können nicht drucken, ohne daß wir die Tastatur verwenden.

Wir trennen nunmehr den Drucker von der Tastatur. Hierbei geschieht grundsätzlich folgendes:

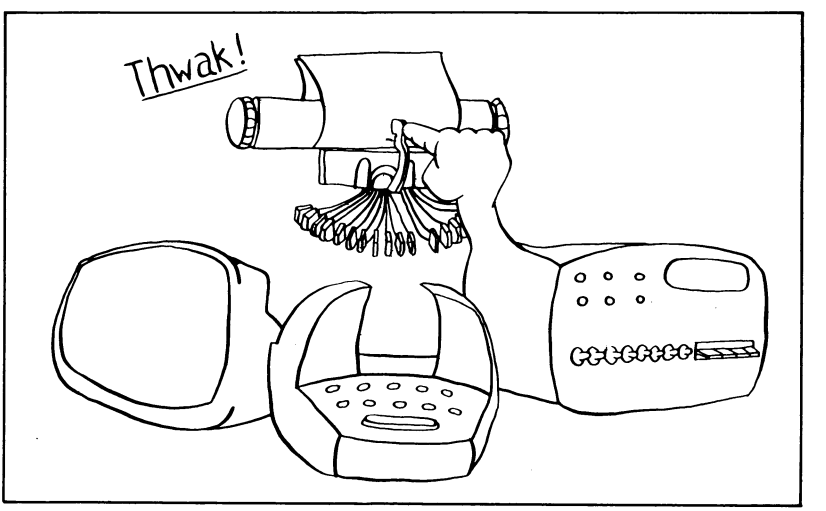




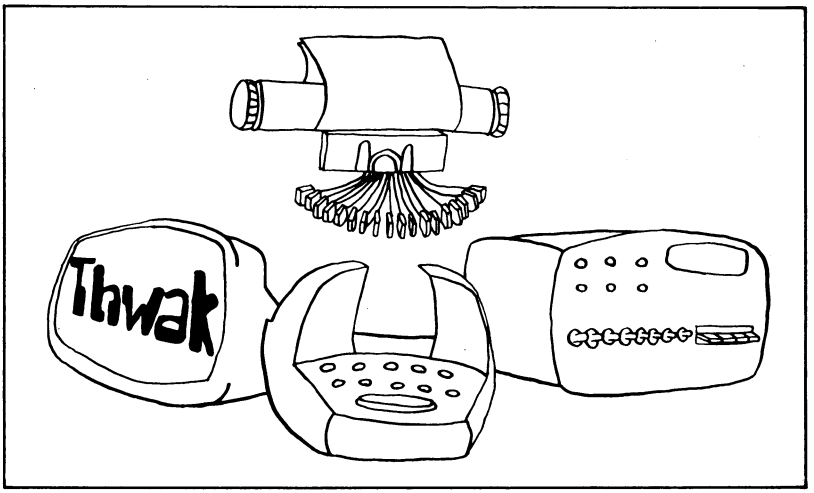
Wenn die Tastatur und der Drucker mechanisch gekoppelt sind, wird bei Betätigung einer Taste automatisch ein Zeichen gedruckt:



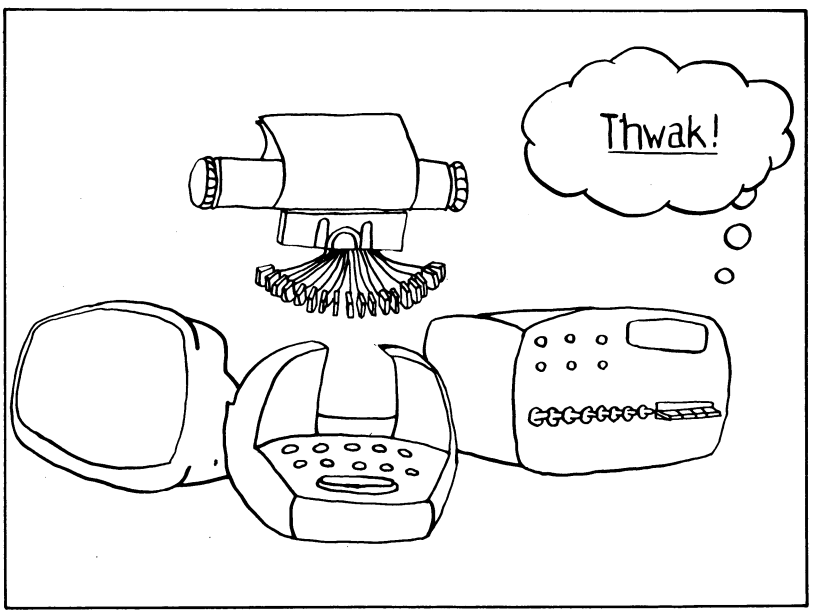
Sind Tastatur und Drucker voneinander getrennt, muß der Mikrocomputer so programmiert werden, daß er das Drucken des eingetippten Zeichens bewirkt, andernfalls erfolgt kein Druck:



Der Mikrocomputer kann auch das Zeichen zurück zur Videoanzeige geben:



Oder der Mikrocomputer kann alle Zeichen für sich behalten:



ECHO-BETRIEBSART

Wenn der Mikrocomputer ein Zeichen entweder zum Drucker oder zur Anzeige zurückgibt, so sagt man er arbeite in der „Echo-Betriebsart".
Wir haben eben ein sehr wichtiges Konzept gezeigt: Die unabhängigen Steuerungen miteinander verbundener Geräte durch den Mikrocomputer. Die Tatsache, daß eine Tastatur und eine Videoanzeige als eine einzige Einheit zusammengebaut ist, bedeutet noch nicht, daß jedes Mal, wenn wir eine Taste betätigen, das entsprechende Zeichen dargestellt werden muß. Wenn ein „Echo" auftritt, so wird dies durch den Mikrocomputer bewirkt.
Sind bei einer Schreibmaschine die Tastatur und die Typenelemente mechanisch miteinander verbunden, so bedeutet dies, daß bei jeder Betätigung einer Taste die Schreibmaschine auch ein Zeichen druckt, ob wir es wollen oder nicht. Daher kann der Mikrocomputer keinen Echobetrieb mit einer Schreibmaschine ausführen, und dies ist in einem Mikrocomputer-System nicht erwünscht.

BAUTEILE, DIE EINE MENGE INFORMATIONEN SPEICHERN

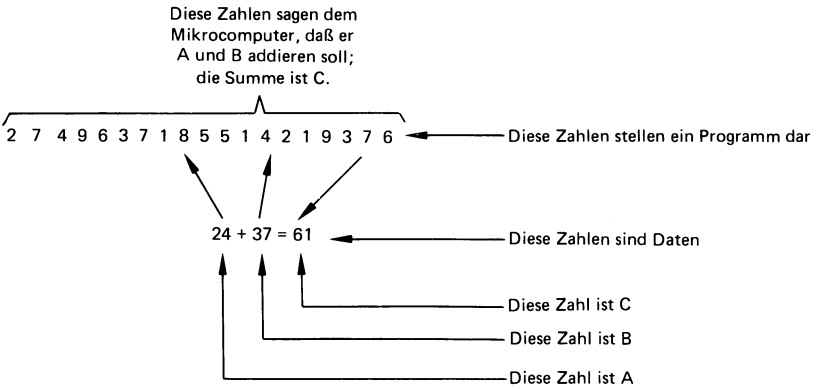
PROGRAMME

Wir haben nun bereits mehrfach über die Dinge gesprochen, die ein Mikrocomputer ausführen muß. Aber wie erledigt der Mikrocomputer diese Dinge? Wir stoßen hierbei zuerst auf den Begriff „Programmieren". Später werden wir den Begriff der Programmierung grundsätzlich behandeln. Im Band „Einführung in die Mikrocomputer-Technik" wird die Programmierung detailliert beschrieben. Im Augenblick interessiert uns nur die Untersuchung der einzelnen Teile eines Mikrocomputer-Systems. Daher besprechen wir nicht so sehr, „wie" Programme geschrieben werden, sondern „was" wir zum Schreiben von Programmen benötigen.
Jedes Computerprogramm besteht einfach aus einer Folge oder Sequenz von Zahlen. Es ist nichts besonderes an diesen Zahlen, sie sind ganz einfach, klare und deutliche Zahlen. **Eine in einem Mikrocomputer gespeicherte Zahl kann einen Programmschritt oder „Daten" darstellen.** Es ist nur eine Frage der Interpretation. Wenn der Mikrocomputer sich eine Zahl holt, wenn er einen Programmschritt benötigt, dann nimmt er einfach an, daß diese geholte Zahl ein Programmschritt ist. Wenn der Mikrocomputer jedoch erwartet, eine Zahl zu empfangen (zum Beispiel, ein oder zwei zu addierende Zahlen), dann nimmt er an, daß die ankommenden Zahlen auch „Daten" sind.

DATEN

Das Wort „Daten" dient zur Beschreibung von Zahlen, wenn diese als Zahlen oder Buchstaben des Alphabets zu interpretieren sind, im Gegensatz zu Zahlen, die als Programmschritte interpretiert werden. Wenn wir beispielsweise zwei Zahlen addieren, dann sind die beiden zu addierenden Zahlen und die zu bildende Antwort alles „Daten". Die Befehle für einen Mikrocomputer, die die Addition der beiden Zahlen

bewirken, stellen ein Programm dar. Das Programm selbst besteht aus einer Folge von Zahlen, aber diese Zahlen stellen keine Daten dar, sondern Programmschritte. Dies kann folgendermaßen gezeigt werden:



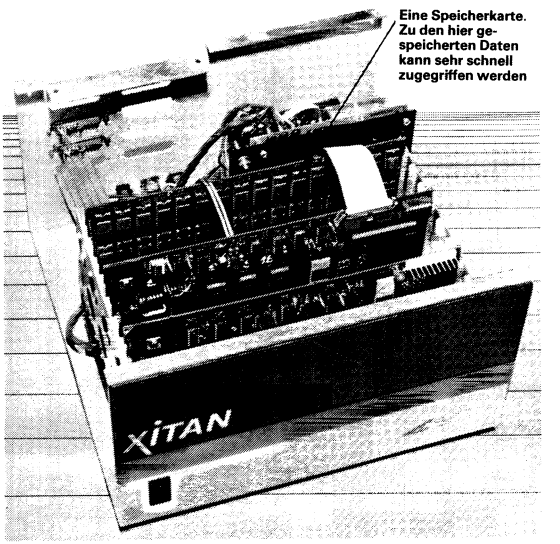
Es ist gar nichts Ungewöhnliches, wenn Zahlen innerhalb eines Mikrocomputers entweder Programmschritte oder Daten darstellen. Wir machen alle etwas Ähnliches tagtäglich. Eine Zahl auf einem Stück Papier kann Teil einer Sozialversicherungsnummer sein, sie kann die Nummer eines Bankschecks, der Geldbetrag auf einem Scheck oder der Geldbetrag auf einer Rechnung sein. Nur durch die Betrachtung und Interpretation können wir sagen, welcher Betrag was darstellt. Wenn wir unseren Bankauszug überprüfen und hierbei zufällig die Kontonummer anstelle eines Scheckbetrages lesen, so werden wir ein sehr merkwürdiges (und offensichtlich falsches) Ergebnis bekommen. Ein derartiger Fehler ist jedoch grundsätzlich möglich. Ähnlich könnte auch ein Mikrocomputer versehentlich Zahlen, die eigentlich Daten darstellen, lesen und diese als einen Programmschritt interpretieren. Nur werden die Ergebnisse sehr merkwürdig sein.
Zahlen stellen nur einen sehr kleinen Teil unserer Welt dar. Sie sind jedoch die ganze Welt der Mikrocomputer. Auch eine kleine Aufgabe, wenn sie für einen Mikrocomputer als Programm definiert wird, führt zur Bildung von Hunderten von Zahlen. Größere Aufgaben für unseren Mikrocomputer können zu Programmen mit Tausenden, ja sogar Millionen von Zahlen führen.
Dies ergibt ein Problem.

SPEICHER

MIKROSEKUNDE

Der Mikrocomputer führt jede spezifizierte Aufgabe durch Abarbeiten eines entsprechend spezifizierten Programmes aus. Das Programm besteht aus einer Folge von Schritten, oder Befehlen. Jeder Befehl wird durch eine eindeutige Zahl identifiziert. Ein Mikrocomputer kann jedoch Hunderte oder Tausende von Programmschritten oder Befehlen in einer einzigen Sekunde durchlaufen. **Tatsächlich führt ein typischer**

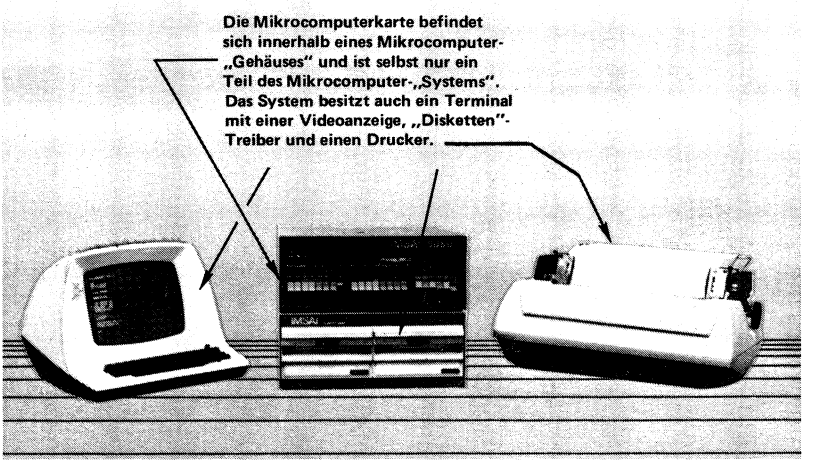
Mikrocomputer einen einzigen Befehl in einem Zeitabstand aus, der von einem Millionstel einer Sekunde bis zu einem Zehnmillionstel einer Sekunde dauert. (Wir bezeichnen ein Millionstel einer Sekunde als eine Mikrosekunde.) Wenn der Mikrocomputer nun imstande ist, einen Befehl innerhalb weniger Mikrosekunden auszuführen, dann muß natürlich die Zahl, die den Befehl darstellt, in irgendeiner Form gespeichert und rasch verfügbar sein. Der Mikrocomputer muß imstande sein, sich diese Zahl, die den Befehl darstellt, wesentlich rascher zu holen, als in den wenigen Mikrosekunden, die er zur Ausführung des ganzen Mikrobefehls besitzt. Ein Mikrocomputer kann sich einen Befehl aus einem Speicher mit einem raschen Zugriff in einer halben Mikrosekunde oder weniger holen. Diese Art der Speicher mit einem schnellen Zugriff sind etwas aufwendig. **Der Mikrocomputer besitzt daher einen relativ kleinen, schnellen Speicher, der gewöhnlich im Mikrocomputergehäuse enthalten ist:**



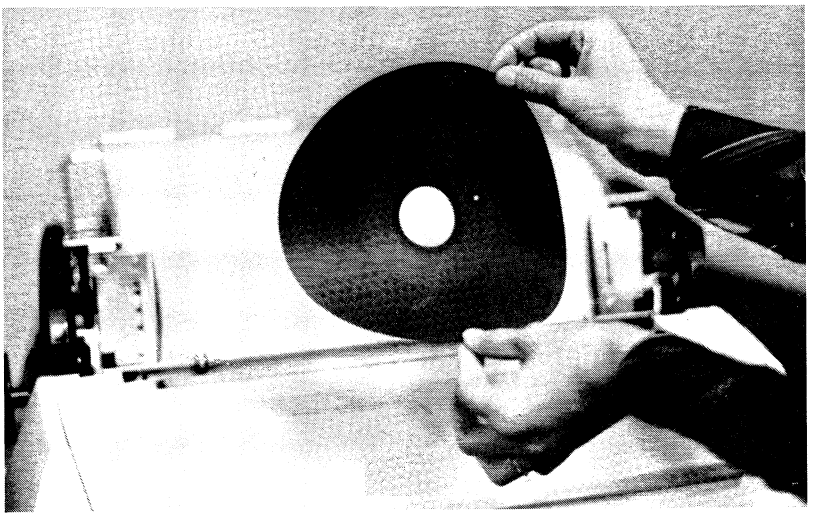
Programmschritte und Daten, die der Mikrocomputer laufend benötigt, müssen in diesem Speicher mit schnellem Zugriff enthalten sein.

FLOPPY-DISK-GERÄTE

Programme und Daten, die der Mikrocomputer momentan nicht benötigt, werden in irgendeinem langsameren (und billigeren) Massenspeichergerät enthalten sein. In Bild 1.2 wird ein derartiger Massenspeicher in Form eines „Floppy-Disk-Systems" gezeigt.



Es gibt gewisse grundlegende Ähnlichkeiten zwischen einem Floppy-Disk-System und einem Plattenspieler. Das Floppy-Disk-System speichert seine Informationen auf Floppy-Disk, die ihren Namen wegen ihrer Weichheit und Biegsamkeit erhalten haben (floppy = biegsam, schlapp) und werden manchmal auch als Disketten bezeichnet:



Da diese Platten weich sind, werden sie in einer entsprechenden Hülle aus steifem Karton aufbewahrt:

MINI-FLOPPY

Es gibt zwei Größen von Floppy-Disks: Eine Standardgröße mit ca. 203 mm (8 Zoll) und „Mini-Floppies" mit einem Durchmesser von ca. 133 mm (5 1/4").
Während eine Schallplatte eine Oberfläche mit Rillen besitzt, hat ein Floppy-Disk eine glatte, magnetische Oberfläche. Auf dieser glatten, magnetischen Oberfläche werden die Informationen als Folgen von magnetischen Impulsen gespeichert.

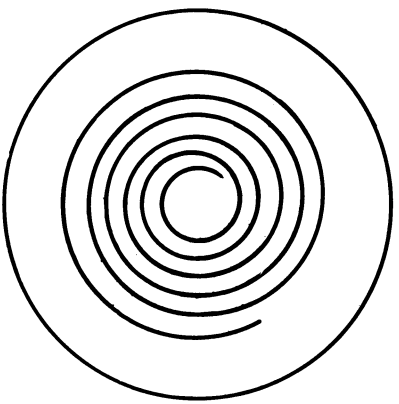
SPUREN

Die magnetischen Impulse werden entlang von „Spuren" auf der Oberfläche der Disketten aufgezeichnet. Im Gegensatz hierzu wird Musik auf der Oberfläche einer Schallplatte innerhalb einer durchlaufenden Rille aufgezeichnet. Die Führung des Tonkopfes erfolgt durch die Bewegung der Nadel in der Rille. Da die Oberfläche einer Floppy-Disk jedoch vollkommen glatt ist, gibt es hier keine Rillen. Die Spur wird hier deshalb zu einer imaginären Linie, entlang der die magnetischen Impulse liegen. Die Informationen werden in diese Spur durch einen magnetischen Schreib-/Lesekopf eingeschrieben oder ausgelesen, der einem Tonabnehmerarm eines Plattenspielers ähnelt. Dieser Schreib-/Lesekopf für die Floppy-Disk-Einheit besitzt natürlich keine Nadel für eine entsprechende Führung. Zum Aufzeichnen oder Abtasten der magnetischen Im-

pulse auf der Oberfläche der Diskette wird dagegen ein winziger Schreib-/Lesekopf, ähnlich dem eines Tonbandgerätes, verwendet.

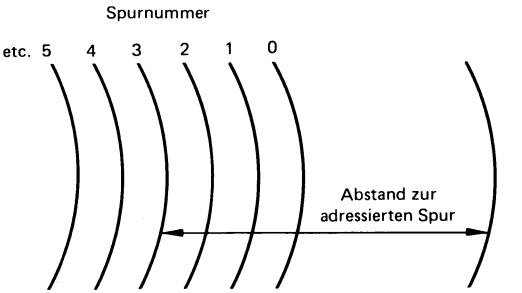
Für das weitere Verständnis des vorliegenden Stoffes ist es für uns zunächst unwesentlich zu wissen, wie die Informationen auf einer Floppy-Disk gespeichert werden. Die anschließende Besprechung soll daher auch nur eine allgemeine Beschreibung der Konzepte darstellen. Die meisten Anwender von Mikrocomputern werden sich auch kaum mit diesen Informationen befassen, so wie es für die meisten Musikliebhaber auch unwesentlich ist, wie die Musik auf einem Band, Kassette oder Schallplatte aufgezeichnet wird. Wir wissen, daß beim Abspielen einer Schallplatte Musik entsteht. Ähnlich werden beim Abspielen einer Diskette Zahlen erzeugt. Wir können auch Zahlen in eine Floppy-Disk schreiben, so wie wir Musik auf einem Magnetband oder einer Kassette aufzeichnen.

Es wäre denkbar, die Informationen auf die Oberfläche einer Diskette zu schreiben, so wie Musik auf der Oberfläche einer Schallplatte aufgezeichnet wird, indem man einfach eine kontinuierliche Rille (oder in diesem Fall eine kontinuierliche Spur) verwendet, die sich spiralenförmig zum Zentrum der Floppy-Disk bewegt:



Das Problem bei diesem Aufzeichnungsschema liegt darin, daß es sehr schwierig ist die aufgezeichnete Information genau zu verfolgen. Wir kennen dies, wenn wir versuchen ein spezielles Wort in einem Lied auf einer Schallplatte zu finden. Wenn wir lange genug probieren, werden wir wahrscheinlich diese Aufgabe lösen können. Für die Elektronik muß dies jedoch reproduzierbar möglich sein. Wir ersetzen daher eine einzige kontinuierliche spiralförmige Spur durch eine große Anzahl konzentrischer Spuren.

Wir können nunmehr eine bestimmte Spur durch ihre Spurennummer auswählen, die dadurch adressiert werden kann, daß der exakte Abstand der Spur vom Rand oder Zentrum der Floppy-Disk festgelegt wird:



Die einzelnen Hersteller verwenden eine unterschiedliche Anzahl von Spuren auf der Oberfläche einer Diskette. Der einzige bestehende Standard verwendet auf Disketten mit 200 mm Durchmesser 77 konzentrische Spuren (numeriert von 0 bis 76). Bei manchen Disketten wird die Information nur auf einer Seite der Platte gespeichert, während andere beide Seiten verwenden.

Man kann etwa 250000 Zeichen auf einer Oberfläche einer 200-mm-Floppy-Disk unterbringen. Es ist jedoch nicht so einfach, diese Informationen auf der Basis von Spur-Nummern aufzuteilen. Zunächst ist einmal die Länge der Spuren am Rand größer als in der Mitte, was bedeutet, daß jede Spur eine unterschiedliche Anzahl von Informationen aufnehmen kann. Zweitens kann die Menge der auf einer Spur gespeicherten Informationen sehr groß sein. Trotzdem würde sie die kleinste Einheit einer adressierbaren Information auf der Oberfläche einer Floppy-Disk darstellen. Wenn wir die Oberfläche einer Diskette nur über Spurnummern adressieren, dann müssen wir die Information einer ganzen Spur lesen, oder wir müssen Informationen auf die ganze Spur schreiben.

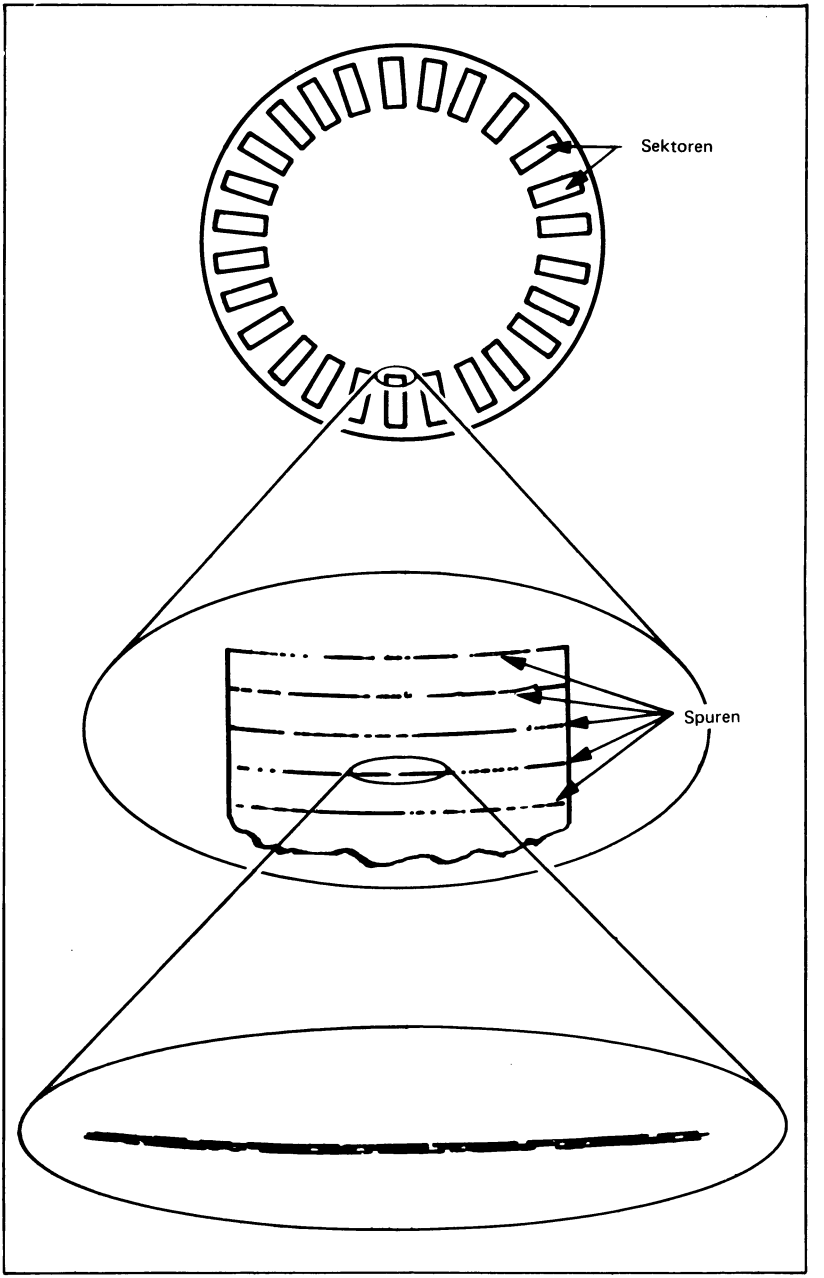
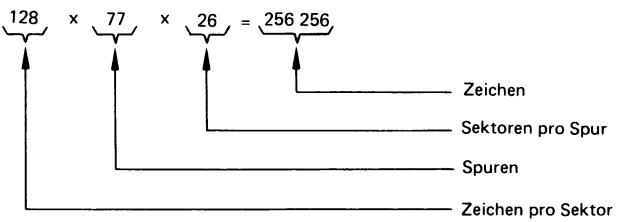


Bild 1.2 Die beschriebene Oberfläche einer Diskette

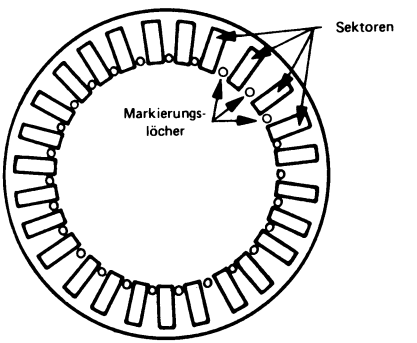
SEKTOR

Wir lösen diese beiden Probleme, indem wir jede Spur in „Sektoren" aufteilen, und die gleiche Anzahl von Informationen auf jede Spur speichern, unabhängig davon wie nahe sich die Spur am Zentrum oder Umfang der Oberfläche der Floppy-Disk befindet. Wie wir aus Bild 1-2 ersehen können, wird um so mehr von einer Spur verschwendet, je weiter wir uns vom Zentrum zum Umfang der Diskette bewegen. Nun können wir jedoch eine Information auf der Oberfläche der Floppy-Disk durch seine Spurnummer und durch die Nummer des Sektors innerhalb der Spur identifizieren. In Bild 1.2 sind 26 Sektoren (numeriert von 1 bis 26) gezeigt. Dies ist eine häufig gebrauchte Anzahl von Sektoren. Gewöhnlich werden 128 Zeichen innerhalb jedes Sektors gespeichert. Wir können daher die gesamte Speicherkapazität der Oberfläche einer Floppy-Disk wie folgt berechnen:



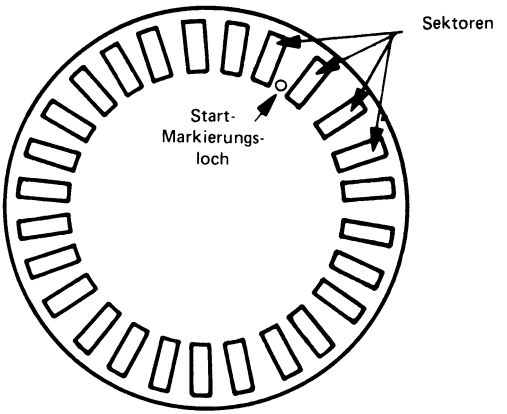
DISKETTEN MIT DOPPELTER DICHTE
HARD SECTORED DISKS

Manche Hersteller speichern 256 Zeichen in jeden Sektor einer Spur. Sie werden Disketten mit doppelter Dichte („double density" floppy disks) genannt. Disketten mit doppelter Speicherdichte verwenden die gleiche Sektorgröße wie gewöhnliche Floppy-Disk, sie zwängen jedoch mehr Informationen in jeden Sektor.
In Disketten werden ein oder mehrere Löcher gestanzt, um dem Antriebsmechanismus das Auffinden der Sektoren zu erleichtern. Bei manchen Disketten ist ein Loch zwischen jedem Sektor eingestanzt. Sie werden als „Hard sectored disks" bezeichnet und sehen etwa folgendermaßen aus:



SOFT SECTORED DISK

Platten mit nur einem eingestanzten Loch nennt man „Soft sectored disks". Dies sieht folgendermaßen aus:



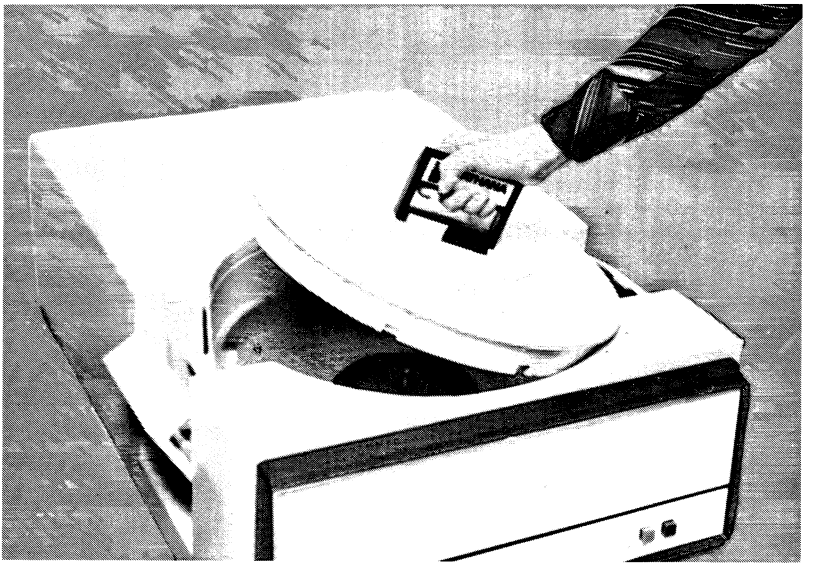
DISKETTEN-FORMATIERUNG

Wenn wir eine völlig neue Diskette kaufen, so ist ihre Oberfläche in der Tat völlig „neu". In dieser Form können wir nicht in die Diskette einschreiben. Vor dem Einschreiben müssen wir einen Schritt durchführen, der als „Formatieren" bezeichnet wird. Während dieses Formatier-Schrittes legt der Antrieb der Diskette bestimmte Sektoren und Spuren unter Verwendung eines entsprechenden magnetischen Codes fest. Dies ist ein automatischer Vorgang, den wir unter Verwendung unseres Mikrocomputer-Systems ausführen. Wir dürfen jedoch diesen Schritt nicht vergessen. Sobald wir eine leere Diskette formatiert haben, ist diese für die Verwendung bereit. Wir können Informationen auf die formatierte Diskette schreiben und die eingeschriebenen Informationen zurücklesen.

Der besondere Vorteil der Disketten liegt darin, daß man sich eine entsprechende Sammlung anlegen kann, so wie man etwa eine Sammlung von Schallplatten besitzen kann. Wir können auch bereits beschriebene Disketten kaufen, die komplett mit Programmen erhältlich sind, oder wir verwenden leere Disketten und zeichnen unser eigenes Programm auf die Disketten auf. Dann bewahren wir die Floppy-Disks mit den Aufzeichnungen in unserer Bibliothek auf. Wir können dann zu jeder Zeit eine Diskette aus unserer Bibliothek entnehmen und das gekaufte oder eigene Programm ablaufen lassen.

GERÄTE MIT STARREN PLATTEN

Disketten sind nicht das einzige Hilfsmittel zur Speicherung von Programmen und Daten. Große Computersysteme verwenden große starre Platten:



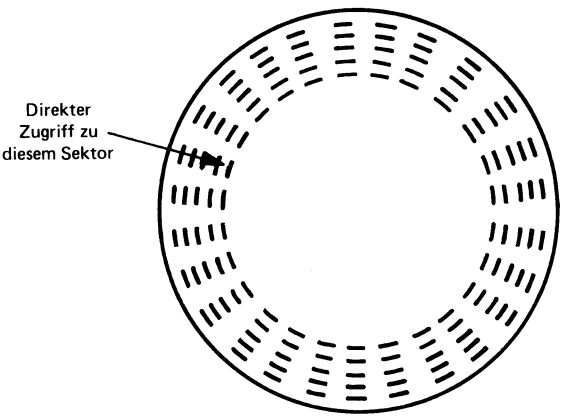
Große Systeme mit starren Platten bieten eine Vielzahl von Möglichkeiten auch in den Abmessungen. Sie speichern alle Informationen auf einer wesentlich größeren Platte, die aber nicht biegsam ist.

Mit starren Platten lassen sich wesentlich mehr Informationen als mit Disketten speichern, sie sind aber teurer.
Tatsächlich sind jedoch starre Platten billiger, wenn man die Kosten pro gespeicherter Informationseinheit betrachtet. Mit einem kleinen Mikrocomputer-System benötigen wir aber einfach keine derart große Speicherkapazität, wie sie diese starren Platteneinheiten besitzen. Manche großen starren Platteneinheiten sind außerdem noch imstande, Informationen rascher zu transferieren, als sie ein Mikrocomputer verarbeiten kann.

PLATTENZUGRIFF

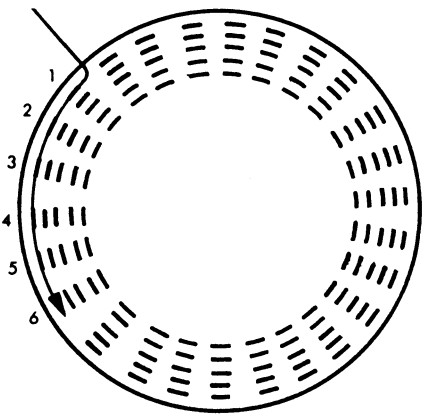
WAHLFREIER ZUGRIFF

Disketten-Einheiten und Einheiten mit starren Platten werden beide als Massenspeichergeräte mit „wahlfreiem Zugriff" (random access) bezeichnet. Der Name „wahlfreier Zugriff" bedeutet, daß wir direkt zu jedem Sektor jeder Spur zum Lesen oder Schreiben von Informationen gelangen können:

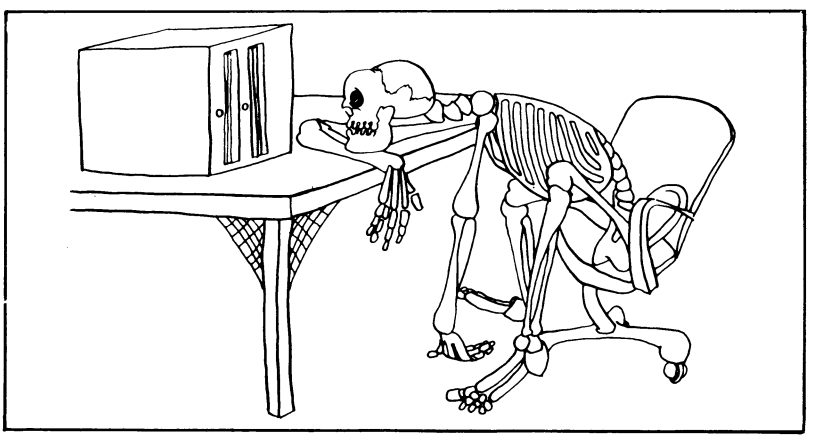


Nehmen wir beispielsweise an, daß wir den Inhalt von Spur 12 in Sektor 8 lesen wollen. Der Lesekopf der Diskette gelangt nun direkt zu diesem Sektor, ohne zuerst zu den Spuren 1 bis 11 und den Sektoren 1 bis 7 der Spur 12 gehen zu müssen.
Wahlfreier Zugriff ist eine außerordentlich nützliche Eigenschaft in jedem Massenspeichergerät. Die Möglichkeit des direkten Zugriffs zu jedem Sektor zum Lesen oder Schreiben erhöht die Geschwindigkeit der Daten-Zugriffsoperationen außerordent-

lich. Wenn wir angenommen die Sektoren sequentiell lesen müßten, würden wir mit den ersten Sektoren kaum Probleme haben:



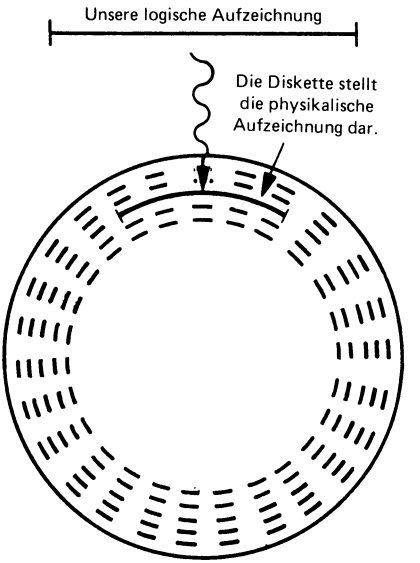
Wir würden aber außerordentlich viel Zeit benötigen, bis wir die letzten Sektoren auf der Oberfläche der Diskette erreichen.



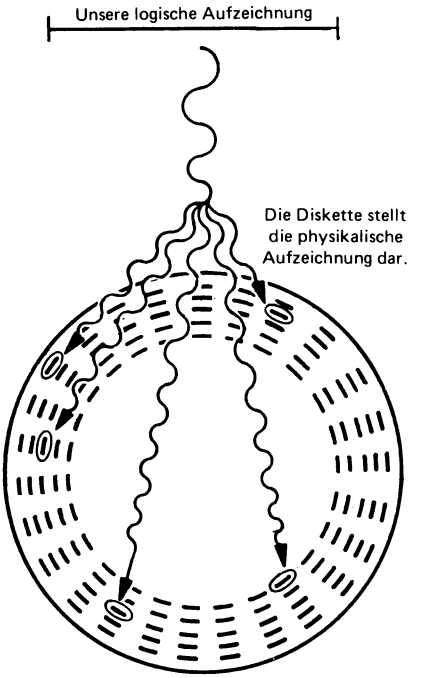
SEKTOREN-VERKETTUNG

Die Möglichkeit, direkt zu jedem Sektor auf der Oberfläche einer Diskette zu gelangen, hat einen anderen weniger offensichtlicheren Vorteil. Was ist, wenn wir einen Block von Informationen haben, der zu groß ist, um in einen einzelnen Sektor zu passen? Nehmen wir einmal an, daß der Informationsblock so groß ist, daß er in fünf Sektoren gespeichert werden muß.

Wir können beispielsweise ein Mikrocomputer-System zum Schreiben „formeller" Briefe verwenden. Wir könnten fünfzig Standardabschnitte auf die Diskette speichern und dann eine Vielzahl von Briefen schaffen, indem wir einfach ausgewählte Abschnitte aneinander hängen. Auf diese Weise werden zum Beispiel die meisten Werbebriefe geschrieben. Jeder Abschnitt wird zu einer Informationseinheit, die in unserem Beispiel in fünf Disketten-Sektoren gespeichert wird. Wahrscheinlich würden wir zunächst denken, daß wir fünf aufeinanderfolgende Sektoren verwenden müssen:



Da wir jedoch auf die Sektoren willkürlich zugreifen können, besteht kein Unterschied darin, ob die fünf Sektoren aneinander hängen, wie oben gezeigt, oder ob sie unregelmäßig über die Oberfläche der Diskette verteilt sind:



VERKETTETE SEKTOREN

Wenn wir eine einzelne Informationseinheit in mehr als einem Sektor speichern, so sagt man, daß die Sektoren „verkettet" sind.

LOGISCHE UND PHYSIKALISCHE AUFZEICHNUNGEN

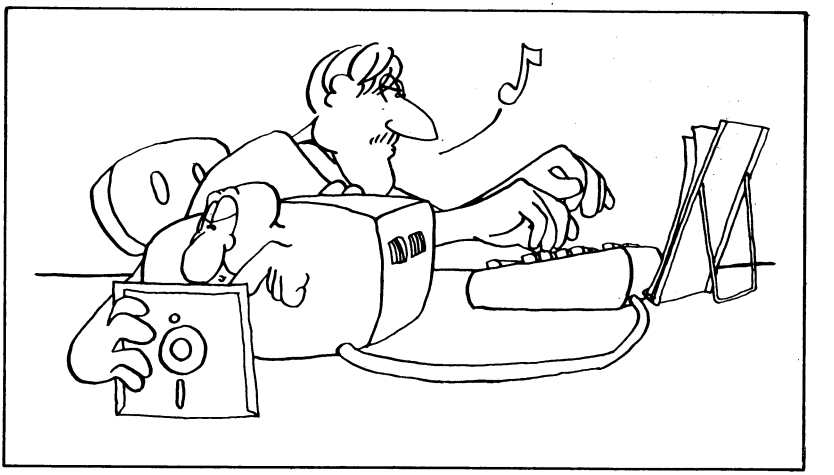
Dies ist ein geeigneter Moment, um ein wesentliches, grundlegendes Computerkonzept einzuführen: Die Beziehung zwischen „logischen" Ideen und „physikalischer" Wirklichkeit.

AUFZEICHNUNG

Betrachten wir einen Abschnitt, der ein Teil eines Werbebriefes ist. Dieser Abschnitt ist in fünf Sektoren eingeschrieben. Bezeichnen wir diesen Abschnitt als eine „Aufzeichnung".

PHYSIKALISCHE AUFZEICHNUNG
LOGISCHE AUFZEICHNUNG

Auf der Oberfläche einer Diskette wird diese Aufzeichnung eine „physikalische Aufzeichnung", die aus fünf verketteten Sektoren besteht, die benachbart oder auch nicht sein können. Aber warum sollten wir uns den Kopf über Sektoren zerbrechen? Es ist wesentlich einfacher, wenn wir Informationen als „Abschnitte" behandeln oder wie wir Informationen lesen wollen. Wir möchten in Wirklichkeit zu den Informationen als „logische Aufzeichnungen" zugreifen, die (hoffentlich) nichts mit Sektoren oder Spuren zu tun haben. Und das ist bei einer guten Disketteneinheit der Fall. Das Disketten-Gerät kümmert sich um das Auffinden der Sektoren und ihrer Verkettung, wenn eine Aufzeichnung in mehr als einem Sektor gespeichert ist. Wir brauchen uns nicht darum zu kümmern, wieviel Sektoren für unsere Aufzeichnung erforderlich sind, oder wo sich die Sektoren befinden. Wir müssen nicht einmal wissen, daß Spuren und Sektoren existieren. **Wir haben es mit der „logischen Idee" eines Abschnittes zu tun, und der Mikrocomputer kümmert sich um die „physikalische Realität" der Sektoren und Spuren.**

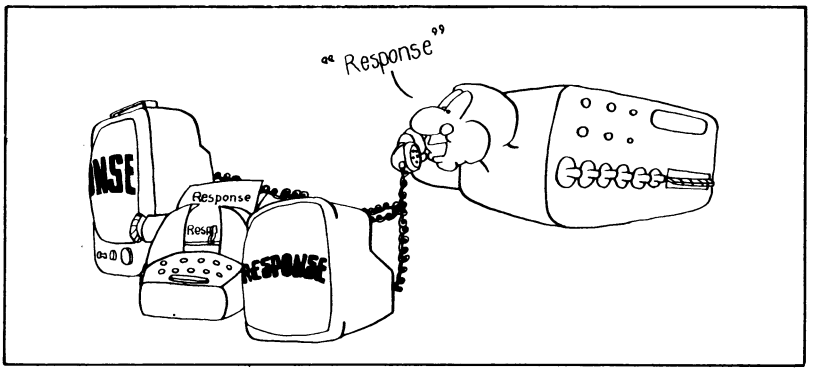


Bei der Verwendung eines gut durchdachten Mikrocomputer-Systems brauchen wir nur an logische Aufzeichnungen zu denken, und können Sektoren, Spuren und derartige Komplikationen ignorieren. Trotzdem ist es gut zu wissen, daß es Sektoren, Spuren und einen wahlfreien Zugriff gibt und ein Disketten-Gerät ein effizientes und schnelles Massenspeicher-Gerät ist.

Das Konzept von „logisch" gegen „physikalisch" kann auch für andere Begriffe als Informationen und Aufzeichnungen verwendet werden.

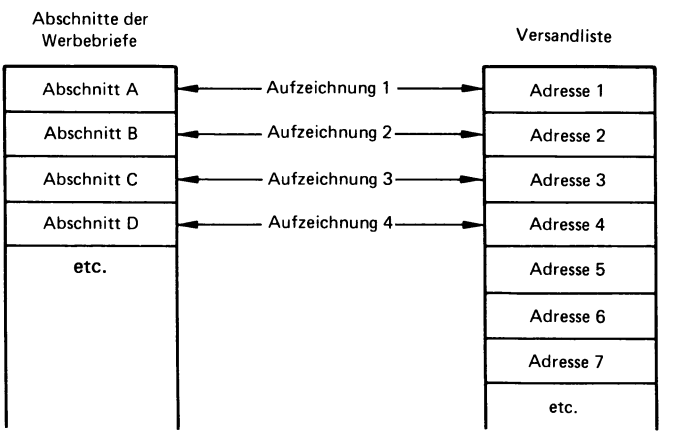
„Logische Einheiten" und „physikalische Einheiten" treten in nahezu jedem Teil eines Computersystems auf. Allgemein gesprochen ist eine „logische Einheit" eine Information, eine Idee oder eine Operation, die von einem Menschen verwendet wird.

Eine „physikalische Einheit" ist die tatsächliche physikalische Ausführung, die physikalische Realität hinter der Idee, Information oder Funktion. In unserer früheren Diskussion haben wir beispielsweise gesehen, wie eine Videoanzeige entweder durch ein Fernsehgerät oder den Drucker einer Schreibmaschine ersetzt werden konnte. In diesem Fall sind alle drei — Video-Anzeige, das Fernsehgerät und der Drucker der Schreibmaschine — die reellen, physikalischen Geräte, die die Idee einer logischen Einheit einer Computerantwort oder Reaktion darstellen.



AUFZEICHNUNGEN UND DATEIEN

Angenommen, wir haben eine Anzahl von Abschnitten von Werbebriefen, die alle auf der Oberfläche einer Diskette als logische Aufzeichnung untergebracht sind. Betrachten wir ferner eine Liste von Namen und Adressen. Jeder Name und jede Adresse könnte als eine einzelne logische Aufzeichnung gespeichert sein. Die beiden logischen Aufzeichnungen könnten folgendermaßen miteinander verglichen werden:



Um eine spezielle Information zu identifizieren, müssen wir die Anzahl der logischen Aufzeichnungen kennen und wissen, ob sie eine der logischen Aufzeichnungen für die Werbebriefe oder eine Aufzeichnung aus der Adressenliste ist.

Wie werden wir nun die logischen Aufzeichnungen der Werbebriefe von den logischen Aufzeichnungen der Adressenliste unterscheiden?

Wir wollen natürlich nicht anfangen zu spezifizieren, wo die Information auf der Diskette gespeichert ist. Der ganze Grund, weshalb wir uns mit logischen Aufzeichnungen befassen, liegt vor allem darin, daß wir uns keine Gedanken machen wollen, wie es auf der Oberfläche der Diskette aussieht. Wir kombinieren deshalb alle Aufzeichnungen in eine Datei. Eine Datei ist einfach eine Sammlung von ein oder mehreren Aufzeichnungen. Wieder haben wir es mit logischen Dateien zu tun und lassen dem Mikrocomputer die Aufgabe zu bestimmen, wo sich die physikalische Datei tatsächlich auf der Oberfläche der Diskette befindet. **Nun besitzen wir eine logische Datei für „Werbebriefe", in der jeder Abschnitt logisch aufgezeichnet ist und wir haben eine Datei für die „Adressenliste", in der jeder Name und Adresse zu einer logischen Aufzeichnung wird.**

Dateien und Aufzeichnungen stellen die fundamentale Struktur dar, die bei der Aufzeichnung großer Mengen von Informationen in Computersystemen verwendet wird, vom kleinsten Mikrocomputer-System bis zur Großcomputeranlage.

Dieses Konzept der logischen Dateien und logischen Aufzeichnungen ist gar nicht so verschieden von unserem Alltagsleben im Büro. Nehmen wir beispielsweise an, daß wir uns einen Brief, den wir von der Firma XYZ erhalten haben, ansehen sollen und der Preisangaben enthält. Wir können unsere Sekretärin bitten, uns den 27. Brief im dritten Fach des Büroschrankes zu holen. Wahrscheinlich werden wir aber unsere Sekretärin bitten, den Brief der Firma XYZ aus dem entsprechenden Ordner zu holen und daraus den Brief über die „Preisangaben" zu entnehmen. Der Ordner der Firma XYZ entspricht einer logischen Datei. Jeder Brief in diesem Ordner entspricht einer logischen Aufzeichnung. Unsere Sekretärin schließlich ersetzt die Intelligenz des Mikrocomputers, der imstande ist auf eine bestimmte Information zuzugreifen, wenn wir ihm eine Beschreibung dieser Information gegeben haben.

KASSETTENGERÄTE

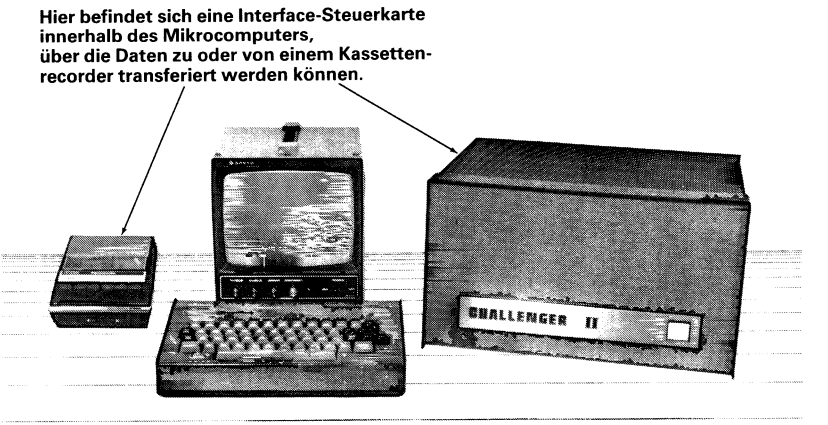
Sicherlich wird jetzt der eine oder andere „Mikrocomputer-Spezialist" feststellen, daß wir uns eigentlich gar kein Diskettensystem leisten können. Richtig. Gibt es preiswertere Möglichkeiten? Ja! Es gibt Kassetten und es gibt Lochstreifen.

Die Kassettengeräte, die zur Speicherung von Informationen für einen Mikrocomputer verwendet werden, sind genau die gleichen Kassettengeräte, wie wir sie sonst gewöhnlich verwenden. Wir können jeden beliebigen Kassettenrekorder zur Speicherung von Informationen für unser Mikrocomputer-System verwenden.

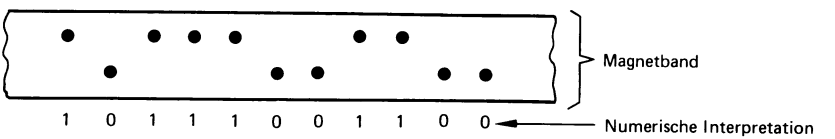
Es ist jedoch empfehlenswert, einen hochwertigen Kassettenrekorder und erstklassige Bänder mit unserem Mikrocomputer-System zu verwenden, da dieses keinerlei Fehler toleriert. Wenn bei einer Musikaufnahme diese infolge eines Fehlers im Band kurzzeitig unterbrochen wird, so spielt dies bei einer derartigen Aufzeichnung keine große Rolle. Bei der Speicherung von Informationen für unseren Mikrocomputer kann dadurch aber unter Umständen die ganze Kassette wertlos werden, da der Mikrocomputer fehlerhafte Zahlen erhält.

KASSETTEN-INTERFACE

Es genügt jedoch nicht, wenn wir einfach einen Kassettenrekorder kaufen, ihn neben unseren Mikrocomputer stellen und erwarten, daß beide ohne weiteres zusammenarbeiten können. Die tatsächliche Kommunikation erfolgt über eine entsprechende Steuerlogik innerhalb unseres Mikrocomputers. Dies kann folgendermaßen dargestellt werden:



Es gibt zwei verschiedene Arten, wie Kassettenrekorder Informationen auf einem Magnetband speichern können. Früher speicherte man Informationen digital, das heißt als eine Folge von magnetisierten Punkten auf der Oberfläche des Bandes:



Nahezu alle industriellen Kassettenrekorder speichern Informationen auf digitale Weise wie gezeigt. **Neuere Kassettenrekorder speichern die Informationen jedoch nicht digital, sondern sie zeichnen die Informationen als Töne auf.** Dies geschieht genau auf die gleiche Weise, wie sie Stimmen, Musik oder ähnliches aufzeichnen würden. Ein bestimmter Ton stellt die Ziffer 0 dar während ein anderer Ton die Ziffer 1 darstellt.

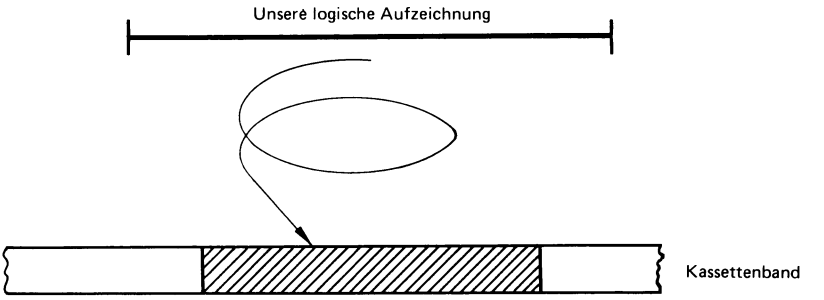
Die Tatsache, daß es eine „alte" und eine „neue" Methode zum Aufzeichnen von Daten auf Kassetten gibt, bedeutet nicht, daß der „neue" Weg besser ist, oder daß er den „alten" Weg überflüssig macht. „Alt" und „neu" bezieht sich nur auf die chronologische Reihenfolge. Tatsächlich werden auf die „alte" Weise mehr Daten auf eine Kassette gespeichert, und es gestattet das Lesen und Schreiben der Daten wesentlich schneller, benötigt jedoch teure Bandeinheiten. Deshalb ist das „alte" Verfahren besser, jedoch teurer als das „neue".
Der prinzipielle Vorteil bei der Verwendung von Tönen zur Aufzeichnung von Daten liegt darin, daß man jeden Standard-Kassettenrekorder für unser Mikrocomputer-System verwenden kann.

PLATTEN-ROMs

Das Platten-ROM (floppy-ROM) ist eine neue Möglichkeit zur Speicherung von Informationen, bei der ebenfalls Töne für die Darstellung von Daten verwendet werden. Das Platten-ROM ist eine Platte, die man einfach auf einem Plattenspieler abspielt. Dieser wird mit einem Kassettenrekorder verbunden und es werden die vom Platten-ROM kommenden Informationen auf eine Kassette überspielt. Die auf der Kassette aufgezeichneten Töne werden von unserem Mikrocomputer-System als binäre Daten interpretiert. Die binären Daten können Programmbefehle, Daten, die von einem Programm verwendet werden, oder beides darstellen.
ROM ist eine Abkürzung für Read Only Memory (Nur-Lesespeicher, Festwertspeicher). Die Platte enthält Informationen und ist daher ein Teil des Speichers des Mikrocomputers. Wir können Daten von der Platte lesen, können jedoch keine Daten einschreiben. Daher ist es ein „Nur-Lese"-Teil des Speichers des Mikrocomputers.
Das Platten-ROM wurde zuerst kommerziell von der amerikanischen Zeitschrift „Interface Age" eingeführt, die dieses Platten-ROM als Hilfsmittel verwendete, um ihren Lesern von Computern lesbare Programme des Magazins zu geben. Vorher mußten Programme in einer Programmiersprache abgedruckt werden, die dem Leser die Aufgabe des Einbringens des Programmes in den Mikrocomputer überließ, das eine mühsame und fehleranfällige Operation darstellte.

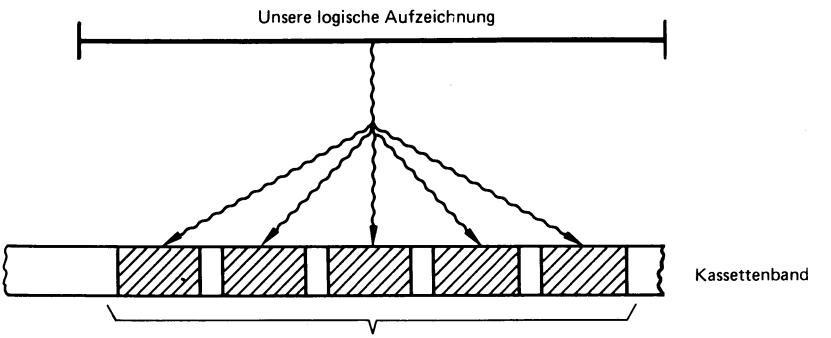
KASSETTEN-AUFZEICHNUNGEN

Informationen werden auf dem Band der Kassette als eine Folge von physikalischen Aufzeichnungen gespeichert. Es gibt kein Standardkassetten-Formatierschema, wie es die beschriebenen Sektoren und Spuren für eine Diskette sind. Die physikalischen Aufzeichnungen auf einer Kassette können entweder gleich lang, verschieden lang oder beliebig lang sein. Eine physikalische Aufzeichnung kann ein Zeichen lang oder so lang sein wie die gesamte Länge des Bandes. Die logische Aufzeichnung, die wir bei der Besprechung der Sektoren und Spuren auf den Disketten beschrieben haben, kann auf einer Kassette als eine einzige physikalische Aufzeichnung eingeschrieben werden:



Physikalische Aufzeichnung auf der Kassette

Oder die logische Aufzeichnung kann über eine Anzahl von physikalischen Aufzeichnungen verteilt sein:

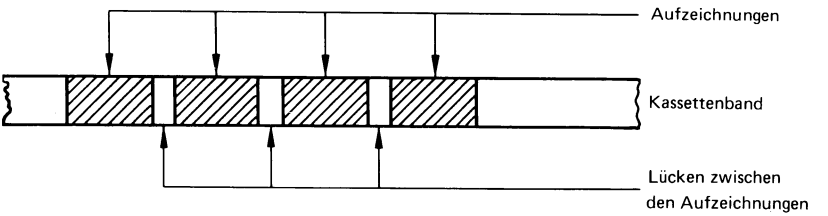


Fünf physikalische Aufzeichnungen auf der Kassette

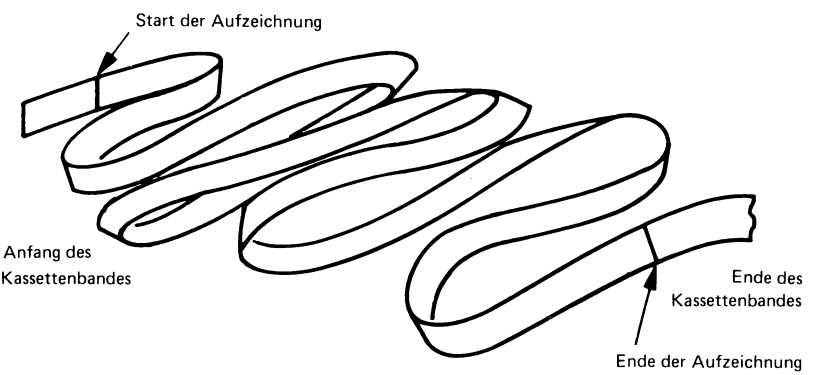
Der wesentlichste Unterschied zwischen einem Kassettengerät und einem Diskettengerät liegt darin, daß das Kassettengerät ein Baustein mit einem sequentiellen (oder seriellem) Zugriff ist, während das Diskettengerät einen Baustein mit wahlfreiem Zugriff darstellt. Wenn wir sagen, daß ein Kassettengerät ein sequentieller Baustein ist, so bedeutet dies, daß wir nicht beliebig auf der Oberfläche der Kassette hin- und herspringen können so wie auf der Oberfläche einer Diskette. Wenn wir zur 25. Aufzeichnung auf dem Band einer Kassette gelangen wollen, müssen wir die davor liegenden 24 Aufzeichnungen abzählen. Wenn Informationen am Ende eines Bandes aufgezeichnet sind, so kann es mitunter ziemlich lange dauern bis wir zu diesen gelangen. Zum Beispiel könnte die letzte Aufzeichnung auf einer Kassette erst in einer Zeit von 10 bis 15 Minuten erreichbar sein.

LÜCKEN ZWISCHEN DEN KASSETTENAUFZEICHNUNGEN

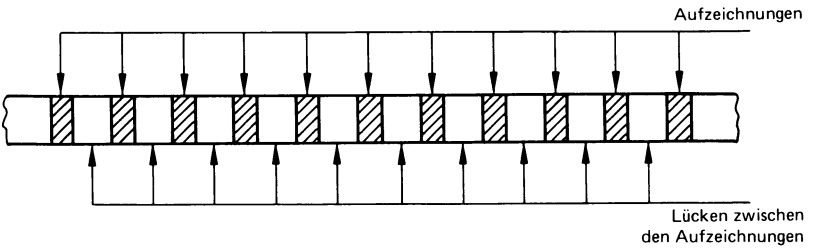
Wenn wir mehr als eine Aufzeichnung auf einem Kassettenband haben, dann müssen diese durch entsprechende Lücken voneinander getrennt werden. Dies kann folgendermaßen dargestellt werden:



In diesen Lücken sind keine nutzbaren Aufzeichnungen gespeichert. Mit steigender Zahl der Aufzeichnungen auf einer Kassette steigt auch die Anzahl der Lücken, so daß die Gesamtzahl der nutzbaren Informationen, die man in einer Kassette speichern kann, abnimmt. **Um ein Maximum auf einer Kassette unterzubringen, müßten wir daher die Informationen kontinuierlich ohne Lücken speichern:**

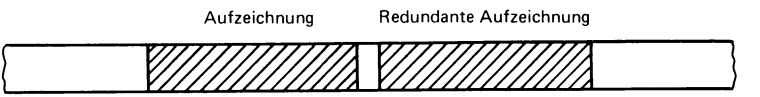


Wenn wir andererseits eine Vielzahl sehr kurzer Aufzeichnungen haben, werden wir den größten Teil der Kassette für diese Lücken verschwenden:

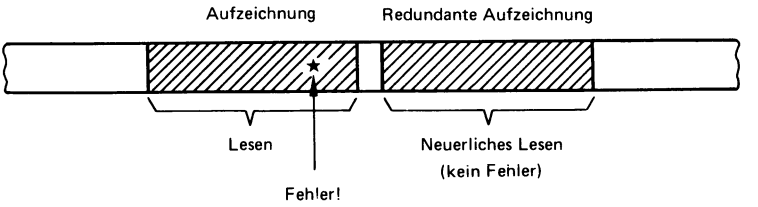


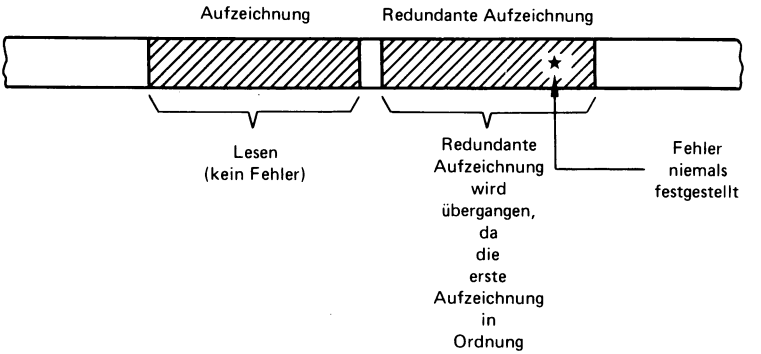
ZUVERLÄSSIGKEIT VON KASSETTEN
REDUNDANTE AUFZEICHNUNG

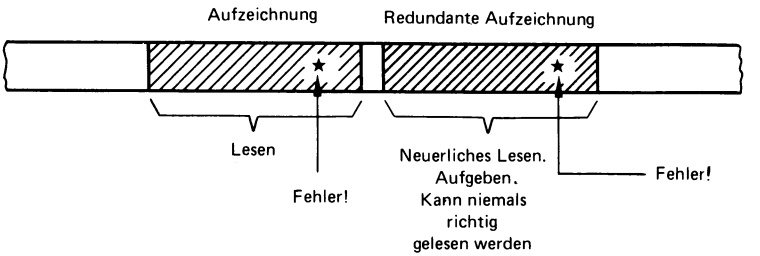
Warum zerbricht man sich dann überhaupt den Kopf mit einer Anzahl von kurzen Aufzeichnungen? Die Antwort ist, um Fehler zu verhindern. Kassetten sind nicht sehr zuverlässig. Man kann sehr leicht die Oberfläche des Bandes zerkratzen oder beschädigen und die magnetische Schicht des Bandes nutzt sich bei wiederholter Verwendung ab. **Zur Vermeidung von Fehlern zeichnet jedes gut entwickelte Mikrocomputer-System jede Information doppelt auf einer Kassette auf. Dies wird als redundante Aufzeichnung bezeichnet.** Wenn wir angenommen nun eine bestimmte Länge einer Aufzeichnung auf einer Kassette haben, so werden daraus zwei Aufzeichnungen, wenn wir die redundante Aufzeichnung verwenden. Dies kann folgendermaßen dargestellt werden:



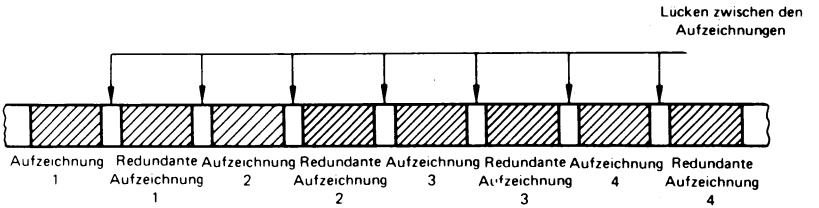
Selbst wenn sich jetzt ein oder mehrere Fehler in einer dieser beiden Aufzeichnungen befinden, so wird die Aufzeichnung korrekt gelesen:



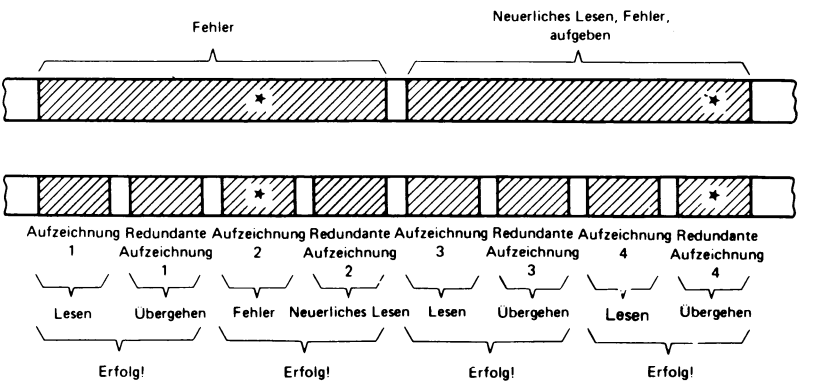




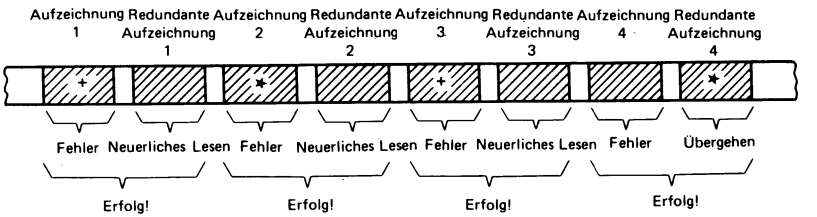
Nun nehmen wir an, daß wir eine längere Aufzeichnung in vier kürzere Aufzeichnungen aufteilen. Jede der vier kürzeren Aufzeichnungen wird seine eigene redundante Aufzeichnung haben, das heißt es gibt nunmehr insgesamt acht Aufzeichnungen auf der Kassette. Dies kann folgendermaßen gezeigt werden:



Die beiden Fehler, die andernfalls die Kassette unbrauchbar machen würden, können nun toleriert werden. Erinnern wir uns daran, daß Fehler auf dem Band auftreten, wenn dieses physikalisch beschädigt ist. Fehlstellen auf einem Band wiederholen sich meist, wenn die Kassette beschädigt wurde. Dies kann folgendermaßen dargestellt werden:



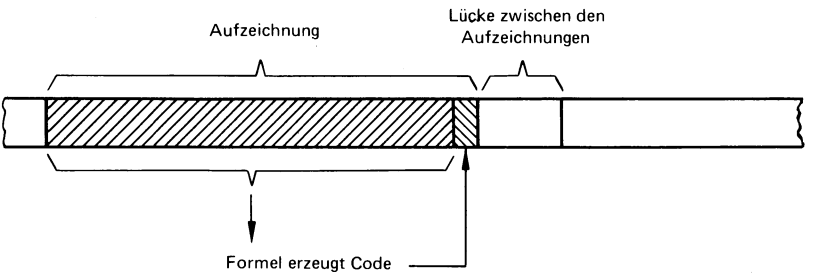
Unsere Kassette ist jedoch gegenüber Fehlern unempfindlich, da wir einfach die Anzahl der Aufzeichnungen erhöht haben.
Sogar zwei weitere Fehler können noch verkraftet werden, bevor wir die Kassette ausscheiden müssen. Dies kann folgendermaßen dargestellt werden:



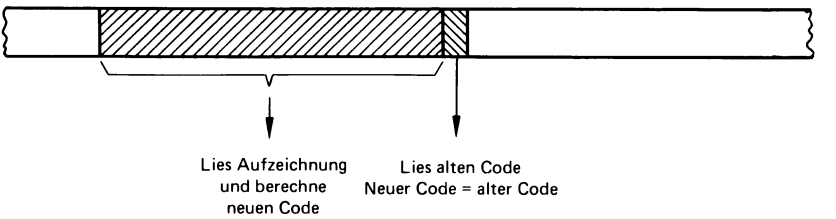
Wir können zusammenfassend sagen, daß mit dem Steigen der Anzahl der Aufzeichnungen auf einer Kassette die Gesamtzahl der möglichen Informationen abnimmt, aber die Empfindlichkeit gegenüber Fehlern stark zurückgeht, vorausgesetzt wir verwenden die redundante Aufzeichnung.

FEHLER-ERKENNUNGS-CODE

Ein Mikrocomputer-System vergewissert sich, ob es eine Aufzeichnung korrekt gelesen hat, indem es einen speziellen Fehler-Erkennungscode am Ende jeder Aufzeichnung schreibt. Dieser spezielle Code wird durch Anwendung einer Formel für alle Zahlen in der Aufzeichnung berechnet. Dies kann folgendermaßen gezeigt werden:



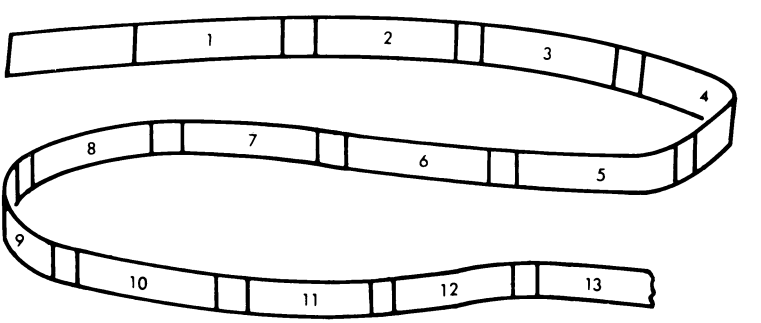
Wenn das Mikrocomputer-System eine Aufzeichnung wieder zurückliest, so berechnet es einen neuen Fehler-Erkennungscode, der diesmal auf den zurückgelesenen Zahlen basiert. Dann liest es den alten Fehler-Erkennungscode. Wenn die zurückgelesene Aufzeichnung korrekt ist, ist der neue und der alte Fehler-Erkennungscode identisch:



Wenn irgendwelche Zahlen nicht korrekt zurückgelesen wurden, dann wird der neue Code mit dem alten Code nicht übereinstimmen. Daher ist anzunehmen, daß die Information nicht richtig ist.

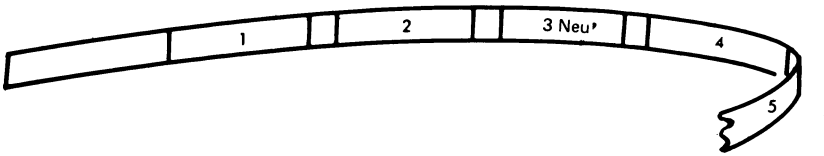
LESEN UND SCHREIBEN VON KASSETTEN

Die Tatsache, daß zu Informationen auf Kassetten sequentiell zugegriffen werden muß, macht es ziemlich schwierig, auf die gleiche Kassette zu schreiben und zu lesen. Nehmen wir beispielsweise an, daß wir eine Liste von Namen und Adressen auf eine Kassette speichern. Dies kann folgendermaßen dargestellt werden:

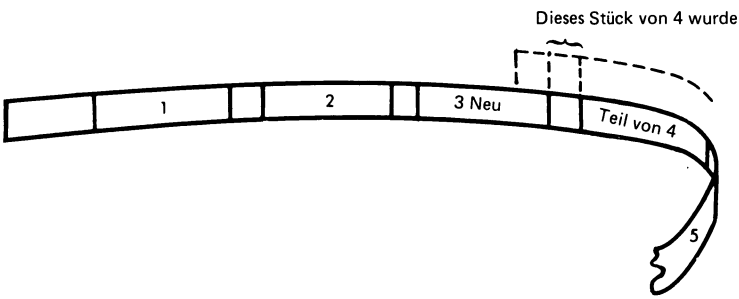


Die Zahlen 1 bis 13 stellen 13 individuelle Adressen dar.

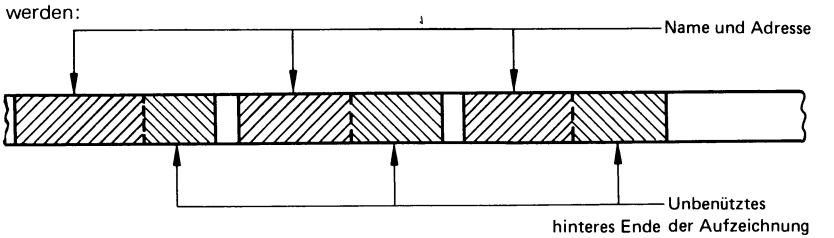
Nun nehmen wir an, daß wir den dritten Namen und Adresse 3 ändern wollen. Dies sieht scheinbar sehr einfach aus. Wir schreiben einfach den neuen Namen und Adresse über die alte Aufzeichnung:



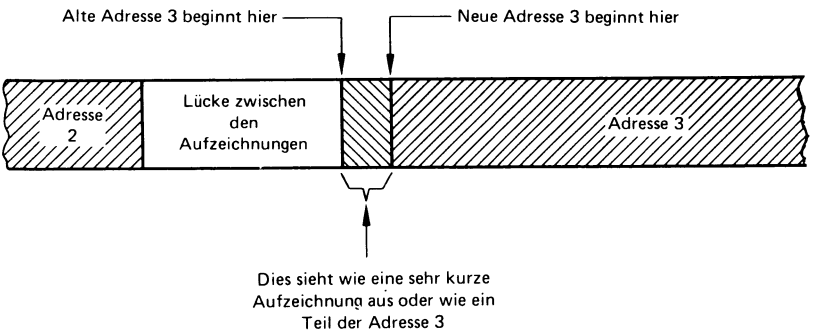
Aber warten wir einen Moment. Es kann dies eine sehr tückische Operation werden. Angenommen der neue Name und Adresse sei länger als der alte, so werden wir nun einen Teil des nächsten Namens und Adresse löschen.



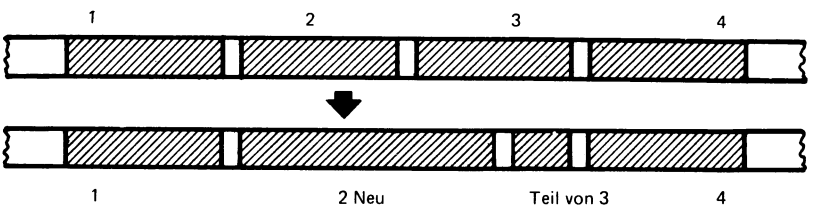
Wir können dieses Problem umgehen, indem wir den gleichen Platz für jeden Namen und Adresse reservieren, unabhängig von der Anzahl der Zeichen, die tatsächlich in dem Namen und der Adresse enthalten sind. Dies kann folgendermaßen dargestellt werden:



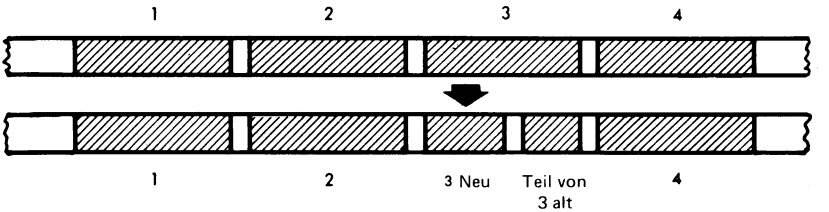
Auch dies ist noch keine ausreichende Lösung, da es voraussetzt, daß der Antrieb unseres Kassettenrekorders äußerst präzise arbeitet. Angenommen es würde dadurch das neue Einschreiben ein ganz kleines Stück zu spät beginnen.



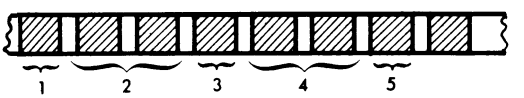
Wenn das Mikrocomputer-System die Adresse 3 von der Kassette liest, so wird es den Rest des alten Namens und Adresse aufnehmen, was zu einem Lesefehler führt. Starten wir dagegen das Schreiben zu früh, wo werden wir am Ende der Aufzeichnung eine Lücke erhalten und es wird beim neuerlichen Lesen wieder ein Fehler entstehen. Auf jeden Fall wird das Überschreiben früherer Aufzeichnungen Ärger verursachen. Billige Kassettenrekorder bescheren uns in der Regel mehr Unannehmlichkeiten als teure Kassettenrekorder, da die billigen Ausführungen einen weniger präzisen Antriebsmechanismus besitzen. Das wahre Problem liegt jedoch darin, daß ein einziger Fehler die ganze Kassette verderben kann. Es spielt keine Rolle, ob dieser eine Fehler häufig oder gelegentlich auftritt. In jedem Fall wird die Kassette unbrauchbar.
Wenn wir lange und kurze Aufzeichnungen auf einer Kassette gemischt haben, dann wird das Lesen und Schreiben auf der Kassette zu einer hoffnungslosen Aufgabe. Wenn jede logische Aufzeichnung auf einer Kassette als eine einzelne physikalische Aufzeichnung gespeichert ist, dann können wir natürlich nicht verschieden lange Aufzeichnungen auf die gleiche Länge schreiben. Eine längere Aufzeichnung wird einen Teil der nächsten Aufzeichnung löschen:



Wenn eine kürzere Aufzeichnung einen Teil der alten Aufzeichnung unverändert läßt, führt das dann aber zu einem Fehler:

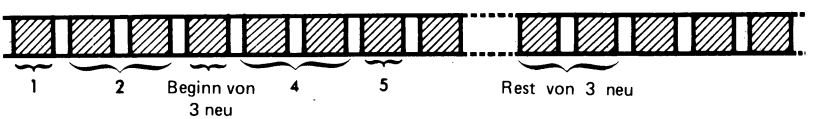


Wenn logische Aufzeichnungen verschiedener Länge auf einer Kassette als eine Folge von physikalischen Aufzeichnungen gleicher Länge gespeichert werden, können wir auf die gleiche Kassette durch Aufteilung der logischen Aufzeichnungen schreiben und lesen, so wie wir es bei den Disketten gemacht haben. Dies kann folgendermaßen dargestellt werden:

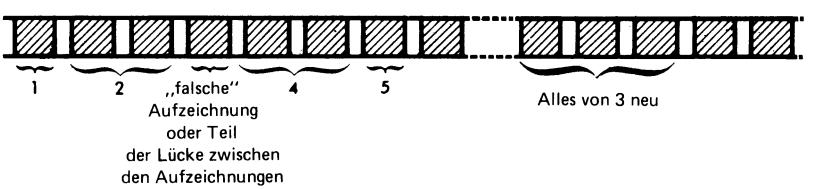


Bei dieser Kassette gibt es zwei Möglichkeiten wie wir die neue Aufzeichnung 3 einsetzen können:

Möglichkeit A



Möglichkeit B



Das Problem mit diesen Schemas liegt darin, daß es endlos dauern kann, wenn wir die Kassette vor- und zurückspulen müssen. Wenn wir die Variante A benützen, so müssen wir folgende Schritte zum Lesen der Aufzeichnung, wie gezeigt, verwenden:

1) Auffinden der logischen Aufzeichnung 2
2) Auffinden des Beginns der logischen Aufzeichnung 3
3) Vorlauf der Kassette zum Auffinden des Rests der logischen Aufzeichnung 3
4) Zurückspulen zum Beginn der logischen Aufzeichnung 4

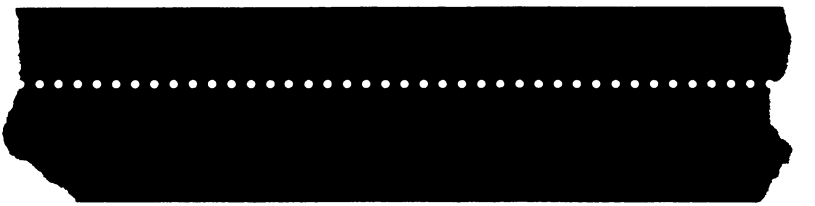
etc.

Wenn wir die Variante B benützen, verschwenden wir weiteren Platz auf dem Band. Wenn unsere Aufzeichnungen in irgendeiner Art (z. B. alphabetisch) geordnet sind, so werden sie bald völlig durcheinander sein. **Wir können abschließend sagen, daß das Vorhandensein nur eines Kassettengerätes in unserem Mikrocomputer-System sehr gefährlich sein kann. Wir müssen wenigstens zwei Kassetteneinheiten besitzen, eines von dem gelesen wird und das andere in das eingeschrieben wird.**

LOCHSTREIFENGERÄTE

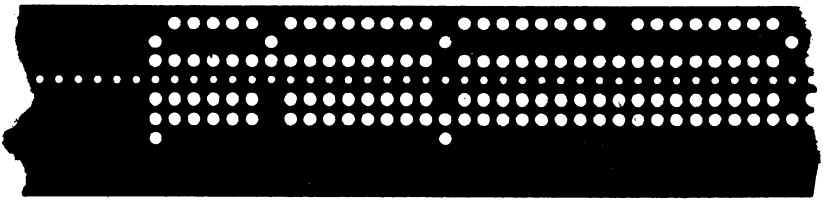
Es gibt einen weiteren Träger für die Speicherung großer Datenmengen, das jedoch primitiver als ein Kassettenrekorder ist. Es ist der Lochstreifen. Es gab eine Zeit, als Lochstreifen das einzige billige Hilfsmittel für die Speicherung von Computerinformationen darstellten, wobei Kassetten jedoch heute mindestens ebenso billig sind und dazu wesentlich schneller. Nichtsdestoweniger gibt es viele Mikrocomputer-Systeme, die Lochstreifen zur Speicherung von Informationen verwenden. Daher müssen wir auch diese besprechen.

Lochstreifen bestehen aus einem langen schmalen Streifen dünnen Papiers:



LOCHSTREIFENZEICHEN

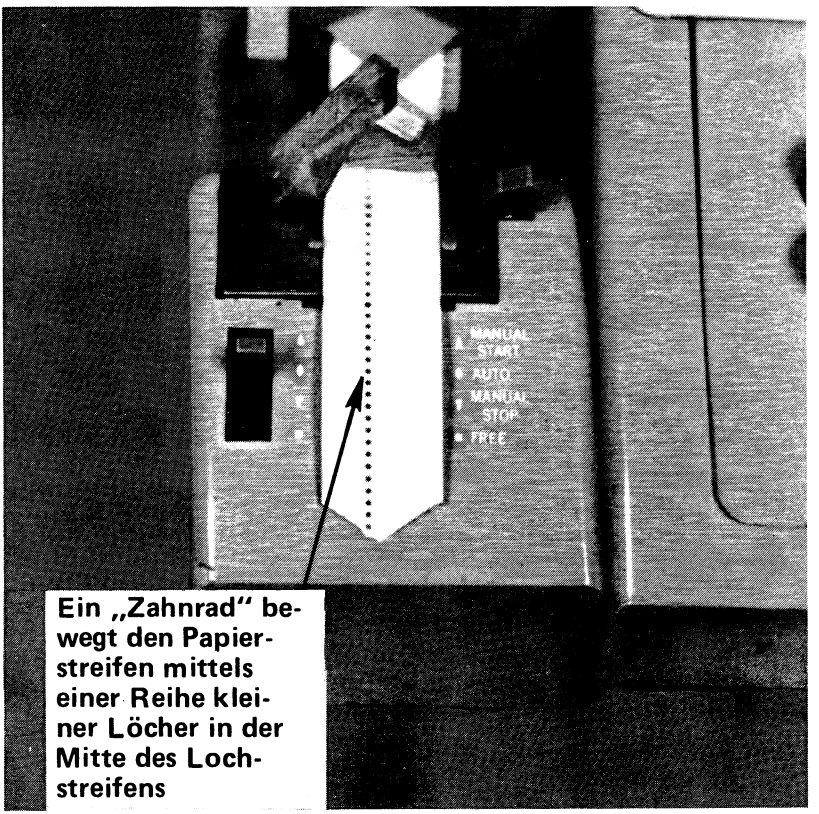
Informationen werden auf einem Lochstreifen durch Stanzen von Löchern in das Papierband gespeichert:



Jede vertikale Reihe von Löchern auf dem Lochstreifen stellt ein Zeichen dar.

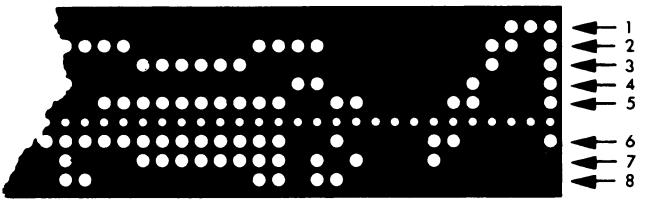
FÜHRUNG DES LOCHSTREIFENS

Zusätzlich zu den Löchern, die die Zeichen darstellen, besitzen die meisten Lochstreifen noch eine Reihe von kleinen Transportlöchern, die in etwa durch die Mitte des Bandes verlaufen, wie oben gezeigt wurde. Diese Perforationslöcher dienen zum Bewegen des Lochstreifens:



LOCHSTREIFENKANÄLE

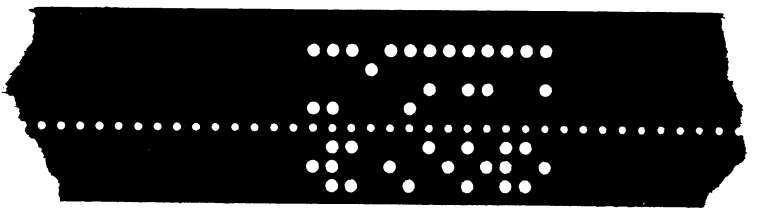
Ein Lochstreifen besitzt fünf, sieben oder acht Spuren. Die nachstehende Abbildung zeigt einen Acht-Spur-Lochstreifen, wobei es acht Positionen auf jeder vertikalen Linie gibt, in die ein Loch gestanzt werden kann:



Die Positionen der acht Löcher werden Kanäle genannt.
Die Kombination der gelochten und ungelochten Positionen stellen einen Code dar, der aus bestimmten Zeichen besteht. Wenn wir von „Zeichen" sprechen, so meinen wir jeden Buchstaben des Alphabets, jede numerische Ziffer oder irgendwelche speziellen Zeichen, wie einen Punkt, ein Komma, ein Ausrufungszeichen, ein Fragezeichen etc.
Die Art des verwendeten Zeichencodes braucht uns im Augenblick nicht näher zu interessieren.
Die Reihen oder Kanäle, die durch die Lochreihen gebildet werden, sind etwa 2,5 mm voneinander entfernt. Auf etwa 2,5 cm des Lochstreifens lassen sich somit 10 Informationszeichen aufbringen. Der Name:

Joe Bitburger

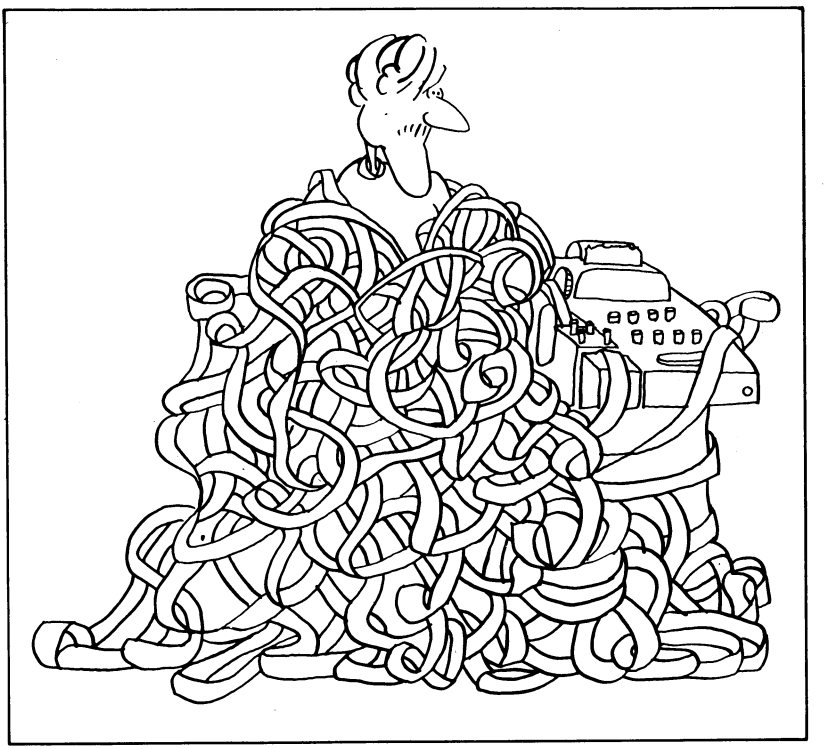
benötigt daher etwa 3,3 cm Lochstreifen, wenn er genau so aufgezeichnet ist wie anschließend dargestellt:



Der grundlegende Nachteil von Lochstreifen liegt darin, daß man außerordentliche Mengen davon braucht. Während die Speicherdichte bei Kassetten sehr hoch ist, benötigen wir etwa 650 m Lochstreifen, um die gleiche Menge an Information aufzuzeichnen wie auf einer einzigen Kassette mit 90 Minuten Spieldauer oder eine Seite einer einzelnen einseitigen Diskette.
Ein weiterer Nachteil des Lochstreifens liegt darin, daß es relativ lange dauert um In-

formationen einzuschreiben oder zurückzulesen. Lese- und Schreibzeiten liegen normalerweise zwischen 10 und 100 Zeichen pro Sekunde. Dies mag zwar ziemlich schnell aussehen, bei Kassetten lassen sich jedoch 100 bis 1000 Zeichen pro Sekunde einschreiben oder auslesen. Die typischen Lese- und Schreibgeschwindigkeiten von Disketten liegen zwischen 1000 und 10000 Zeichen pro Sekunde. Bei starren Magnetplatten liegen die Lese- und Schreibgeschwindigkeiten über einer Million Zeichen pro Sekunde.

Alle zeitlichen Begriffe sind im Zusammenhang mit Computern natürlich relativ. Warum sieht man zehn Zeichen pro Sekunde als langsam an? Der Grund hierfür ist, daß wir gewöhnlich Hunderte oder Tausende von Zeichen zu einer bestimmten Zeit lesen oder schreiben. Hierbei haben wir oft nichts weiter zu tun als diese Schreib- oder Leseoperationen zu beobachten. Aber auch 10 Sekunden sind eine lange Zeit, wenn sie wiederholt auftritt.



KAPITEL 1

FRAGEN

1. Die Technologie, die es ermöglicht, daß moderne Computer heute derart leistungsfähig sind und auf kleinstem Raum untergebracht werden können, nennt man _____ oder _____ .
2. Der zentrale LSI-Baustein, der eigentliche Rechner in einem Mikrocomputer-System, ist der _____ .
3. Wenn man bei einem Mikrocomputer-System die Daten über eine Tastatur eingibt, so „spricht" der Mikrocomputer über eine _____ zu uns.
4. Wenn man für die Bildschirmanzeige ein normales Fernsehgerät verwendet, so liegt der wesentliche Unterschied nur in einer geringeren _____ des dargestellten Textes.
5. Während der Fernseh-Bildschirm im allgemeinen zur Darstellung von Bildern, von Schwarz bis Weiß mit allen dazwischenliegenden Grautönen dient, liefert eine Video-Anzeige kein _____ , sondern nur Schwarz und Weiß.
6. Die heutigen Bildschirmgeräte verwenden zur Anzeige eine _____ engl. _____, die in Zukunft durch andere Anzeige-Technologien ersetzt werden wird.
7. Die manuelle Eingabe von Daten und Befehlen erfolgt bei einem Mikrocomputer-System über eine _____ .
8. Die Schaltung zur Anpassung zweier Geräte oder Geräteteile wird ein _____ oder eine _____ -Schaltung genannt.
9. Die Ausgabe von Daten oder Resultaten aus dem Mikrocomputer kann außer über den Bildschirm auch über einen _____ erfolgen.
10. Spezielle Schreibmaschinen können zum Ausdrucken von Resultaten verwendet werden, sind jedoch wesentlich _____ als Drucker.
11. Wenn Tastatur und Drucker in einem Mikrocomputer-System voneinander getrennt sind, so muß der _____ durch eine entsprechende Programmierung das Drucken des eingetippten Zeichens bewirken.
12. Wenn der Mikrocomputer ein Zeichen zur Anzeige oder zum Drucker zurückgibt, so arbeitet er im sogenannten _____ - _____ .
13. Eine Folge oder Sequenz von Befehlen für einen Mikrocomputer nennt man ein _____ .
14. Ein Befehl besteht für einen Mikrocomputer aus einer bestimmten Zahl. Zahlen können jedoch auch _____ darstellen.
15. Befehle und Daten bestehen für den Mikrocomputer nur aus Zahlen. Er kann sie nur unterscheiden, indem er diese Zahlen richtig _____ .
16. Ein typischer Mikrocomputer führt einen Befehl in einem Zeitintervall von einem _____ bis zu einem _____ einer Sekunde aus.
17. Eine _____ ist in der Computertechnik oder Elektronik die übliche Einheit für ein Millionstel einer Sekunde.
18. Das gebräuchlichste Massenspeichergerät in der Mikrocomputer-Technik ist die einem Plattenspieler ähnelnde _____ - _____ .
19. Für Mikrocomputer-Systeme werden meist _____ - _____ mit einem Durchmesser von ca. 133 mm verwendet.

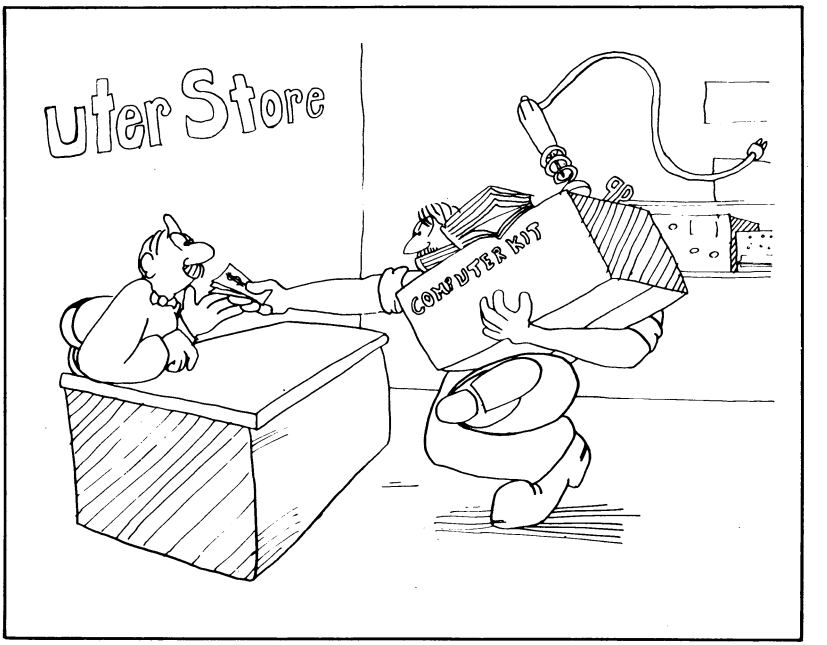
20. Die Aufzeichnung auf den Floppy-Disks erfolgt entlang von _____ mit einem magnetischen Schreib-/Lesekopf.
21. Zum raschen Schreiben oder Auffinden einer Information auf einer Floppy-Disk wird jede Spur in _____ aufgeteilt.
22. Zum Auffinden der Sektoren werden für den Antriebsmechanismus in die Floppy-Disk _____ gestanzt.
23. Vor der Verwendung einer neuen Diskette muß diese zuerst _____ werden, was vom Mikrocomputer-System automatisch ausgeführt wird.
24. Mit _____ Platten kann man bei großen Mikrocomputer-Systemen eine größere Anzahl von Informationen speichern und rascher verarbeiten.
25. Disketten und starre Platten nennt man allgemein Massenspeicher-Geräte mit _____, im Gegensatz zu Magnetband-Speichern mit seriellem Zugriff.
26. Eine Information, eine Idee oder eine Operation, nennt man in der Computertechnik allgemein eine _____ Einheit.
27. Die _____ Einheit ist dagegen die tatsächliche physikalische Ausführung hiervon.
28. Die fundamentale Struktur der Aufzeichnung großer Mengen von Informationen sind die _____ Dateien und _____ Aufzeichnungen.
29. Eine sehr preiswerte Aufzeichnung kann außer mit Disketten auch mit _____ erfolgen.
30. Die Aufzeichnung der digitalen Informationen 0 und 1 auf Kassetten erfolgt in Form von _____.
31. Der wesentliche Unterschied zwischen einem Disketten-System und einem Kassettengerät liegt darin, daß bei der Kassette ein _____ oder _____ Zugriff zu den Aufzeichnungen erfolgt.
32. Zur Vermeidung von Aufzeichnungsfehlern wird jede Information auf einer Kassette doppelt aufgezeichnet. Man bezeichnet dies als _____ Aufzeichnung.
33. Ein weiteres preiswertes Aufzeichnungs-Medium ist der _____.
34. Auf einem Lochstreifen werden die Informationen in mehreren Spuren aufgezeichnet, wobei eine _____ Reihe von Löchern ein Zeichen darstellt.

ANTWORTEN, 1. KAPITEL

1. Großintegration, LSI
2. Mikroprozessor
3. Video-Anzeige
4. Auflösung
5. Grau
6. Kathodenstrahl-Röhre (engl. CRT)
7. Tastatur
8. Interface, Schnittstellen
9. Drucker
10. langsamer
11. Mikrocomputer
12. Echo-Betrieb
13. Programm
14. Daten
15. interpretiert
16. Millionstel, Zehnmillionstel
17. Mikrosekunde
18. Floppy-Diskeinheit
19. Mini-Floppies
20. Spuren
21. Sektoren
22. Löcher
23. formatiert
24. starren
25. wahlfreiem Zugriff
26. logische
27. physikalische
28. logischen, logischen
29. Kassetten
30. Tönen
31. sequentieller, serieller
32. redundante
33. Lochstreifen
34. vertikale

Kapitel 2
WIR VERWENDEN EINEN MIKROCOMPUTER UND BEOBACHTEN WIE ER WÄCHST

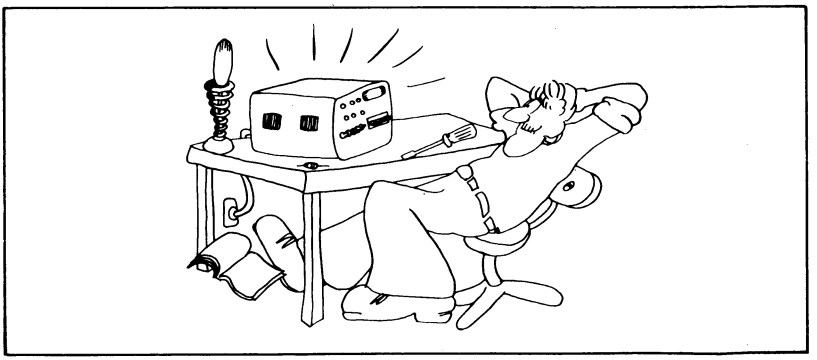
Sehen wir uns nun die vielen Verwendungsmöglichkeiten, die ein Mikrocomputer-System bietet, genauer an.
Um uns diese Aufgabe zu erleichtern, lernen wir Joe Bitburger, einen unerschrockenen Computer-„Hobbyisten" kennen. Nachdem dieser gelegentlich mit Computern zu tun hat, kauft sich Joe eines Tages in einem Anfall von Unvernunft einen Mikrocomputer.



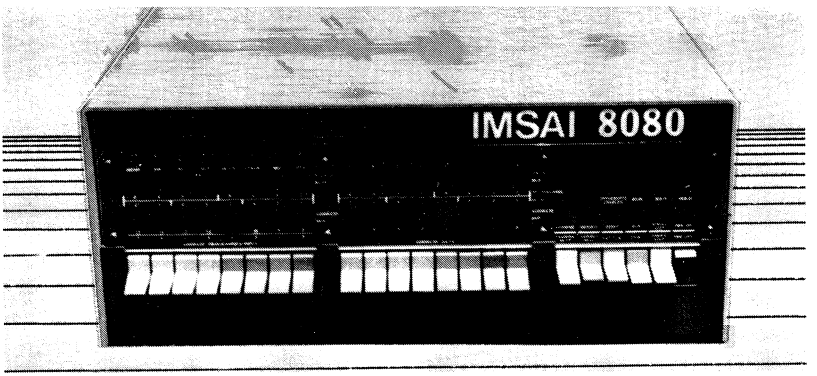
Joe's Schritt ist weder ungewöhnlich, noch sehr vernünftig. Joe hat eine Menge zu lernen.

Wenn gesagt wurde, daß Joe gelegentlich mit Computern arbeitet, so bedeutet dies, daß er in seiner Firma gelegentlich Computerprogramme schreibt und hierbei eine Computersprache – genannt FORTRAN – verwendet. Jeden Morgen nimmt Joe pflichtgemäß einen Packen Karten (Computerkarten, keine Spielkarten) zum Compu-

terzentrum in seinem Büro. Verläuft alles ordnungsgemäß, so kann er kurz danach nach dem Mittagessen im Computerzentrum eine große Menge Papier mitnehmen, auf dem seine Resultate gedruckt sind.
Einer der Gründe für Joe's überraschende Handlung war vielleicht die Tatsache, daß er zwar seit über zwei Jahren Computerprogramme schreibt, aber selbst mit dem Computer niemals Kontakt hatte.
Joe nimmt seinen Mikrocomputer stolz mit nach Hause um ihn zusammenzubauen und zu verwenden.
Wir müssen nun einige Wochen (oder Monate) von Joe Bitburgers Leben überspringen.
Joe hat sich einen Mikrocomputer-Bausatz gekauft. Wie uns jeder Computer-Hobbyist bestätigen kann, verschafft uns der Zusammenbau eines solchen Bausatzes viele Stunden mit Sorgen, in denen wir sicher Zweifel am Verstand der Bausatzhersteller und der Verfasser der zugehörigen Handbücher bekommen.
Schließlich, nach vielen Rückfragen beim Lieferanten des Computerbausatzes und manchen Auseinandersetzungen mit diesem wegen schlechter Lötstellen, mangelhafter Funktionsbeschreibung im Handbuch etc. **bringt Joe Bitburger seinen Mikrocomputer zum Arbeiten.**

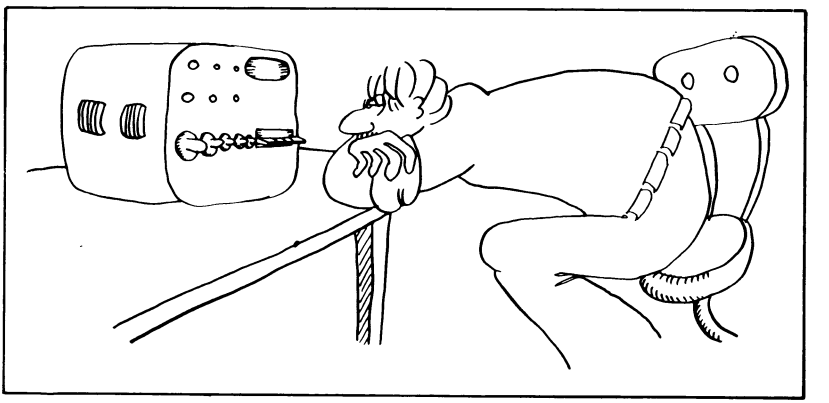


Armer Joe, nur die Freude an einem überstandenen großen Abenteuer hält ihn glücklich, **alles was er hat ist ein Mikrocomputer-Gehäuse:**



EIN PROGRAMM SCHAFFEN UND ES ZUM ARBEITEN BRINGEN

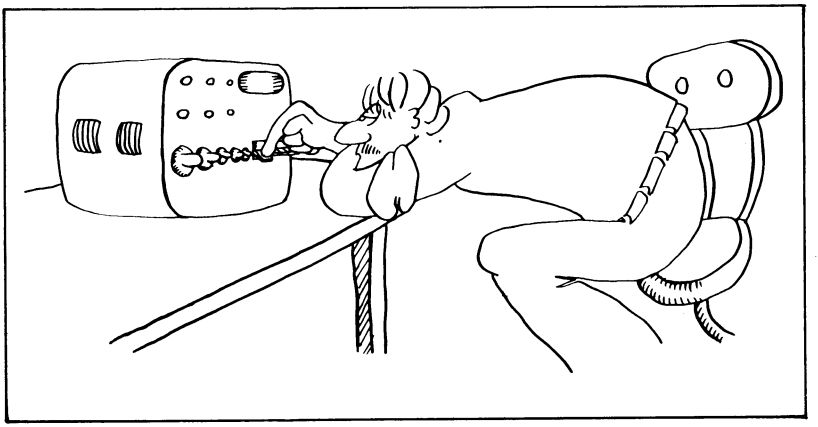
Nachdem er seinen Mikrocomputer zusammengebaut hat, möchte Joe auch etwas damit anfangen. Es ist reichlich wenig, was man damit anfangen kann. **Joe entscheidet sich schließlich dafür, ein Programm zu schreiben, bei dem sich einfach ein Licht auf der Frontplatte bewegt:**



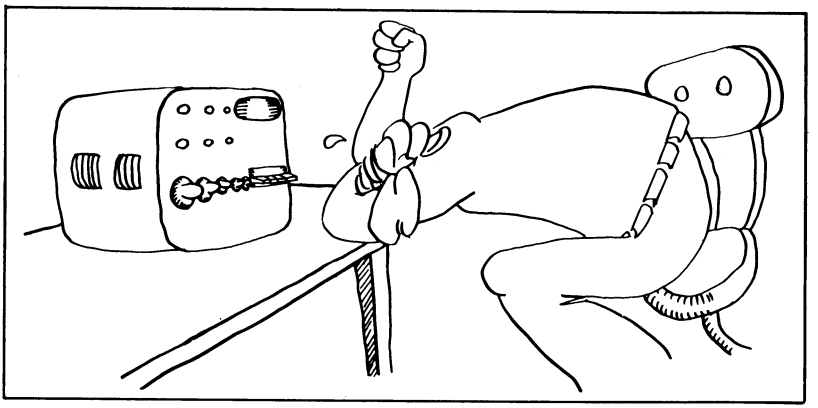
Joe schreibt sein Programm auf ein Blatt Papier unter Verwendung einer „Programmiersprache", die er für diesen Zweck lernen muß. Dann wandelt er das Programm in eine Folge von Zahlen um. Erinnern wir uns daran, daß alle Programme zu einer Folge von Zahlen werden müssen, da dies die einzige Art und Weise ist, in der ein Mikrocomputer ein Programm verstehen kann. Nachdem Joe nun diese Folge von Zahlen gebildet hat, muß er sie in den Speicher des Mikrocomputers laden. Joe hat jedoch keine Tastatur, sondern nur seinen Mikrocomputer. Was macht er also unter diesen Umständen?

DIE FRONTPLATTE EINES MIKROCOMPUTERS

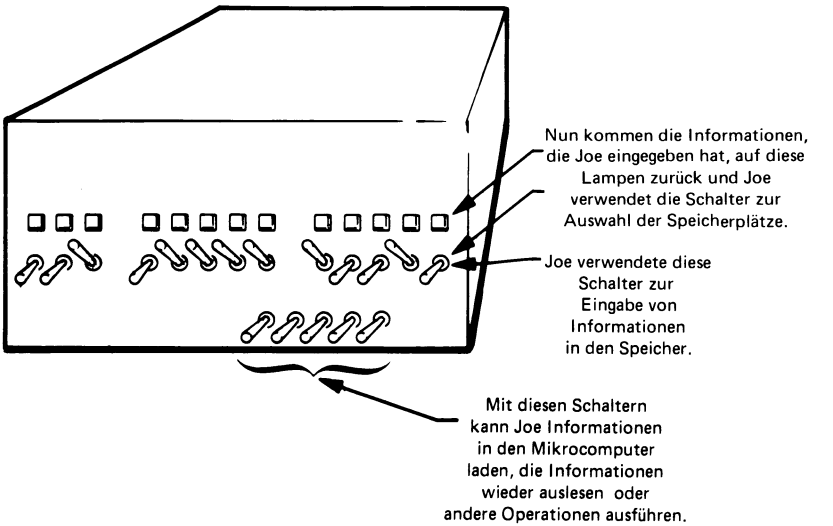
Der Mikrocomputer, den Joe gekauft hat, besitzt eine Frontplatte mit Schaltern und Lampen. **Joe lernt daher, wie man Zahlen in den Speicher des Mikrocomputers bringt, indem man die Schalter auf der Frontplatte in der richtigen Reihenfolge betätigt:**



Zwanzig Minuten später, und mit wunden Fingern hat Joe diese Aufgabe erledigt. **Gespannt schaltet er den Mikrocomputer ein. Was geschieht? Nichts.**



Daher entschließt sich Joe, die tatsächlich in dem Mikrocomputer gespeicherten Informationen zu überprüfen. Dazu benützt Joe wieder die Schalter und Lampen auf der Frontplatte. Es sind dieselben Schalter und Lampen, die er zum Laden der Information in den Speicher des Mikrocomputers verwendet hatte. Dies kann grundsätzlich folgendermaßen dargestellt werden:



FRONTPLATTEN-FUNKTIONEN

Wir brauchen uns jetzt nicht den Kopf darüber zu zerbrechen, wie die Schalter und Lampen auf einer Frontplatte genau arbeiten. Das würde uns im Moment unnötig verwirren. Die vorher gezeigte Frontplatte ist wahllos aus dem großen Angebot der vielen Mikrocomputer-Hersteller herausgegriffen. In Wirklichkeit unterscheiden sich die vielen verschiedenen Mikrocomputer-Typen schon im Design. **Grundsätzlich werden Frontplatten zum Laden von Informationen in den Speicher des Mikrocomputers verwendet, sowie zur Prüfung des Inhalts des Speichers und allgemein zur Steuerung der Operationen von Mikrocomputern.**

Um den Inhalt des Speichers des Mikrocomputers zu überprüfen, liest Joe einfach sein Handbuch und folgt den Befehlen Schritt für Schritt. Genau das würden wir auch machen, wenn wir uns in der mißlichen Lage von Joe befänden.

Was findet Joe nun?

Zwei Schalter, die eingeschaltet sein sollten, waren auf „Aus" und ein Schalter, der ausgeschaltet sein sollte, war auf „Ein".

Unverzagt stellt Joe die falschen Schalterstellungen richtig und versucht es noch einmal.

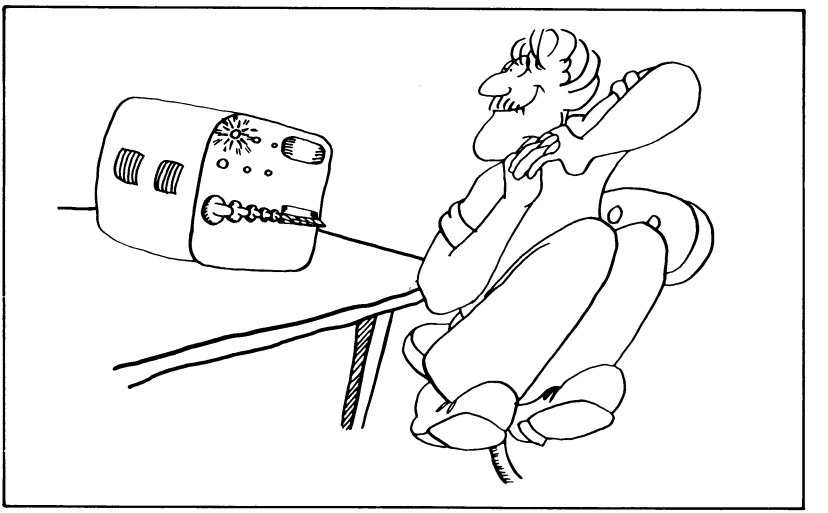
Aber noch immer funktioniert das Programm nicht.

Also unterzieht sich Joe nochmals der mühsamen Aufgabe, den Inhalt des Speichers seines Mikrocomputers zu überprüfen. Alles ist nunmehr in Ordnung. Aber das Programm arbeitet nicht, warum?

Also kann nur mehr die Zahlenfolge, die das Programm darstellt, falsch sein. Joe kehrt zu seinem Programm zurück. Er überprüft es sorgfältig und entdeckt, daß hier in der Tat Fehler vorhanden sind.

Was Joe nun tut, nennt man Fehlersuche und Fehlerbeseitigung (debugging) eines Programmes. Die einzelnen Fehler in einem Programm werden im Englischen „Bugs" (Wanzen) genannt.

Das Korrigieren der fehlerhalten Schalterstellungen und dann das Finden mehrerer Fehler im Programm kann ziemlich viel Zeit beanspruchen. **Joe hat tatsächlich hierzu zwei Tage gebraucht. Er sitzt nun triumphierend vor seinem Mikrocomputer und beobachtet wie die Lichter über die Frontplatte eilen:**



Was für eine Enttäuschung.

Hat Joe eine Menge Geld und Zeit ausgegeben, nur um zu sehen, wie sich ein paar Lichter auf der Frontplatte bewegen? Das ist in der Tat etwas ganz anderes, als die großen Dinge, die geschehen, wenn Joe seinen Kartenstapel im Computerzentrum abgibt und dann die fertig ausgedruckten Ergebnisse erhält.

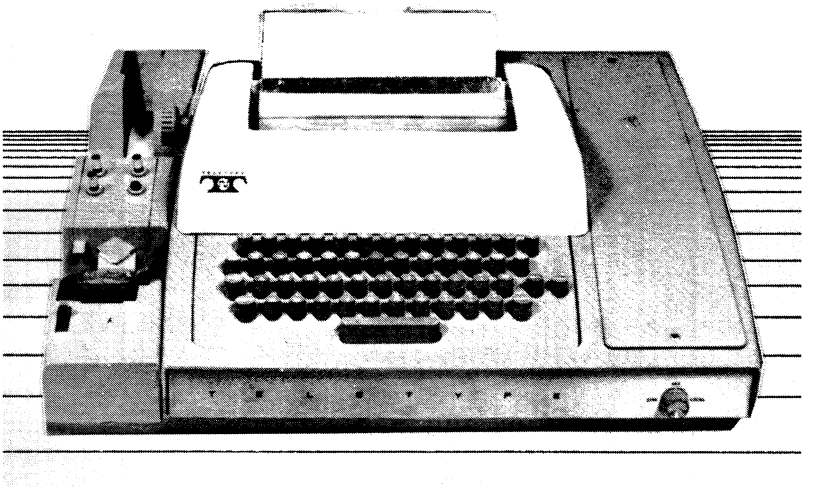
Offensichtlich muß Joe seinen Mikrocomputer mit einigen „Augen" und „Ohren" ergänzen.

Joe leiht sich einen Fernschreiber von einem Freund aus.

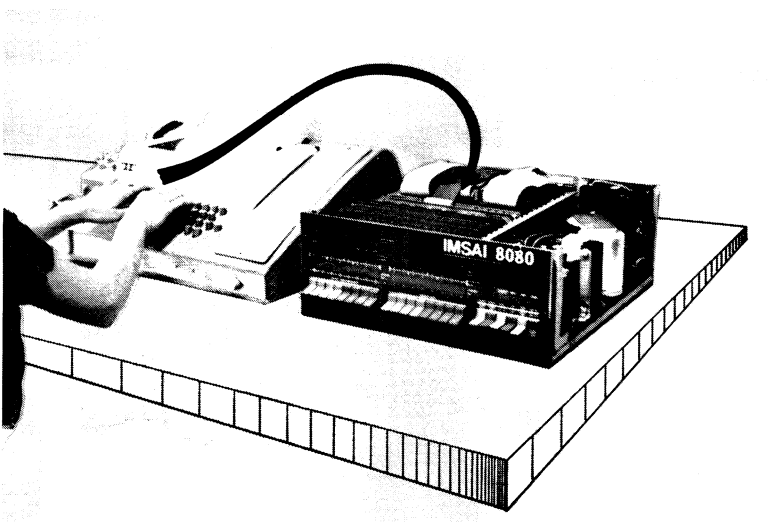
DER FERNSCHREIBER

Ein Fernschreiber (Teletype-Terminal) ist ein sehr interessantes Gerät. Seit mehr als zwanzig Jahren wird es nahezu unverändert eingesetzt. Es wird wahrscheinlich kaum einen anderen Teil eines Computers geben, über den mehr gelästert wurde, und trotzdem existieren wahrscheinlich mehr Fernschreiber als andere Computer-Terminals. Der Grund, weshalb Fernschreiber so populär sind und weiter in Benützung stehen, ist, daß sie so ziemlich alles beinhalten, was wir zur Ergänzung eines Computers be-

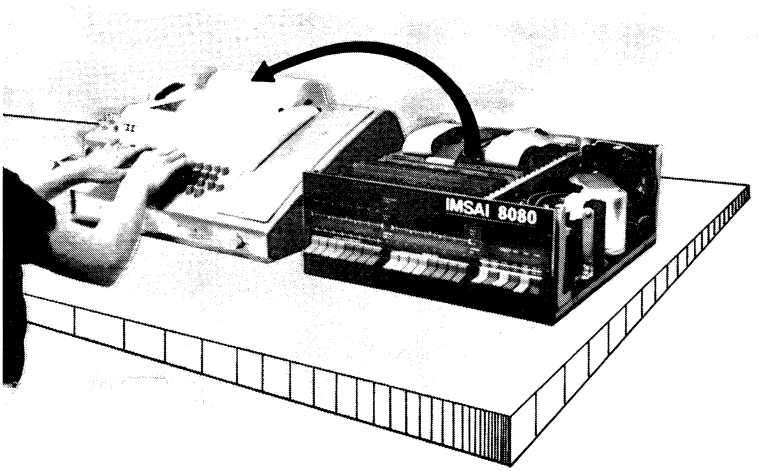
nötigen. Sie sind außerdem sehr zuverlässig und lassen uns selten im Stich. **Sehen wir uns einen derartigen Fernschreiber an:**



Ein Fernschreiber besitzt eine Tastatur, mit der wir Informationen in den Speicher unseres Mikrocomputers eingeben können:



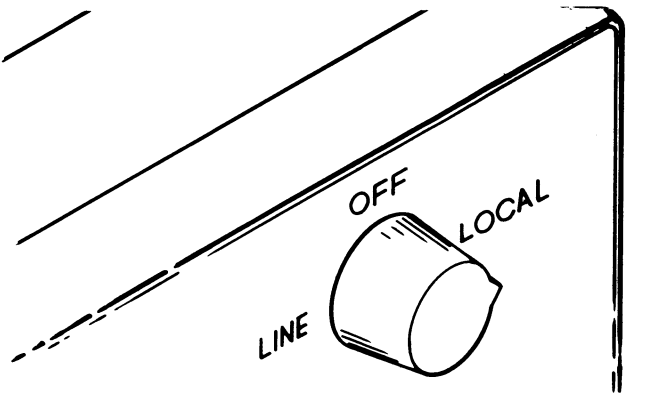
Der Fernschreiber besitzt ferner einen Drucker, mit dem wir Informationen ausdrukken können oder einen Dialog mit dem Mikrocomputer führen können:



Der Drucker eines Fernschreibers besitzt die kombinierten Funktionen einer Videoanzeige und eines Druckers. In Kapitel 1 haben wir besprochen, wie der Drucker einer Schreibmaschine das gleiche tut. Der Drucker des Fernschreibers ist jedoch nicht unbedingt mit der Tastatur des Fernschreibers verbunden.

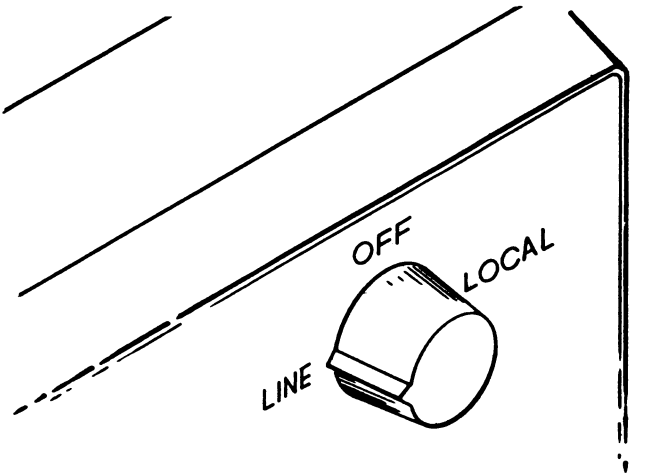
SELBSTÄNDIGER BETRIEB DES FERNSCHREIBERS

Wenn wir wollen, können wir einen Fernschreiber wie eine Schreibmaschine verwenden, d. h. getrennt vom Mikrocomputer. Unter der Tastatur befindet sich häufig ein Schalter, der beim Umschalten auf „local" bewirkt, daß der Fernschreiber in dieser Betriebsart arbeitet:



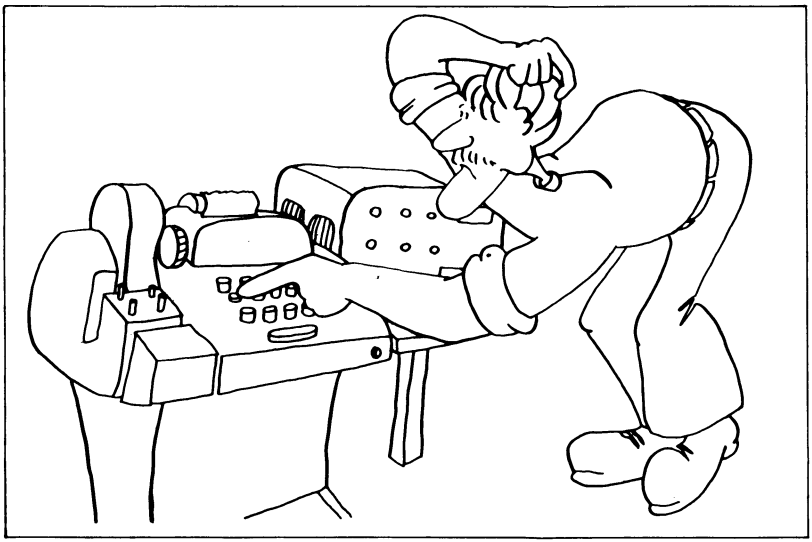
FERNSCHREIBER IN DER LINE-BETRIEBSART

Wenn der Fernschreiber in der „local"-Betriebsart arbeitet, so druckt er jedesmal ein Zeichen, wenn die entsprechende Taste betätigt wird. **Befindet sich der Fernschreiber jedoch in der Betriebsart „Line":**



ECHO-BETRIEB

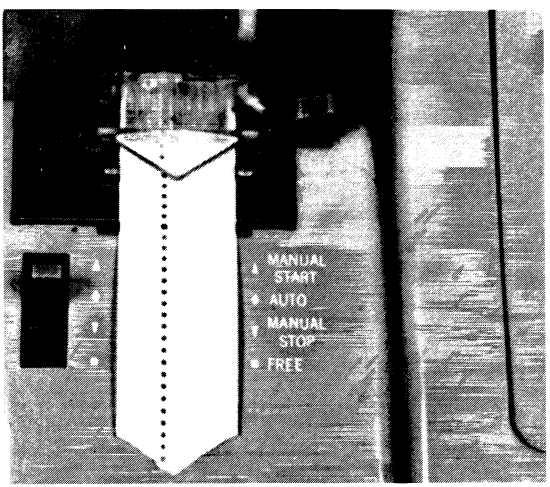
dann wird der Fernschreiber vom Mikrocomputer gesteuert. Wenn wir nun eine Taste drücken, so wird der entsprechende Code zum Mikrocomputer gesendet, dieser liest den Code und bewahrt ihn im Speicher auf, vorausgesetzt es wird ein entsprechendes Programm beim Drücken der Taste ausgeführt. Wenn das Programm das Eingangssignal der Taste auf den Drucker zurückgibt, arbeitet der Fernschreiber scheinbar genauso wie eine Schreibmaschine. Erinnern wir uns daran, daß wir dies als „Echo-Betrieb" bezeichnet haben. Wenn das Programm die an der Tastatur eingegebenen Informationen jedoch nicht zum Drucker zurücksendet, so werden diese auch nicht ausgedruckt und es liegt kein Echo-Betrieb vor. Wir könnten jedoch ein Programm schreiben, das zum Beispiel ein anderes Zeichen als das eingegebene zurückschickt. Wir könnten beispielsweise den jeweils nächsten Buchstaben des Alphabets, also B nach A, C nach B etc. zurückschicken. Alles hängt davon ab, wie wir unseren Mikrocomputer programmiert haben.
Wenn wir eine Taste des Fernschreibers betätigen, während unser Mikrocomputer ein Programm ausführt und nicht annimmt, daß Eingangssignale vom Fernschreiber kommen, dann wird er unsere eingegebenen Daten ignorieren:



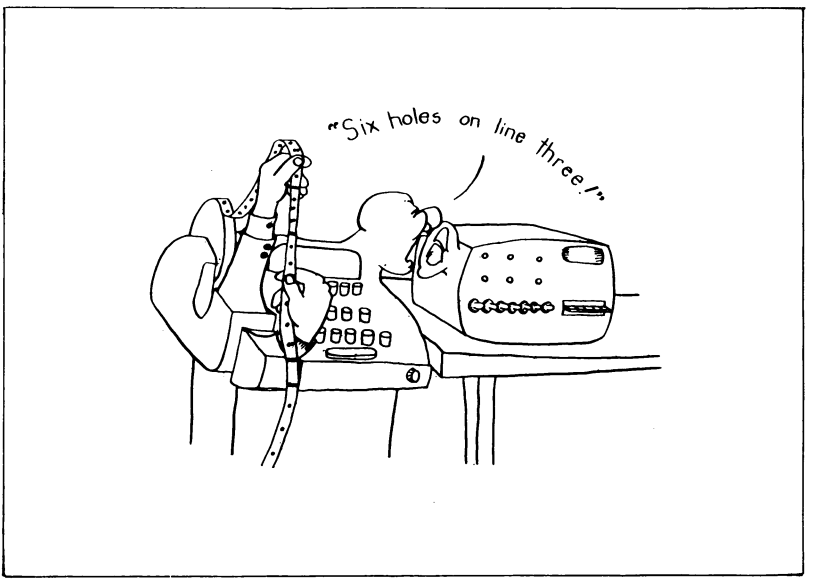
Wenn der Mikrocomputer kein Programm ausführt, bei dem Dateneingaben vom Fernschreiber erwartet werden, dann können wir Tasten drücken so viele wir wollen, es wird nichts geschehen. Die Informationen werden vom Mikrocomputer ignoriert und es wird nichts ausgedruckt.

FERNSCHREIBER-LOCHSTREIFENLESER

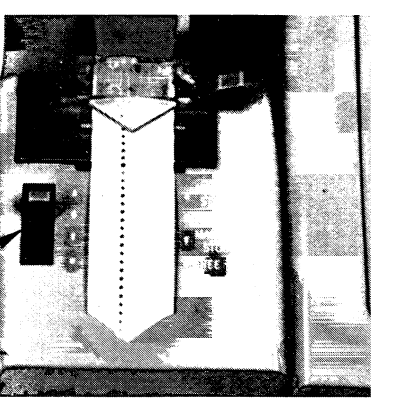
Manche Fernschreiber besitzen einen Lochstreifenleser:



Wenn sich der Lochstreifen durch den Lochstreifenleser bewegt, so stellt dieser das Vorhandensein oder Nichtvorhandensein von Löchern in jeder vertikalen Zeile fest und sendet den entsprechenden Code zum Mikrocomputer:



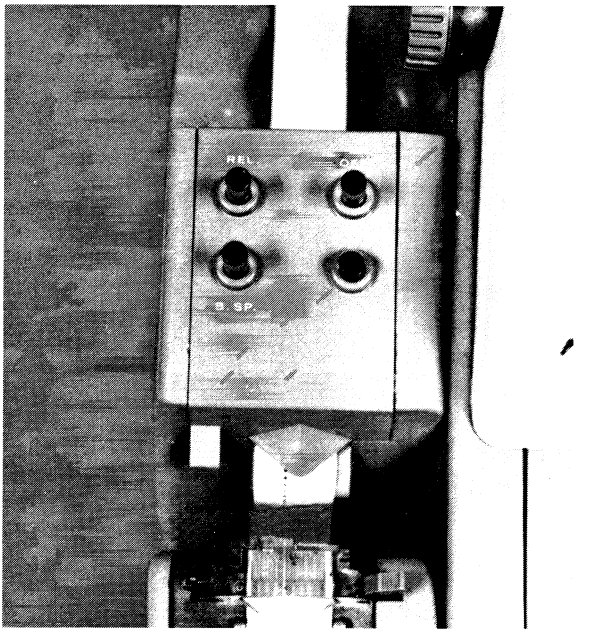
Mit diesem Schalter kann man den Lochstreifenleser starten:



Der RUN-Schalter wird zum manuellen Starten und Stoppen des Lochstreifen-lesers verwendet

Manche Programme, die den Lochstreifenleser verwenden, warten einfach, bis dieser Leser durch Einschalten gestartet wird. Kompliziertere Programme können den Lochstreifenleser selbständig einschalten.

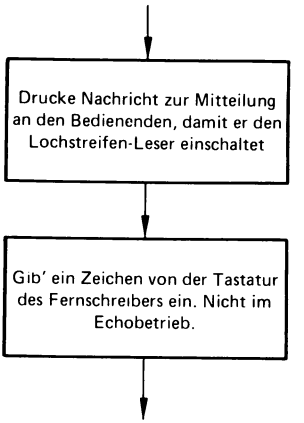
Manche Fernschreiber besitzen auch einen Lochstreifenstanzer. Dieser besitzt beispielsweise vier Tasten oberhalb des Stanzers:



FERNSCHREIBERDRUCKER UND LOCHSTREIFENSTANZER

Mit den Schaltern „Ein" und „Aus" wird, wie schon ihr Name sagt, der Lochstreifenstanzer ein- und ausgeschaltet. **Diese Schalter sind an dem Lochstreifenstanzer erforderlich, da bei einem Fernschreiber der Drucker und der Lochstreifenstanzer als gleiche physikalische Einheit angesehen werden.** Wenn der Lochstreifenstanzer eingeschaltet ist, so wird alles, was gedruckt wird auch gleichzeitig in den Lochstreifen gestanzt. Umgekehrt wird alles, was in den Lochstreifen gestanzt wird auch gleichzeitig

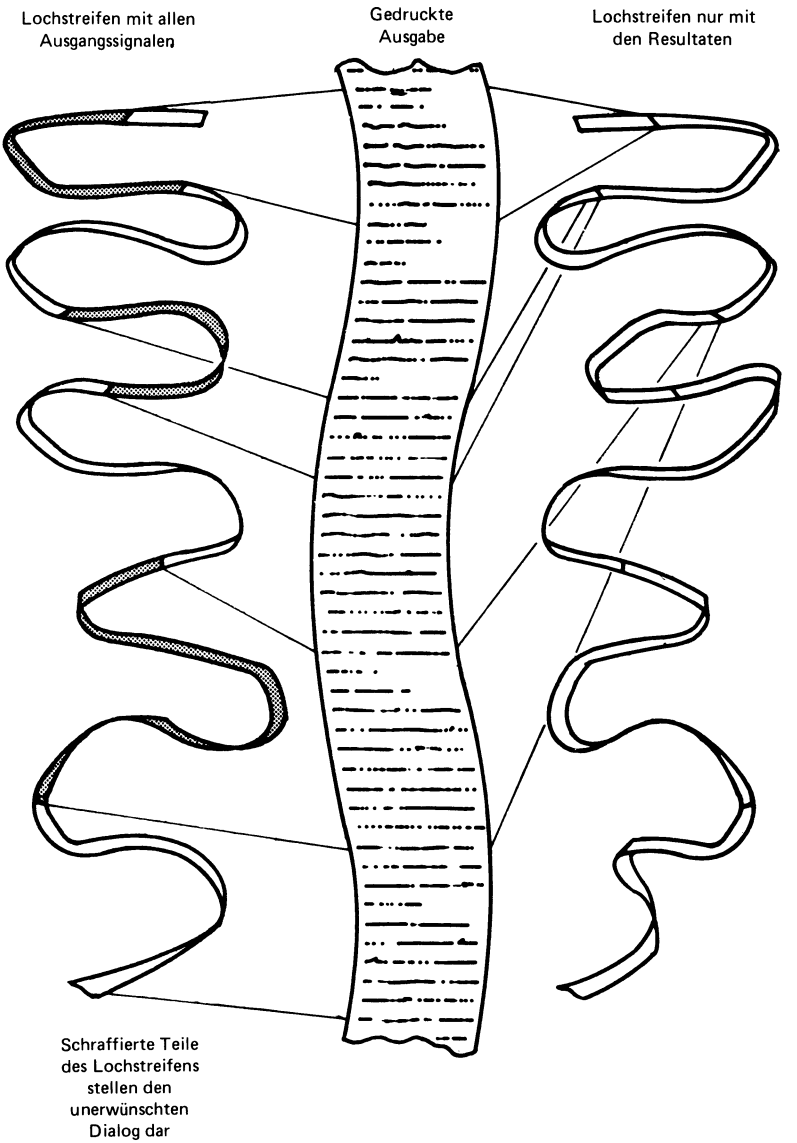
gedruckt. Wenn dies der Fall ist, so lassen wir natürlich den Stanzer ausgeschaltet, außer wir wollen speziell einen gelochten Streifen haben. **Dann werden wir ein Programm schreiben, das uns genügend Zeit gibt, den Lochstreifenstanzer einzuschalten.** Eine typische Programmlogik kann grundsätzlich folgendermaßen aussehen:



Die gezeigte Programmlogik bewirkt, daß eine Nachricht auf den Drucker des Fernschreibers ausgedruckt wird, die uns sagt, daß wir den Lochstreifenstanzer einschalten sollen. Dann stoppt das Programm, bis wir eine Taste – eine beliebige Taste – auf der Tastatur drücken. So haben wir genügend Zeit, um den Lochstreifenstanzer einzuschalten. Das Signal der Taste, die wir auf der Tastatur gedrückt haben, darf nicht zurückgeschickt werden, da das zurückgesandte Zeichen gedruckt und gelocht würde. Das Lochen dieses zurückgesandten Zeichens würde in den Lochstreifen gestanzt und den echten Informationen vorausgehen, die wir ausgeben wollen.

Wenn wir mit dem Lochen des Streifens fertig sind, muß uns unser Programm wieder Zeit geben, den Lochstreifenstanzer auszuschalten. Diesmal können wir jedoch keine Nachricht ausdrucken, die dem Bedienenden sagt, daß er den Lochstreifenstanzer ausschalten soll, da jede derartige Nachricht auch in den Lochstreifen gestanzt würde. Das Mikrocomputer-System wird daher einfach stoppen. Wir müssen dann wissen, daß wir den Lochstreifenstanzer ausschalten sollen und dann irgendeine Taste auf der Tastatur drücken um die Ausführung des Programms fortzuführen.

Warum befassen wir uns mit all diesen umfangreichen Vorkehrungen, nur damit der Lochstreifenstanzer für eine bestimmte Zeit eingeschaltet ist? Die Antwort ist, einfach zur Bequemlichkeit. Wir werden wahrscheinlich den Drucker des Fernschreibers für viele Zwecke verwenden, zum Ausdrucken von Ergebnissen und zum Drucken eines Dialoges während der Dateneingabe und der normalen Computerarbeitsweise. Wenn der Stanzer ständig eingeschaltet ist, so werden wir das ganze Band absuchen müssen, um festzustellen welche Teile des Streifens Ergebnisse erhalten, die wir aufbewahren wollen und jene Teile des Lochstreifens, die nur den Dialog und unnötige Informationen enthalten. Dies kann folgendermaßen dargestellt werden:

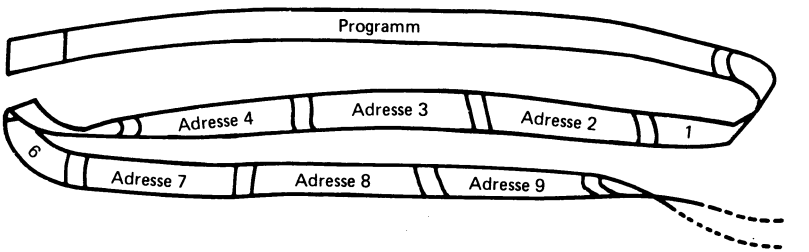


Ein Lochstreifenstanzer enthält meist zwei weitere Tasten. Eine dieser Tasten dient zum Einlegen und Herausnehmen des Lochstreifens. Mit der anderen Taste können wir den Lochstreifen rückwärts bewegen, jeweils einen Schritt beim Drücken dieser Taste.

VERWENDUNG EINES EINFACHEN MIKROCOMPUTER-SYSTEMS

Mit Hilfe dieses Fernschreibers kann Joe nunmehr mit seinem Mikrocomputer eine Menge nützlicher Dinge tun. Als erstes wird Joe ein Programm schreiben, das ihm hilft seine Rechnungen zu bezahlen. Das ist nur gerecht, da der Mikrocomputer sein Budget nicht unbeträchtlich belastet hat.

Wie kann nun um alles in der Welt ein Mikrocomputer Joe helfen seine Rechnungen zu bezahlen? Er kann Joe einfach von lästigen Schreibarbeiten befreien. **Joe's Programm enthält eine Liste von Namen und Adressen, denen er regelmäßig Zahlungen leisten muß:** Der Postbehörde für die Telefonrechnungen, der Versicherungsgesellschaft, die Fernsehgebühren etc. **Joe entwickelt sein Programm auf einem Lochstreifen, der etwa folgendermaßen aussehen könnte:**



Nun dauert es nicht lange, bis Joe herausfindet, daß das Herumhantieren mit den Programmen zur Steuerung des Fernschreibers eine Zeitverschwendung ist. Erinnern wir uns daran, daß die Ausführung eines Programmes im Mikrocomputer erforderlich ist, um die Informationen anzunehmen, die wir über die Tastatur des Fernschreibers oder einen Lochstreifenleser eingeben, sowie zur Zurückgabe dieser Informationen an den Drucker. **Wenn Joe seinen Fernschreiber an den Mikrocomputer anschließt, so sind im Mikrocomputer keine Programme enthalten, die sich um den Fernschreiber kümmern. Was also Joe zunächst tun muß, ist, solche Programme zu schreiben und sie in den Speicher des Mikrocomputers mittels der Schalter auf der Frontplatte einzugeben.**

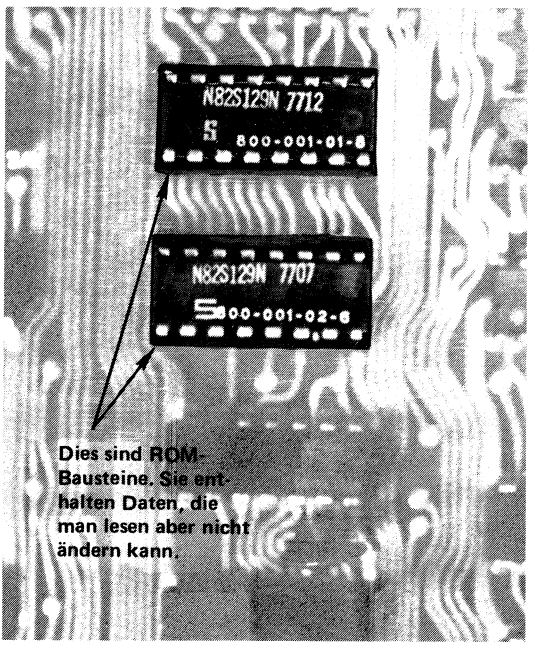
Das Problem bei diesem Verfahren liegt jedoch darin, daß jedes Mal wenn Joe den Netzschalter des Mikrocomputers ausschaltet, auch alles was sich im Speicher des Mikrocomputers befindet, gelöscht wird. Daher muß Joe jedesmal wenn er seinen Mikrocomputer einschaltet, den mühsamen Vorgang der Eingabe des Steuerprogramms für den Fernschreiber über die Schalter auf der Frontplatte durchführen.

In Kürze wird Joe aufhören, das Steuerprogramm für den Fernschreiber überhaupt als Programm anzusehen, es wird zu einem notwendigen Teil seines Mikrocomputer-Systems.

Joe geht zurück zum Verkäufer seines Mikrocomputers und sucht Hilfe und er erhält sie.

FESTWERTSPEICHER

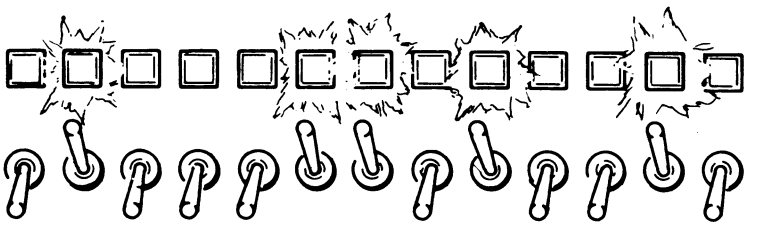
Jeder, der einen Fernschreiber verwendet, braucht ein zugehöriges Steuerprogramm. Joe entdeckt, daß man ein derartiges Programm, das sich in einem kleinen Speicherchip befindet, kaufen kann, dessen Inhalt niemals verlorengehen kann. **Dieses Programm wird ein „Urlader" (bootstrap loader) genannt. Joe kauft ein derartiges Programm und erhält folgendes:**



Der englische Ausdruck „bootstrap" (deutsch etwa: Stiefelschlaufe) stammt von dem Konzept, bei dem sich ein Mann selbst aus einem Loch befördert, indem er an seinen Stiefelstrippen zieht. Über dieses Ladeprogramm startet sich der Computer selbst.

Das Ladeprogramm ist in einem Chip eines Nur-Lesespeichers (oder Festwertspeichers) aufbewahrt, das, wie der Name sagt, ein Speicherbaustein ist, dessen Inhalt man lesen, jedoch in den man nicht einschreiben kann. Der Inhalt eines Festwertspeichers ist für immer festgelegt und kann nicht geändert werden.

Um den Unterschied zwischen einem Festwertspeicher und einem Schreib-/Lesespeicher zu verstehen, stellen wir uns vor, daß der Speicherchip aus Tausenden von Schaltern mit den zugehörigen Lampen besteht:



Ein Speicherchip entspricht nur grundsätzlich derartigen Schaltern und Lampen. In Wirklichkeit enthält der Speicherchip jedoch mikroskopisch kleine elektronische Strukturen, die natürlich nicht wie derartige Schalter oder Lampen aussehen.

Wenn wir in einen Speicherbaustein einschreiben, entspricht unsere Tätigkeit dem Betätigen von bestimmten Schaltern. Über jedem Schalter, der auf „Ein" steht, brennt eine Lampe.

Wenn wir aus einem Speicherbaustein Informationen lesen, so entspricht dies einem Überprüfen der Lampen, um festzustellen ob sie „Ein" oder „Aus" sind.

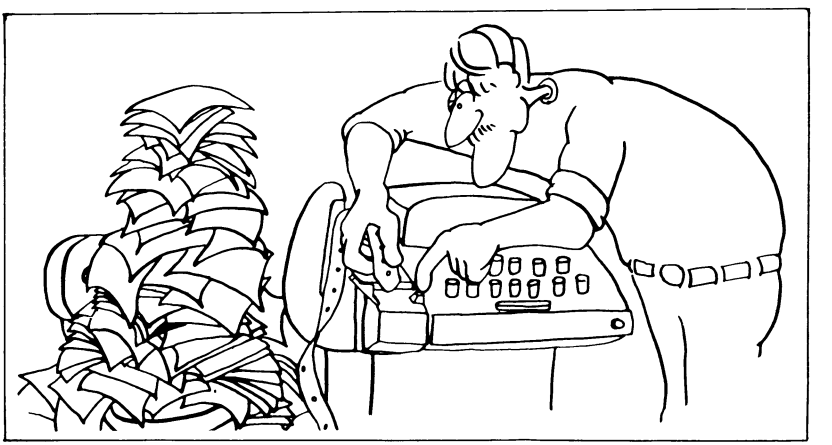
Nun nehmen wir an, daß wir in den Speicherchip einschreiben, indem wir die entsprechenden Schalter ein- und ausschalten, wenn wir jedoch sicher sind, daß alle Schalter in der richtigen Position sind, entfernen wir die Schalter. Nun bleiben die Lampen ständig eingeschaltet. Das entspricht im Prinzip dem Vorgang in einem Festwertspeicher.

Der Vorteil eines Festwertspeichers liegt darin, daß er seinen Inhalt behält, egal was wir auch damit anfangen, außer wir zerstören ihn.

ROM

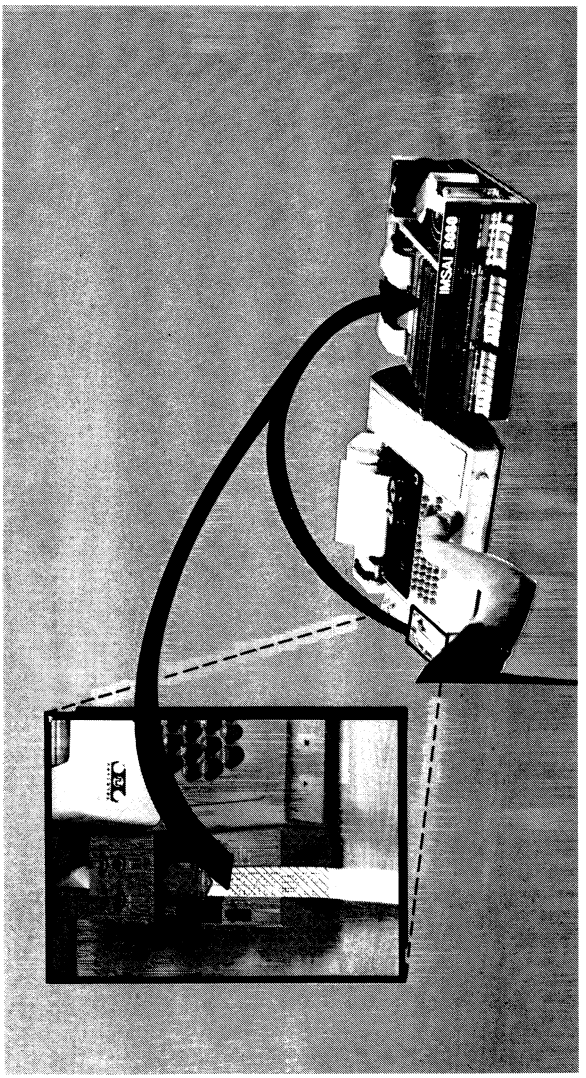
Ein Festwertspeicher wird häufig ein ROM (Read Only Memory) genannt.

Wenn nun Joe seine Rechnungen bezahlen möchte, stapelt er die Rechnungen aufeinander und lädt dann den Lochstreifen für die Bezahlung der Rechnungen in den Lochstreifenleser des Fernschreibers:

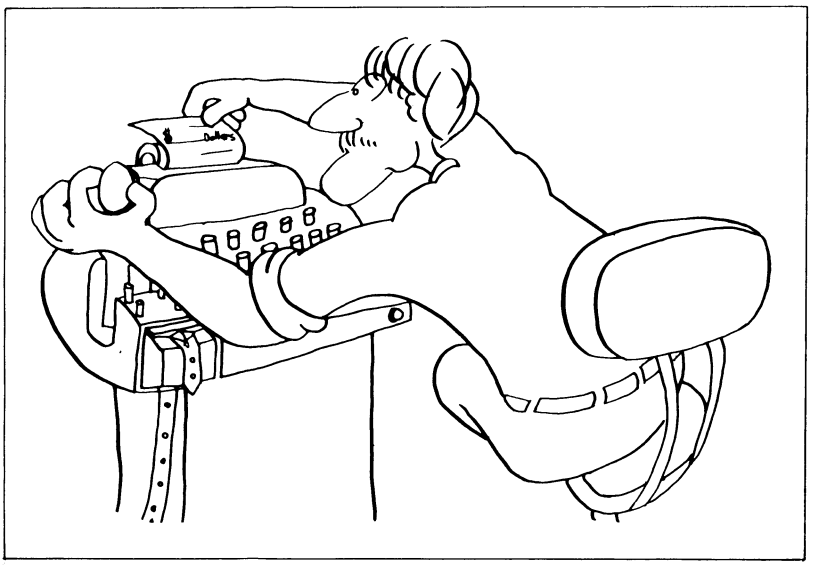


Zunächst betätigt Joe ein paar Schalter, indem er den Anweisungen sorgfältig folgt, und das Laderprogramm im Festwertspeicher (ROM) übernimmt das weitere. Ein Befehl nach dem anderen des Ladeprogrammes gelangt in den Mikrocomputer, veranlaßt diesen den Lochstreifenleser einzuschalten und liest dann den Lochstreifen. **Das Laderprogramm ist intelligent genug um zu wissen, wann das Ende des Programmes auf dem Papierstreifen erreicht ist. An diesem Punkt schaltet es den Lochstreifenleser aus.**

Joe's Programm ist nunmehr in den Speicher des Mikrocomputers eingelesen worden:



Das Ladeprogramm weiß, daß seine Arbeit erledigt ist und überläßt nun die Steuerung dem Programm von Joe. Neue Befehle von Joe's Programm gelangen einer nach dem anderen in den Mikrocomputer und bewirken, daß dieser Joe's Aufgaben erledigt.
Zu Beginn möchte Joe jedoch, daß sein Programm nichts tut. Es soll warten, bis er einen Streifen mit Überweisungen in den Drucker des Fernschreibers eingeführt hat:

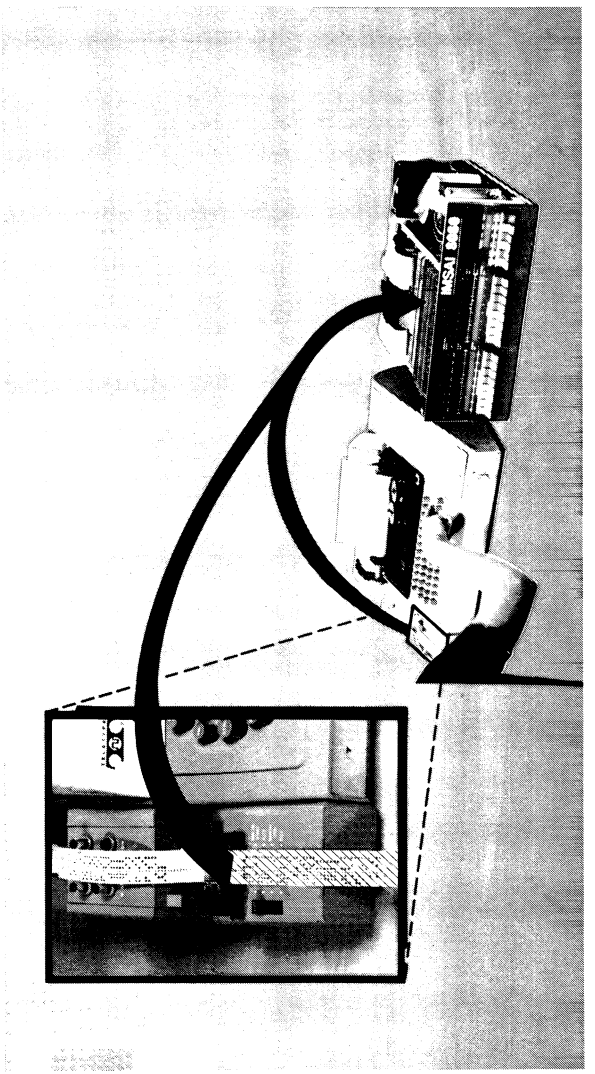


EINGABE ÜBER DIE TASTATUR OHNE ECHO

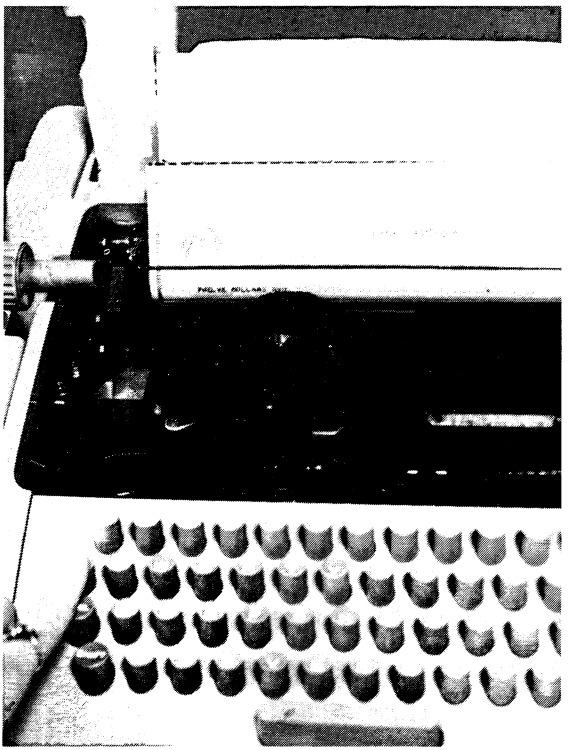
Um Joe genügend Zeit zu lassen, damit er seine Überweisungen in den Drucker des Fernschreibers einlegen kann, besitzt sein Programm eine Logik ähnlich der, die den Lochstreifenstanzer eines Fernschreibers ein- und ausschaltet.
Der einzige Weg jedoch, über den Joe's Programm wissen kann, daß sich die Formulare im Drucker befinden, besteht darin, daß Joe irgendeine Taste auf der Tastatur des Fernschreibers betätigt. Er schreibt sein Programm derart, daß es für eine Eingabe über die Tastatur ohne Echo wartet. Natürlich, wenn er eine Taste des Fernschreibers drückt und das Programm das Zeichen zurücksendet, so wird das gedruckte Zeichen auf seinem Überweisungsformular erscheinen und dieses unbrauchbar machen.
Joe erweitert daher seine Programmschritte, damit er genügend Zeit für das Einlegen der Formulare in den Drucker hat. Joe's Programm ist derart aufgebaut, daß es die Zeichen, die über die Tastatur eingegeben werden, überprüft und feststellt ob sich hier der Buchstabe „A" befindet. Nur wenn der Buchstabe „A" eingegeben wurde, wird das Programm fortgesetzt. Wenn irgendein anderer Buchstabe eingegeben wird, so wartet das Programm einfach, bis ein neues "A" über die Tastatur des Fernschreibers ankommt.

Wenn nun Joe zufällig irgend eine Fernschreibtaste drückt, wird er das Programm nicht unabsichtlich starten. Nur das Drücken der „A"-Taste wird das Programm in Gang setzen. Joe ist gegen dumme Fehler geschützt, wenn er sein Programm ablaufen läßt.

Sobald der Buchstabe „A" in die Tastatur eingetippt ist, schaltet Joe's Programm den Lochstreifenleser des Fernschreibers ein um den ersten Namen und Adresse einzulesen, der, wie wir uns erinnern, auf dem Papierstreifen direkt nach dem Programm gespeichert ist:



Joe's Programm liest diesen Namen und Adresse vom Lochstreifen und speichert sie als Daten in den Speicher.
Als nächstes bewegt Joe's Programm das Formular im Drucker des Fernschreibers weiter, bis der entsprechende „DM-Betrag" sich direkt hinter dem Druckelement befindet. Das Programm veranlaßt den Mikrocomputer zu warten, bis Joe den entsprechenden Betrag eingetippt hat, zuerst in Worten, dann als Zahl:



TASTATUREN

Sehen wir uns einmal an, was der Mikrocomputer jedesmal tut, wenn Joe ein Zeichen in die Tastatur des Fernschreibers eintippt. Erinnern wir uns daran, daß im Verhältnis zum Mikrocomputer Joe natürlich sehr langsam ist. Die größte Geschwindigkeit, mit der Joe tippen kann, beträgt etwa drei Zeichen pro Sekunde. **Ein typischer Mikrocomputer führt einen Befehl in etwa 5 Mikrosekunden aus, d.h., daß er etwa 200000 Befehle pro Sekunde erledigen kann.**

$$5 \text{ Mikrosekunden} = \frac{5}{1000000} \text{ Sekunde}$$

$$\text{Befehle pro Sekunde} = \frac{1000000}{5} = 200000$$

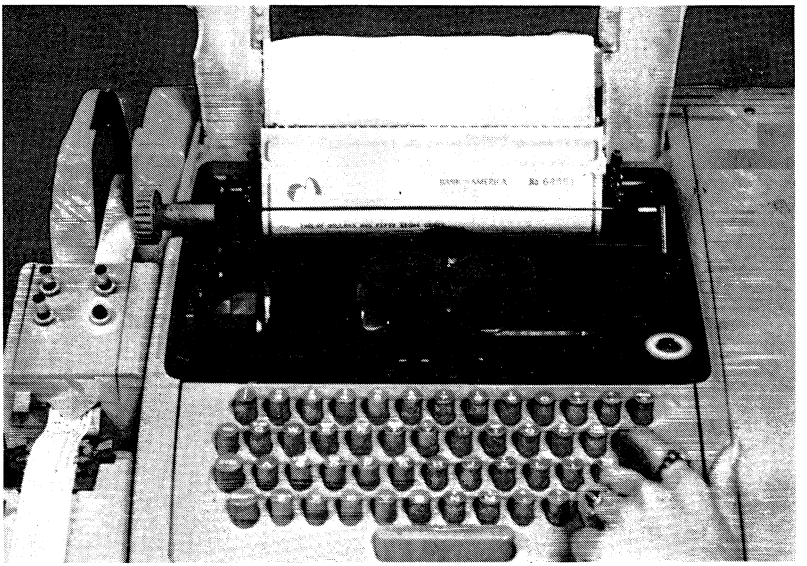
Da Joe drei Anschläge pro Sekunde schafft,

$$\text{Befehle pro Anschlag} = \frac{200000}{3} = 67777$$

Der Mikrocomputer kann also 67777 Befehle im Durchschnitt zwischen jedem Anschlag ausführen. So hat natürlich Joe's Programm genügend Zeit um das ankommende Zeichen zu lesen und es zurückzuschicken, dies erfordert weniger als 20 Befehle.

FEHLERBESEITIGUNG

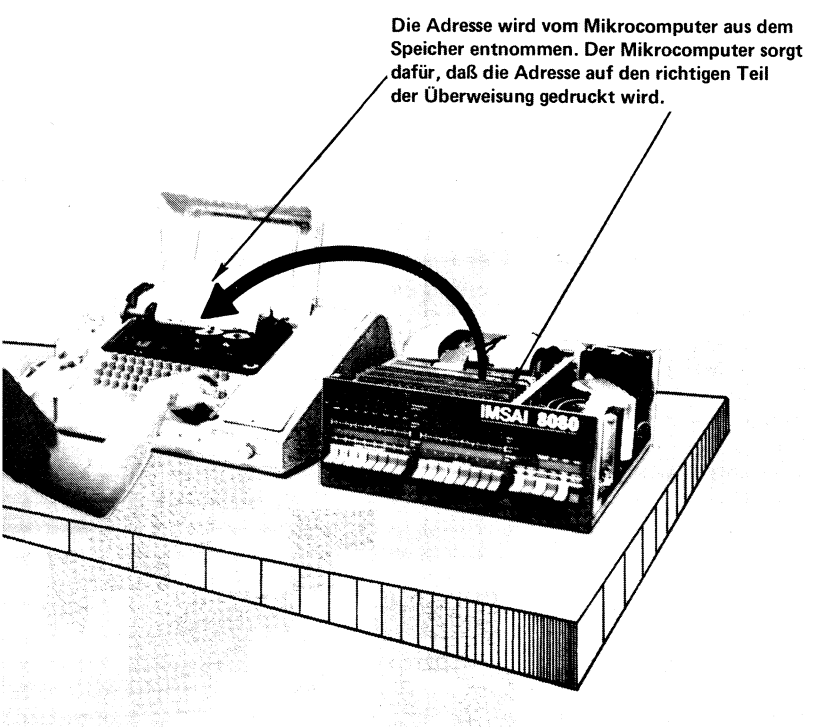
Da nun eine Menge überschüssiger Zeit vorhanden ist, entschließt sich Joe die Tastatur als Hilfsmittel bei verschiedenen Schwierigkeiten zu verwenden, denn wenn man einen Fehler machen kann, so macht man ihn auch sicher. **Joe verwendet also die Tastatur zur Kontrolle seines Programmes.** Bei Empfang eines Zeichens von der Tastatur prüft Joe's Programm ob das Zeichen ein „Escape" (Code-Umschaltung), eine spezielle Taste auf der Fernschreibtastatur, ist.



ERNEUTER START

Bei Feststellung eines „Escape" (= Code-Umschaltung) führt das Programm die Formulare im Drucker des Fernschreibers zum Beginn der nächsten Überweisung, führt den Drucker zum Anfang der Zeile zurück und beginnt dann neuerlich mit den momentanen Namen und Adressen. So kann Joe jederzeit, wenn er einen Fehler gemacht hat, einfach abbrechen und neuerlich starten.

Als nächstes prüft Joe's Programm auf das Zeichen „Carriage Return" (Wagenrücklauf). **Wird ein derartiges Zeichen festgestellt, so liest das Programm von Joe den Namen und die Adresse, die sich derzeit im Speicher befinden und druckt sie auf das Formular:**



FLUSSDIAGRAMM FÜR DIE PROGRAMMLOGIK
FLUSSDIAGRAMM-SYMBOLE

Um das Verständnis der Logik von Joe's Programm zu fördern, ohne im einzelnen darauf einzugehen wie Joe sein Programm geschrieben hat, sehen wir uns Bild 2.1 an, das ein Flußdiagramm der Programmlogik zeigt. Die Rechtecke, Kreise und Rhomben, die in Bild 2.1 verwendet werden, sind ein Teil eines Standardsatzes von Symbolen für Flußdiagramme, die allgemein in gleicher Weise verwendet werden. Der vollständige Satz dieser Symbole ist in Anhang B dargestellt.

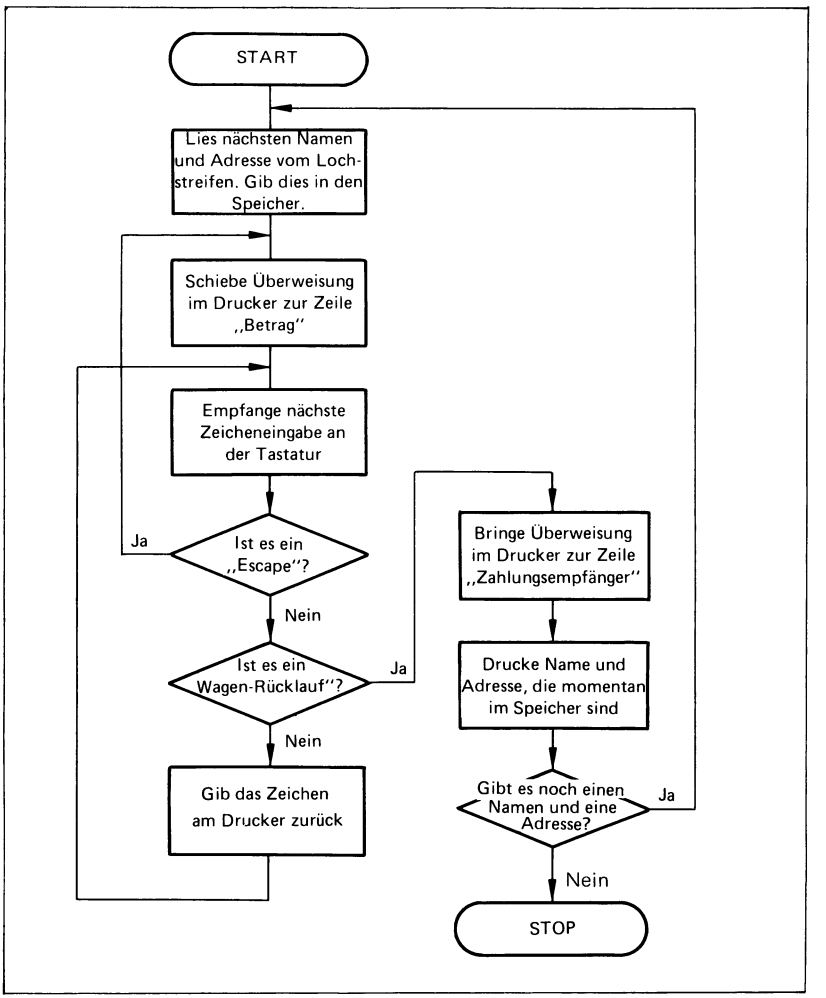


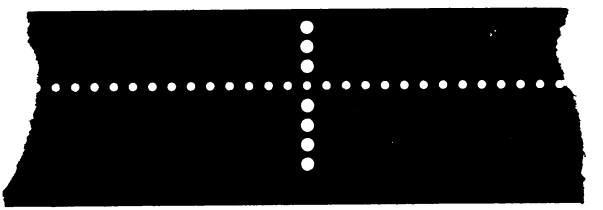
Bild 2.1 Ein Flußdiagramm für Joe's Programm zur Zahlung von Rechnungen

Wenn wir uns Joe's Programm ansehen, so besteht dieses aus einer Folge von Hauptschritten, in denen ein Name und eine Adresse gelesen wird sowie eine Überweisung in jedem Hauptschritt ausgeschrieben wird. **Während eines Hauptschrittes geschieht folgendes:**

1) Der Mikrocomputer bringt das Formular in die richtige Position, damit Joe den DM-Betrag eingeben kann.
2) Joe tippt den DM-Betrag ein.
3) Joe tippt ein „Wagenrücklauf" zur Beendigung des DM-Betrages ein. Der Mikrocomputer führt automatisch den Wagenrücklauf aus und druckt dann Name und Adresse des Zahlungsempfängers aus.
4) Der Mikrocomputer liest den nächsten Namen und Adresse vom Lochstreifen und führt die Formulare zur nächsten Zeile „DM-Betrag" in der nächsten Überweisung weiter.

Wenn Joe einen Fehler macht, so drückt er die „Escape"-Taste. Damit beginnt ein neuerliches Ausfüllen eines Überweisungsformulares, d. h. dasjenige, das fehlerhaft ausgefüllt wurde.

Wie weiß Joe's Programm, wenn keine weiteren Namen und Adressen vorhanden sind? **Joe's Programm zum Lesen des Lochstreifens überprüft die ankommenden Zeichen. Joe wählt eine Reihe von acht Löchern als spezielles Zeichen für „Ende der Daten". Sobald das Programm eine Reihe von acht Löchern entdeckt, nimmt es an, daß alle Namen und Adressen gelesen wurden und stoppt daher:**



Nunmehr hat Joe eine Reihe von Überweisungen, die er unterzeichnen und in entsprechende Fensterkuverts einlegen kann. Es brauchen keine Namen und Adressen auf die Umschläge geschrieben zu werden und Joe hat eine Menge Zeit gespart.

EINIGE ANWENDUNGEN FÜR MIKROCOMPUTER

Joe's Programm für die Bezahlung von Rechnungen ist nur ein einfaches Beispiel, wie man ein Mikrocomputer-System verwenden kann. Aber es zeigt, wie die verschiedenen Teile eines Mikrocomputer-Systems zur Ausführung eines Programmes zusammenarbeiten.

Tatsächlich wird sich dieses Programm nur lohnen, wenn Joe eine große Anzahl von Rechnungen zu zahlen hat, andernfalls wird es wahrscheinlich schneller gehen, wenn er die Überweisungsformulare von Hand ausfüllt. Die Programmierung des Mikrocomputers für diese Aufgabe wird wahrscheinlich nicht viel Zeit sparen. Wir haben dieses Programm für die Bezahlung von Rechnungen als Beispiel verwendet, nicht um

zu zeigen, daß ein Mikrocomputer-System eine sehr ökonomische Anschaffung ist, sondern um zu zeigen, wie man ein Mikrocomputer-System zum Arbeiten bringt. **Wenn** es auch für Joe, der etwa 10 bis 15 Rechnungen pro Monat bezahlt keine **Rechtfertigung** gibt, diese Zahlungen mittels eines Computers auszuführen, so kann **jedoch** das gleiche von Joe geschriebene Programm Hunderte von Rechnungen ebenso **leicht** handhaben wie seine 10 oder 15. Wenn Hunderte von Rechnungen zu zahlen sind, so wird das Mikrocomputer-System eine große Zeitersparnis bieten, verglichen mit einer Ausführung dieser Aufgabe von Hand.

Aber ein ganz wesentlicher Punkt ist hierbei zu beachten: Durch einfache Änderung des Lochstreifens kann Joe sein ganzes Mikrocomputer-System für eine vollkommen andere Aufgabe einsetzen. Wenn das Mikrocomputer-System für eine Aufgabe nicht wirtschaftlich verwendet werden kann, so ist dies sofort der Fall, wenn es 10 ausführt. Wenn beispielsweise Joe DM 2000,– für sein Mikrocomputer-System ausgegeben hat, das ein Programm auf einem Lochstreifen enthält, so kostet das Mikrocomputer-System die DM 2000,– für eine Aufgabe. Bei Verwendung des Mikrocomputer-Systems für 10 Programme kostet eine Aufgabe jedoch nur mehr DM 200,– je Programm. Je mehr Verwendungsmöglichkeiten Joe für sein Mikrocomputer-System findet, desto niedriger werden die effektiven Kosten pro Anwendung. Und das ist es, was Mikrocomputer-Systeme so populär macht. Sie tun alles, was wir über ein Programm definieren können. Die einzige Grenze für die Zahl der geschriebenen Programme ist die Zeit, die das Mikrocomputer-System zu deren Ausführung benötigt.

Joe denkt noch an zahlreiche andere Anwendungen für sein Mikrocomputer-System. Zusätzlich zur Bezahlung seiner Rechnungen kann er sein Adressenbuch auf dem neuesten Stand halten oder sein Scheckbuch kontrollieren, nur um einige typische kaufmännische Probleme zu erwähnen.

Es gibt natürlich auch zahlreiche nicht-kaufmännische Dinge, die der Mikrocomputer ausführen kann. Er eignet sich beispielsweise auch für Spiele. Mit einem Mikrocomputer-System kann man einfache Spiele auf dem Bildschirm eines Fernsehempfängers ausführen oder es kann für komplexe Spiele wie Schach programmiert werden. Joe plant sogar völlig neue Spiele zu erfinden.

Aber im Stillen denkt Joe bereits daran noch interessantere Anwendungen für seinen Mikrocomputer zu finden, z. B. synthetische Musik herzustellen. Man kann Lautsprecher direkt von einem Mikrocomputer steuern. Daher kann man Programme schreiben, die Töne erzeugen. Diese Möglichkeiten interessieren Joe besonders. Aber dafür muß er seinen Mikrocomputer noch besser verstehen lernen.

KAPITEL 2

FRAGEN

1. Bevor ein Programm für einen Mikrocomputer in eine Folge von Zahlen umgewandelt wird, schreibt man es unter Verwendung einer_____ auf ein Blatt Papier.
2. Das Programm, das nun aus einer Folge von Zahlen besteht, muß in den_____ __des Mikrocomputers geladen werden.
3. Besitzt man keine Tastatur, so kann man das Programm in den Speicher des Mikrocomputers mit Hilfe der auf der Frontplatte befindlichen_____ laden.
4. Mittels dieser Schalter kann man nun auch die gespeicherten Informationen überprüfen, wobei eine Anzeige auf den ebenfalls auf der Frontplatte befindlichen____ _____ erfolgt.
5. Jedes Programm muß im allgemeinen auf _____ überprüft werden, bevor es ordnungsgemäß arbeitet.
6. Zur Eingabe und Ausgabe von Informationen für einen Mikrocomputer kann man sehr einfach die Tastatur und den Drucker eines _____ verwenden.
7. Längere Programme kann man auch über den_____ des Fernschreibers eingeben.
8. Die meisten Fernschreiber besitzen auch einen_____ zur Herstellung eines Lochstreifens.
9. Ein integrierter Baustein, in dem bestimmte Informationen fest aufbewahrt sind und auch ohne anliegende Stromversorgung nicht verlorengehen, nennt man einen _____ oder englisch _____ .
10. Zum selbständigen Laden eines Programmes in einen Mikrocomputer von einem Lochstreifen verwendet man zweckmäßigerweise einen sogenannten _____ , den man auf einem Festwertspeicher käuflich erwerben kann.
11. Wenn man den Ablauf eines Programmes übersichtlich darstellen will, so verwendet man hierzu ein sogenanntes _____ .
12. Der Standardsatz von Symbolen für Flußdiagramme besteht im wesentlichen aus _____ , _____ und _____ .
13. Der wesentlichste Vorteil eines Mikrocomputer-Systems besteht darin, daß es für völlig verschiedene Aufgaben verwendet werden kann und hierbei nur das jeweilige_____ ausgetauscht werden muß.

ANTWORTEN, 2. KAPITEL

1. Programmiersprache
2. Speicher
3. Schalter
4. Lampen
5. Fehler
6. Fernschreibers
7. Lochstreifenleser
8. Lochstreifenstanzer
9. Festwertspeicher, ROM
10. Urlader
11. Flußdiagramm
12. Kreisen, Rechtecken, Rhomben
13. Programm

Kapitel 3
MIKROCOMPUTER-SYSTEM-KOMPONENTEN — WAS WIR SEHEN IST NICHT IMMER DAS WAS WIR BEKOMMEN

Eines Tages passierte Joe etwas Unangenehmes. Sein Freund wollte den Fernschreiber zurück.
Sein Mikrocomputer-System wieder ohne „Augen" und „Ohren" zu verwenden steht außer jeder Frage. Schließlich soll der teuere Mikrocomputer nicht nur zum Beobachten an- und ausgehender Lampen dienen, sondern Joe hat auch viel Zeit zum Schreiben nützlicher Programme investiert. Joe schnallt daher seinen Gürtel enger, nimmt sein Scheckbuch und kehrt zu seinem Computer-Lieferanten zurück.

Joe möchte sein Mikrocomputer-System erweitern.
Die Möglichkeiten sind für Joe verwirrend, und es würde uns wahrscheinlich ebenso gehen, bis wir genauer verstehen was wir hier vor uns haben. Wir werden dann etwas klarer sehen.
Wenn wir ein Mikrocomputer-System zusammenstellen, müssen wir erst die Funktionen auswählen, die unser Mikrocomputer ausführen soll: Das Empfangen von Dateneingaben, Drucken der Resultate und Speichern von Informationen — all dies sind „Funktionen". Als nächstes müssen wir die gewünschte physikalische Einheit auswählen (und sie uns leisten können) um die gewünschte Funktion auszuführen. Beispielsweise können wir Informationen auf Lochstreifen, Kassetten oder Disketten speichern. Jedesmal, wenn wir eine bestimmte physikalische Einheit wählen, müssen wir eine Entscheidung über die Möglichkeiten und zusätzlichen Eigenschaften treffen, für die wir gewillt sind zu bezahlen.
Wir gehen nun daran die verschiedenen Komponenten, die wir für unser Mikrocomputer-System kaufen können, zu identifizieren, die Funktionen, die sie innerhalb eines Mikrocomputer-Systems ausführen und die gewöhnlich zusätzlich angebotenen Möglichkeiten.
Wir beginnen damit, uns die ausgeführten Funktionen anzusehen und die physikalischen Komponenten, die wir zur Ausführung jeder Funktion erwerben können.

PHYSIKALISCHE UND LOGISCHE EINHEITEN IN MIKROCOMPUTER-SYSTEMEN

Wenn wir nochmals Bild 1.1 in Kapitel 1 ansehen, so werden wir dort folgende Komponenten abgebildet finden:

1) Den Mikrocomputer selbst.
2) Eine Tastatur
3) Eine Videoanzeige
4) Einen Drucker
5) Einen Massenspeicher.

Erinnern wir uns daran, wie in Kapitel 1 Aufzeichnungen und Dateien entweder als „physikalisch" oder „logisch" geschrieben wurden? Die „physikalische" Aufzeichnung oder Datei ist die Form, in der Informationen tatsächlich von einem Massenspeichergerät gespeichert werden, während eine „logische" Aufzeichnung oder Datei der Weg ist, in dem ein Mikrocomputeranwender die gespeicherten Informationen ansieht und verwendet. Wir können dieses Konzept um einen weiteren Schritt erweitern, indem wir die fünf Komponenten des in Bild 1.1 gezeigten Mikrocomputer-Systems nehmen und sehen, welche Funktionen sie ausführen.

LOGISCHE EINHEIT FÜR INFORMATIONSEINGABE

Die Tastatur wird zu einer logischen Einheit für „Informations-Eingaben". Die logische Einheit für die Informationseingabe muß nicht unbedingt eine Tastatur sein. Es kann die Frontplatte des Mikrocomputers mit den Schaltern sein, oder jede andere beliebige Hardware, die imstande ist Informationen zu lesen, sogar ein Lochstreifenleser.

LOGISCHE EINHEIT FÜR MITTEILUNGEN AN DEN BEDIENENDEN

Die Videoanzeige wird zu einer logischen Einheit für „Mitteilungen an den Bedienenden". Es kann eine Videoanzeige sein, ein Drucker eines Fernschreibers, ein Drucker einer Schreibmaschine oder jede beliebige andere Hardware, die imstande ist Nachrichten oder Mitteilungen in für den Menschen lesbarer Form auszugeben.

LOGISCHE EINHEIT FÜR DIE AUSGABE VON RESULTATEN

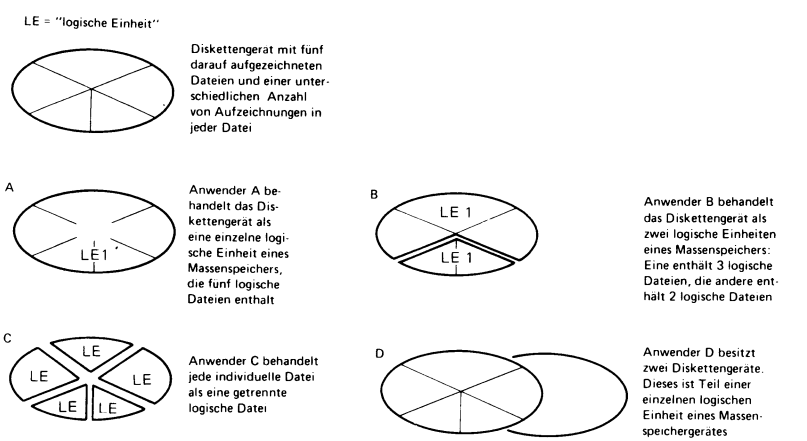
Der Drucker wird eine logische Einheit zur „Ausgabe von Resultaten". Wir haben gesehen, wie Resultate durch eine Schreibmaschine, den Drucker eines Fernschreibers oder eines alleinstehenden Druckers dargestellt werden. Es gibt jedoch viele weitere Möglichkeiten, mit denen wir unsere Resultate ausgeben können. Beispielsweise könnten wir Resultate auf eine Kassette schreiben und daran denken, die Resultate später auszudrucken, wenn das Mikrocomputer-System gerade für diesen Zweck zur Verfügung steht.

LOGISCHE EINHEIT FÜR DIE MASSENSPEICHERUNG VON INFORMATIONEN

Bisher haben wir drei Ausführungen für die logische Einheit zur „Massenspeicherung von Informationen" gesehen: Diskettengeräte, Kassettenrekorder und Lochstreifengeräte.

Ferner haben wir Geräte mit starren Platten als Alternative zu den Diskettengeräten betrachtet.

Ein Gerät zur Speicherung großer Mengen von Informationen wurde nicht immer als eine einzelne logische Einheit behandelt. Manchmal können individuelle Dateien zu individuellen logischen Einheiten werden, oder eine Gruppe von logischen Dateien bildet eine logische Einheit. Daher können aus einer physikalischen Einheit zahlreiche logische Einheiten werden. Andererseits kann mehr als eine physikalische Einheit eine einzelne logische Einheit bilden. Jede denkbare Beziehung zwischen physikalischen und logischen Einheiten ist möglich. Dies kann folgendermaßen dargestellt werden:



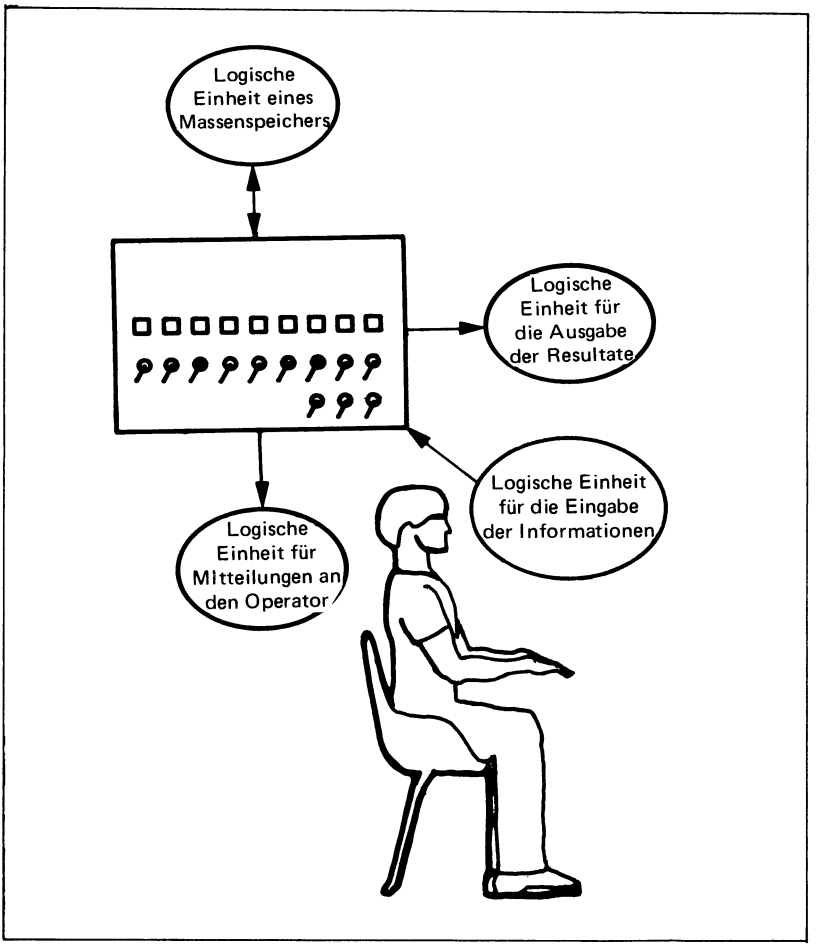


Bild 3.1 Logische Einheiten, die einen Mikrocomputer umgeben

Nur der Mikrocomputer selbst kann durch nichts ersetzt werden. Daher müssen in diesem Fall die physikalischen und logischen Einheiten immer ein und dasselbe sein. Sehen wir uns einige Bilder an, mit denen die Beziehungen zwischen physikalischen und logischen Einheiten noch deutlicher dargestellt werden. Zunächst zeigt Bild 3.1 die vier logischen Einheiten, die den Mikrocomputer umgeben. In Bild 3.2 sehen wir, wie diese vier logischen Einheiten in dem anfangs in Kapitel 1 Bild 1.1 eingeführten Mikrocomputer-System enthalten sind. Schließlich zeigt Bild 3.3 wie die logischen und physikalischen Einheiten in Joe Bitburgers einfachem Mikrocomputer-System, das aus dem Mikrocomputer und einem Fernschreiber besteht, zusammengefügt sind. **Wenn auch das Konzept der logischen und physikalischen Einheiten schwer verständlich erscheinen mag, so finden wir jedoch viele Parallelen in unserem Alltagsleben.**

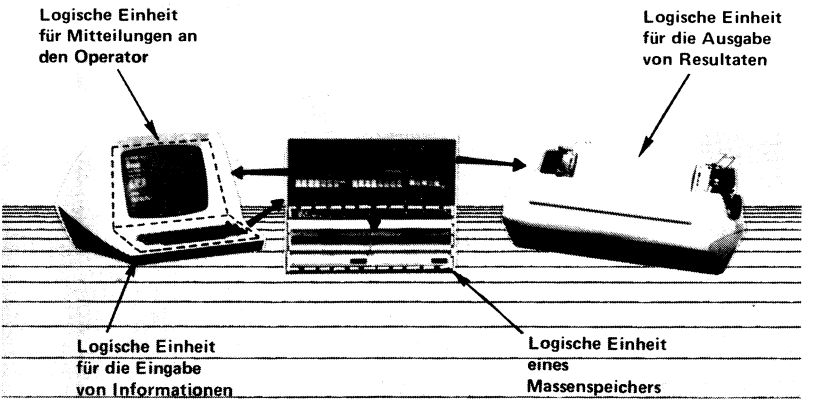


Bild 3.2 Die logischen Einheiten für das Mikrocomputer-System von Bild 3.1

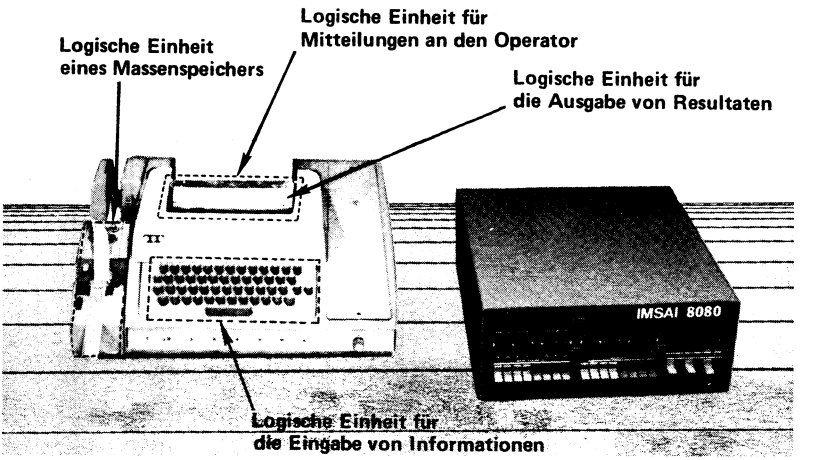


Bild 3.3 Logische Einheiten für ein Fernschreiber-Terminal

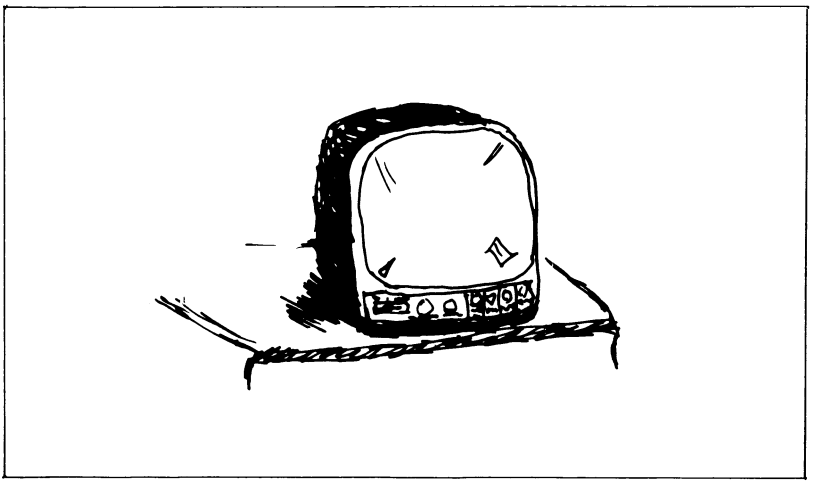
Joe Bitburger braucht morgens eine logische Einheit des „Aufweckens" oder er wird nicht rechtzeitig aus dem Bett kommen. Daher stellt eine physikalische Einheit „Wecker" die logische Einheit des „Aufweckens" dar. Eines Tages funktioniert der Wecker nicht. Glücklicherweise ist das Wetter in dieser Jahreszeit so, daß der Wecker gerade klingelt wenn die Sonne aufgeht. Joe läßt daher seine Vorhänge offen, während der Wecker fix eingestellt bleibt. Er verwendet die aufgehende Sonne, die durch sein offenes Fenster scheint als die physikalische Einheit, die die logische Einheit des Weckens darstellt. Hätte sich Joe nun darauf eingestellt, daß er nur auf das Geräusch des Weckers am Morgen reagiert, würde das Wecken durch die Sonnenstrahlen nicht funktionieren. Joe hat sich aber klugerweise selbst so programmiert, daß er jeden Weckreiz annimmt. Daher stellt es für ihn kein Problem dar, wenn er vom Läuten des Weckers zur hereinfallenden Sonne durch das offene Fenster umschaltet, das ihn zum Aufwachen bringt.

Joe trinkt gewöhnlich eine Tasse Kaffee zum Frühstück. Um sich diese Tasse Kaffee zu machen, benötigt er eine logische Einheit „Wassererhitzer". Die logische Einheit „Wassererhitzer" kann eine physikalische Einheit „Kessel" sein. Wenn der Kessel jedoch ein Loch hat, ersetzt Joe diese durch eine physikalische Einheit „Kochtopf".
Nach dem Frühstück benötigt Joe die logische Einheit „Transport zur Arbeit". Normalerweise ist ein „Bus" die physikalische Einheit, die Joe's logische Einheit „Transport zur Arbeit" darstellt. Eines Tages fällt der Bus wegen eines Streikes aus, so daß Joe mit dem Fahrrad zur Arbeit fährt. Das „Fahrrad" wird nun die physikalische Einheit, die die logische Einheit „Transport zur Arbeit" darstellt. Hätte sich Joe selbst dazu programmiert, nur mit einem exakten Busfahrplan zu leben, würde er nun ein Problem haben. Joe hat sich jedoch selbst so programmiert, daß er verschiedene Hilfsmittel verwenden kann, um zur Arbeit zu gelangen. Er hat sich die Zeit so eingeteilt, daß er bei einem ausfallenden oder verspäteten Bus trotzdem genügend Zeit für sein Fahrrad besitzt.
Dies sind nur drei Beispiele von Analogien logischer und physikalischer Einheiten, denen wir im täglichen Leben begegnen.

MIKROCOMPUTER-HARDWAREKOMPONENTEN

Sehen wir uns nun einige der Kombinationen physikalischer Einheiten an, die wir bei unserem nächsten Computerlieferanten finden.
Wir finden zum Beispiel Kombinationen von Tastatur und Videoanzeige:

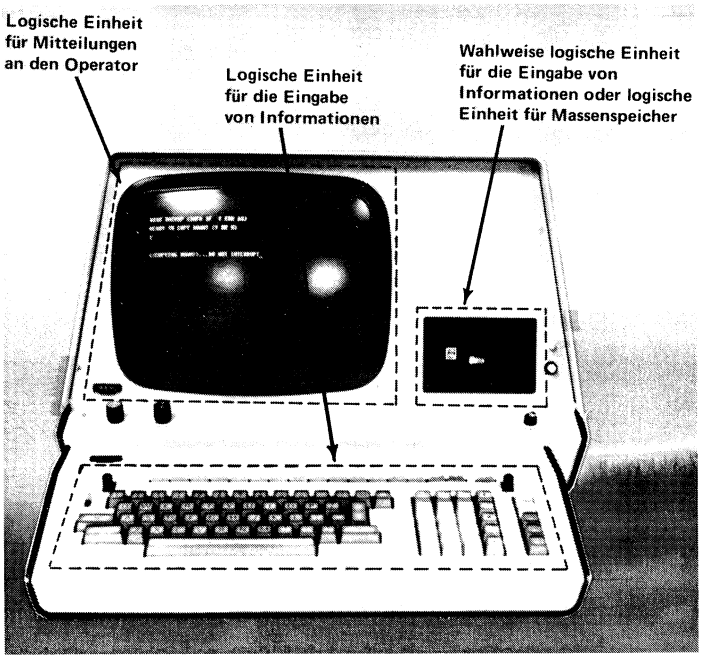
Diese Kombination ist so gebräuchlich, daß viele Leute denken, man müßte die beiden zusammenkaufen. Aber sie müssen nicht. Man kann eine Videoanzeige allein erwerben:



Oder wir können unser Fernsehgerät als Videoanzeige verwenden.
Wir können auch eine Tastatur allein kaufen:

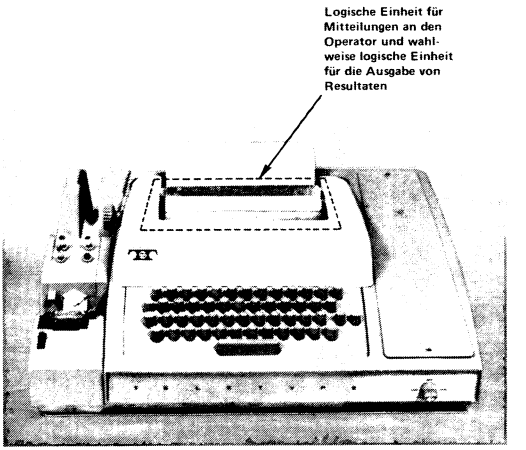
Logische Einheit für die Eingabe von Informationen

Es gibt auch Einheiten mit Tastatur und Videoanzeige, die einen Kassettenantrieb enthalten. Dieser Kassettenantrieb kann ein Teil oder die gesamte logische Einheit für unseren Speicher für Masseninformationen darstellen:

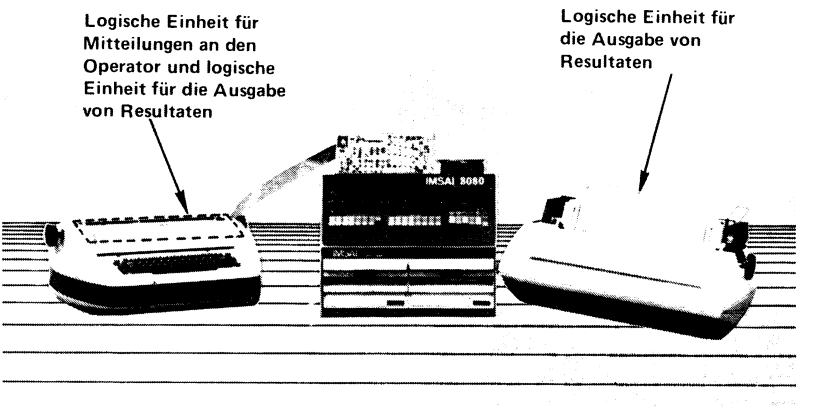


Wir können auch ein Gerät kaufen, das wie eine Tastatur aussieht, jedoch wesentlicher Teil eines Mikrocomputers ist:

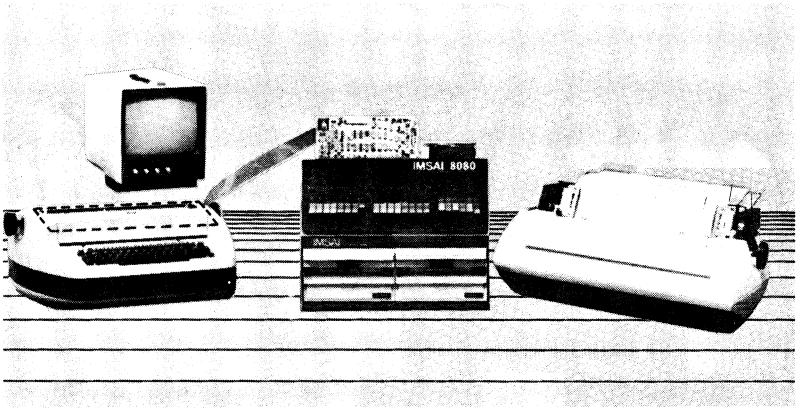
Drucker können auch manchmal verwirren. Manchmal wird die logische Einheit der „Mitteilung an den Bedienenden" und logische Einheit „Ergebnisausgabe" zur gleichen physikalischen Einheit:



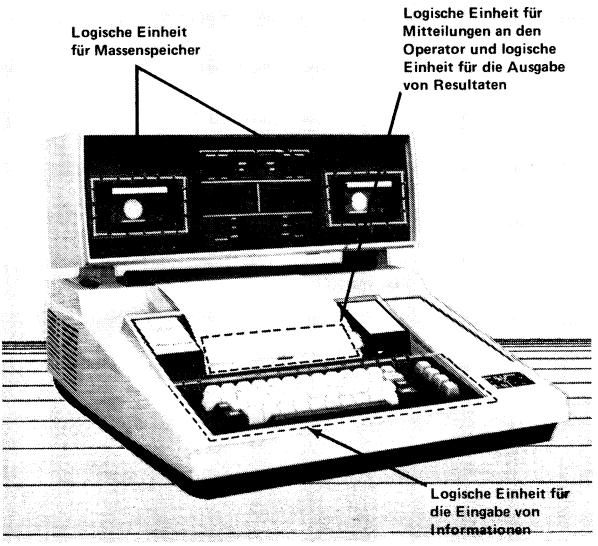
Aber wir können das gleiche Mikrocomputer-System nehmen, einen getrennten Drucker hinzufügen und nunmehr sind die logische Einheit der „Ergebnisausgabe" und die logische Einheit der „Mitteilung an den Bedienenden", obwohl sie beide Drucker sind, getrennte physikalische Einheiten:



Fügen wir eine Videoanzeige hinzu, so können wir entweder die Schreibmaschine oder den Drucker als logische Einheit für die Ausgabe der Resultate verwenden, während der Fernschreiber oder die Videoanzeige als logische Einheit der „Mitteilung an den Bedienenden" dienen:

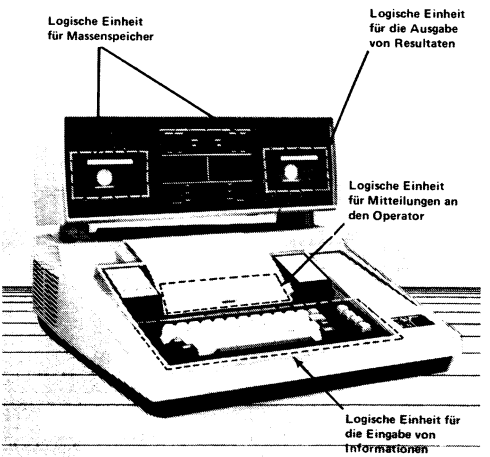


Es gibt eine sehr populäre Terminal-Serie, die Silent 700 von Texas Instruments. Eine Version dieses Terminals kombiniert Tastatur, Drucker und Kassetten. Dieses Terminal kann für die logischen Einheiten der Informationseingabe, Mitteilung an den Bedienenden, Resultatausgabe und Massenspeicher von Informationen dienen:

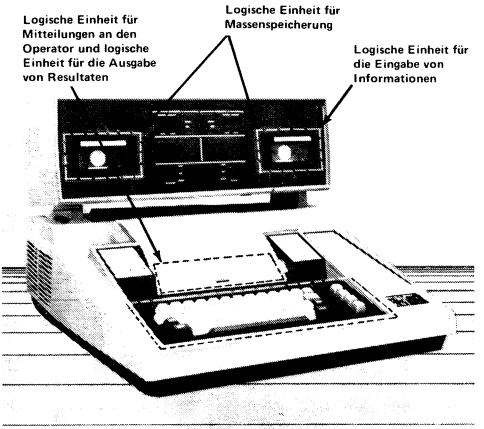


AUFBEWAHRUNG VON RESULTATEN IN KASSETTEN

Wenn wir das Terminal Silent 700 von Texas Instruments verwenden, können wir eine andere interessante Möglichkeit betrachten. Es liegen Ergebnisse vor, die wir ausdrucken wollen, wir benötigen jedoch momentan diese Ergebnisse nicht, oder wir können es uns leisten auf den Drucker zu warten. Wir bestimmen einen der Kassettenantriebe als physikalische Einheit, entsprechend der logischen Einheit für die Ausgabe von Resultaten:



Nun werden unsere Resultate sehr rasch in die Kassette eingeschrieben. **Wir entfernen die Kassette und bewahren sie getrennt auf. Wenn wir sie wieder einlegen, so bestimmen wir den Kassettenantrieb neu als physikalische Einheit, entsprechend der logischen Einheit für die Massenspeicherung von Informationen und den Drucker als logische Einheit für die Resultatausgabe:**



GERÄTETREIBER

Das Schöne an einem Mikrocomputer-System ist seine Vielseitigkeit. Wenn wir die nötige Hardware beisammen haben, so brauchen wir nur ein kleines Programm um jede physikalische Einheit zur Verkörperung einer beliebigen logischen Einheit, die physikalisch vernünftig ist, zu verwenden. Beispielsweise würde die Zuweisung der physikalischen Einheit Drucker als logische Einheit für die Informationseingabe unrealistisch sein, da es physikalisch unmöglich ist. Ein Drucker kann nur Daten empfangen, während eine logische Einheit zur Dateneingabe Daten senden können muß.

Wir können jede vernünftige physikalische Einheit jeder beliebigen logischen Einheit zuweisen, wenn wir das entsprechende Programm zur Verbindung dieser beiden in unserem Mikrocomputer-System besitzen. Diese Programme werden als „Gerätetreiber" bezeichnet. Gerätetreiberprogramme sind funktionsmäßig in Bild 3.4 dargestellt.
Es lohnt sich Bild 3.4 genauer anzusehen, da es eine Anzahl von Ideen und Konzepten beinhaltet, die für einen Anfänger etwas schwierig zu erfassen sind.
Alle physikalischen Einheiten, die in der unteren Reihe von Bild 3.4 dargestellt sind, sind alle einzelne Hardwareteile, die man sehen und berühren kann. Jede dieser physikalischen Einheiten muß ihr eigenes Gerätetreiberprogramm besitzen, das die spezifischen Anforderungen der physikalischen Einheiten befriedigt. Ein Baustein-Treiberprogramm ist einfach ein Computerprogramm, eine Folge von Zahlen wie jedes andere Programm. Was ein Baustein-Treiberprogramm von einem anderen Programm unterscheidet, ist die Logik des Programms – die Tatsache, daß diese Logik die physikalischen Möglichkeiten einer physikalischen Einheit nutzbar macht, damit die Funktionen, die von der logischen Einheit gefordert werden, erfüllt werden.
Gerätetreiberprogramme sind für uns nicht mehr neu. Erinnern wir uns, daß Joe Bitburger ein Baustein-Treiberprogramm (auf dem Chip eines Festwertspeichers) für die Steuerung seines Fernschreibers gekauft hat.

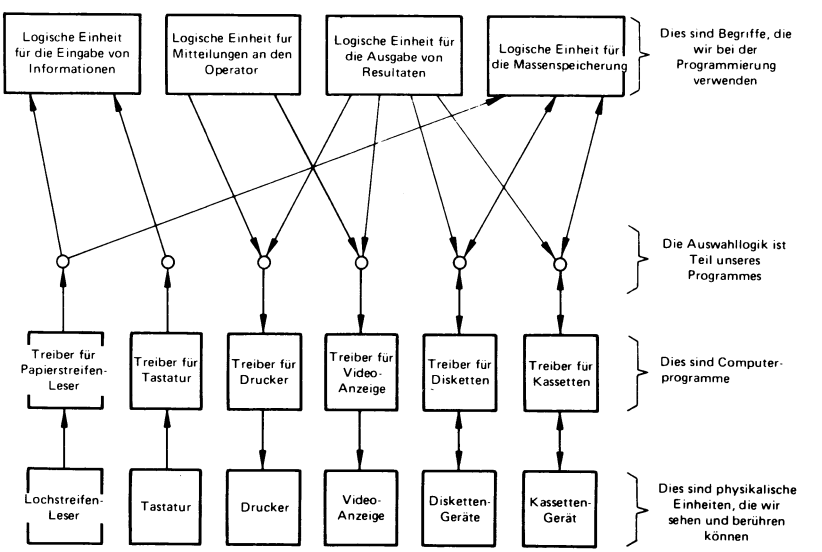
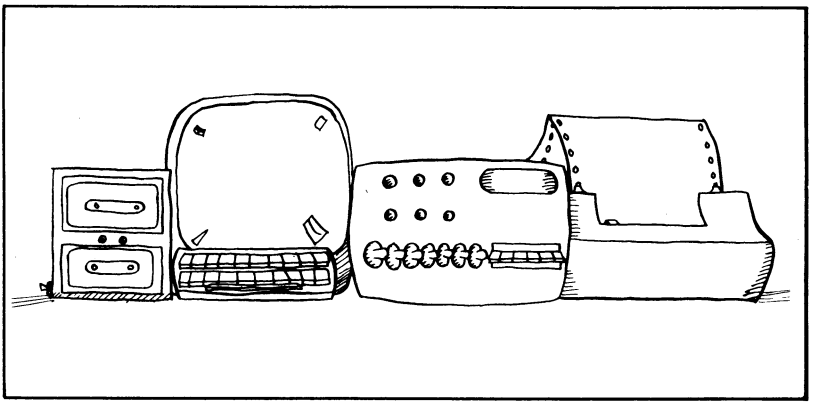


Bild 3.4 Logische und physikalische Einheiten, die mittels eines Gerätetreibers verbunden sind

Eine logische Einheit ist nichts weiter als eine Idee oder ein Gedanke. Wir können niemals eine logische Einheit sehen oder berühren. In Joe Bitburgers Programm für die Bezahlung von Rechnungen wird der Lochstreifenleser beispielsweise zur logischen Einheit für die Massenspeicherung von Informationen, da der Lochstreifenleser Joe Bitburgers Namen und Adressen liefert. Das Konzept der gespeicherten Informationen, in Joe Bitburgers Fall Namen und Adressen, ist eine Idee. Diese Idee ist mit einer physikalischen Realität verbunden, nämlich dem Papierstreifenleser mittels des Fernschreiber-Gerätetreiberprogrammes.

Nehmen wir nun an, daß Joe Bitburger mit dem Schreiben seines Zahlungsprogrammes eine Aufgabe gut gelöst hat. Er hätte sein Programm für die Bezahlung von Rechnungen geschrieben, indem er alle den Mikrocomputer umgebenden Geräte als logische Einheiten angesehen hat. Nun, da Joe seinen Fernschreiber verloren hat, nehmen wir an, daß er diesen durch eine Videoanzeige, eine Tastatur, einen alleinstehenden Drucker und ein Paar von Kassettenantrieben ersetzt hat:



Muß Joe Bitburger nun wieder von vorne anfangen und sein Programm für die Bezahlung von Rechnungen völlig neu schreiben? In der Tat nicht. **Alles was er tun muß, ist sein Gerätetreiberprogramm für den Fernschreiber durch entsprechende Gerätetreiberprogramme für die Videoanzeige, Tastatur, Drucker und Kassette zu ersetzen.** Dann muß er einfach die logischen Einheiten mit den physikalischen Einheiten in einer neuen korrekten Weise verbinden, und sein gesamtes Mikrocomputer-System wird einwandfrei arbeiten. Dies ist in Bild 3.5 dargestellt:

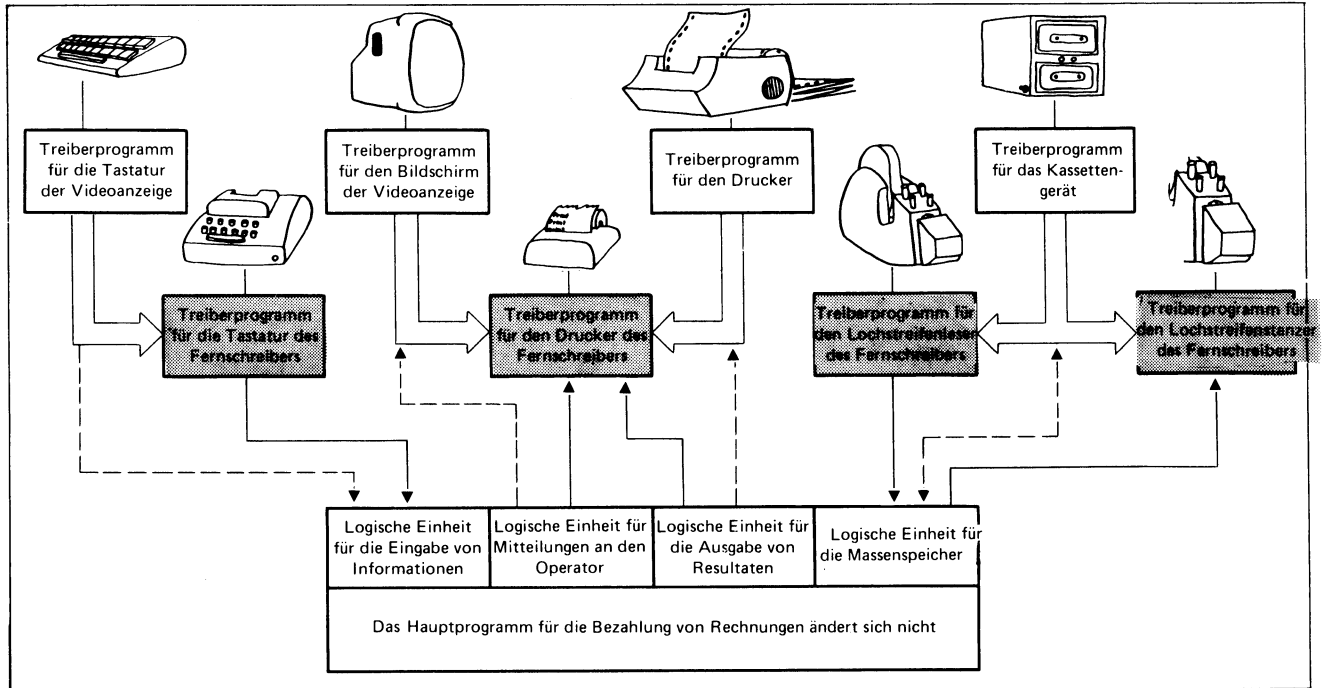


Bild 3.5 Die Verwendung von Gerätetreiberprogrammen zum Ersatz physikalischer Einheiten

In Bild 3.5 wurde das Gerätetreiberprogramm für den Fernschreiber von Joe Bitburger in vier Teile aufgeteilt. Diese vier Teile sind ein Gerätetreiberprogramm für die Fernschreibertastatur, ein Gerätetreiberprogramm für den Fernschreiberdrucker, ein Gerätetreiberprogramm für den Lochstreifenleser des Fernschreibers und ein Gerätetreiberprogramm für den Lochstreifenstanzer des Fernschreibers. In Wirklichkeit würden alle vier Teile des Treiberprogrammes für den Fernschreiber in einem einzigen Festwertspeicher-Chip enthalten sein, und das gesamte Paket würde als ein einziges Gerätetreiberprogramm behandelt. Joe Bitburger muß seinen Festwertspeicher-Chip entfernen und ihn durch vier neue Gerätetreiberprogramme ersetzen. Die vier neuen Gerätetreiberprogramme ersetzen nicht die vier alten in einem Verhältnis 1 zu 1. Wie gezeigt, ersetzen zwei neue Gerätetreiberprogramme ein einziges Gerätetreiberprogramm für den Fernschreiberdrucker, während ein Gerätetreiberprogramm für die Kassetteneinheit sowohl das Programm für den Lochstreifenleser des Fernschreibers als auch den Lochstreifenstanzer des Fernschreibers ersetzt.

Während die vier Gerätetreiberprogramme für den Fernschreiber wie ein Programm auf einem einzigen Festwertspeicher-Chip aussehen, werden die vier neuen Programme wie vier separate und unterschiedliche Programme aussehen:

1) Die Gerätetreiberprogramme für die Tastatur und den Bildschirm für Videoanzeigen.
2) Das Gerätetreiberprogramm für den Drucker.
3) Das Treiberprogramm für die Kassetteneinheit.

Jedes getrennte Programm unterstützt eine getrennte physikalische Einheit. Das getrennte Programm kann als Festwertspeicher-Chip erhältlich sein, aber alle drei Programme werden normalerweise nicht als einzelne Festwertspeicher-Chips zu bekommen sein. Der Grund hierfür ist, daß man getrennte und bestimmte physikalische Einheiten in zu vielen verschiedenen Kombinationen kaufen kann. Nehmen wir beispielsweise an, wir hätten drei verschiedene Videoanzeigen (A, B und C), drei verschiedene Drucker (P, Q und R) und drei verschiedene Kassetteneinheiten (X, Y und Z). Wir würden 27 verschiedene Festwertspeicher-Chips benötigen, um alle Kombinationen der Gerätetreiberprogramme zu erhalten, die diese neun physikalischen Einheiten in jeder möglichen Kombination erfordern würden. Dies kann folgendermaßen gezeigt werden:

ROM 1 A+P+X
ROM 2 A+P+Y
ROM 3 A+P+Z
ROM 4 A+Q+X
ROM 5 A+Q+Y
ROM 6 A+Q+Z
-
-
-
ROM 25 C+R+X
ROM 26 C+R+Y
ROM 27 C+R+Z

Wir müssen zwar wissen was Gerätetreiber sind, wir werden aber wahrscheinlich niemals selbst Gerätetreiber schaffen müssen. So wie Joe seinen Festwertspeicher-Chip für seine Fernschreiber-Ladeprogramme (oder Gerätetreiber) gekauft hat, so werden wir einfach Gerätetreiberprogramme als fertige Bauteile kaufen.

WAHLMÖGLICHKEITEN FÜR MIKROCOMPUTER-SYSTEMKOMPONENTEN

In Kapitel 1 und 2 haben wir die erforderlichen Funktionen beschrieben, die von jedem Teil eines Mikrocomputer-Systems auszuführen sind. Wir haben noch nicht die Wahlmöglichkeit bei den Komponenten besprochen.
Man kann nicht immer klar zwischen einer Notwendigkeit und einer zusätzlichen Möglichkeit unterscheiden. Wir wollen jedoch annehmen, daß die in den Kapiteln 1 und 2 beschriebenen Komponenten eines Mikrocomputer-Systems Notwendigkeiten darstellen, während wir nun die verschiedenen Erweiterungsmöglichkeiten beschreiben wollen.

SICHTGERÄTE

Sehen wir uns zunächst einmal die Möglichkeiten an, die wir in Videoanzeigegeräten finden.
Bild 3.6 illustriert die Logik eines Gerätetreiberprogrammes zur Steuerung der meisten elementaren Tastaturen und Video-Terminals. Das Programm in Bild 3.6 macht eigentlich sehr wenig. Das Programm wartet auf das Antippen einer Taste auf der Tastatur. Bei Feststellung einer Tastenbetätigung verzweigt die Programmlogik in eine von drei Richtungen:

1) Sie zeigt ein darstellbares Zeichen an
2) Sie antwortet auf einen Videoanzeigen-Steuercode.
3) Sie sendet jeden Steuercode für das Mikrocomputer-System zum Mikrocomputer.

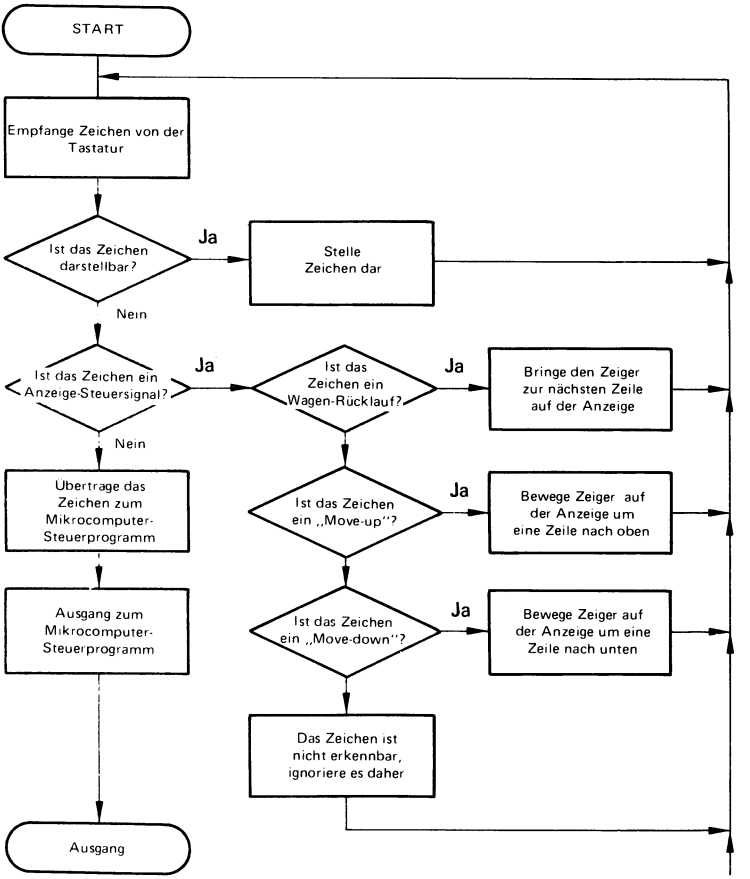


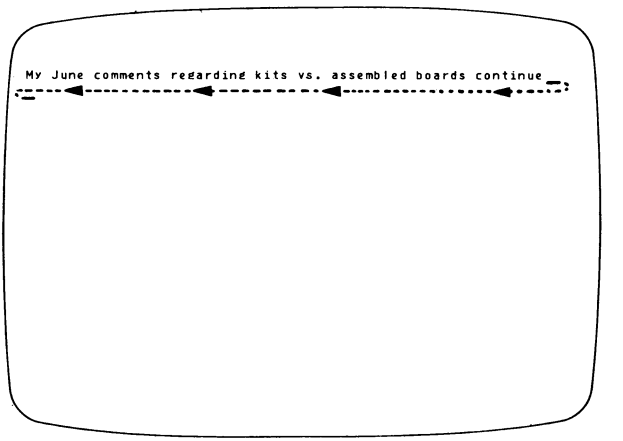
Bild 3.6 Flußdiagramm für ein einfaches Videoanzeigen-Treiberprogramm

Wenn mit einem Tastenanschlag ein Buchstabe des Alphabets, eine Zahl oder ein anderes darstellbares Zeichen eingegeben wird, so schreibt das Programm das gleiche zurück auf die Videoanzeige. Erinnern wir uns daran, daß dies „Echobetrieb" genannt wird.

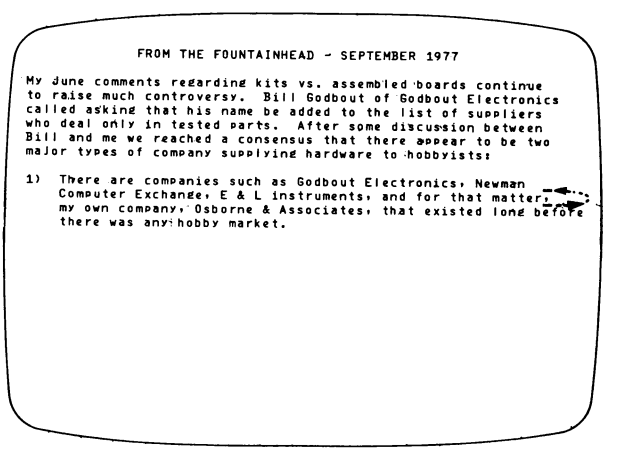
BILDSCHIRMZEIGER

Der Tastenanschlag kann ein Steuersignal für die Videoanzeige darstellen. Die in Bild 3.6 gezeigten Steuersignale enthalten einen „Wagenrücklauf" (carriage return), mit dem man eine Bewegung eine Zeile hinauf oder eine Zeile hinunter erhält. Da es hier

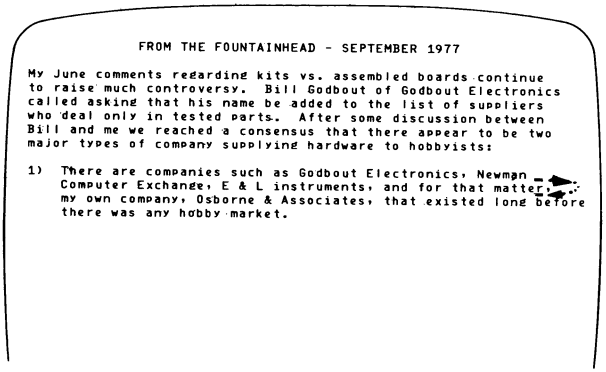
keinen Druckmechanismus gibt, der uns sagt wo wir uns gerade auf dem Bildschirm befinden, verwenden alle Videoanzeigen einen sogenannten „Zeiger" (Cursor). Ein Zeiger ist ein kleiner Punkt, ein Strich oder ein kleines leuchtendes Quadrat, das sich an der Stelle befindet, an der das nächste Zeichen dargestellt wird.
Wenn wir die Taste für den Wagenrücklauf am Bildschirm-Terminal betätigen, so bewegt sich der Zeiger zum Anfang der nächsten tieferliegenden Zeile:



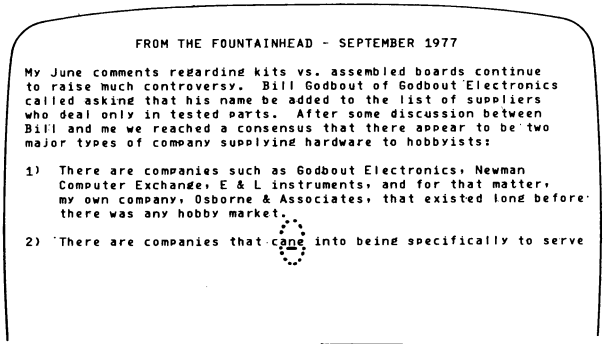
Die beiden nächsten Steuersignale für die Videoanzeige, die in der Logik von Bild 3.6 gezeigt sind, bewegen den Cursor eine Zeile nach oben:

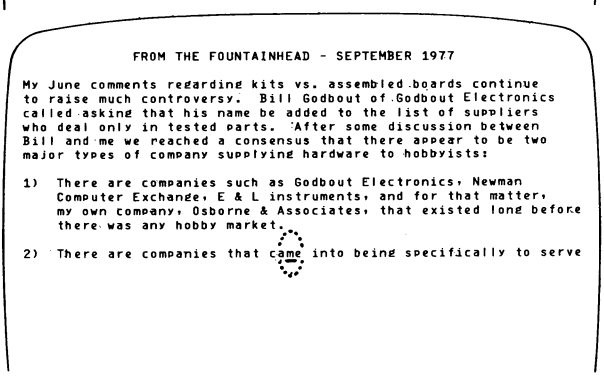


Oder eine Zeile nach unten:



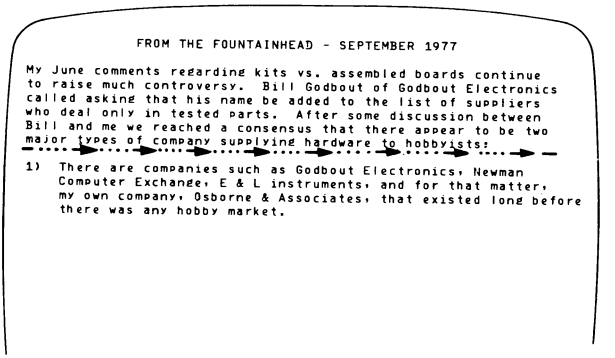
Wie oben dargestellt wurde, verwenden wir den Zeiger zur Änderung eines auf dem Bildschirm dargestellten Textes. Wo immer sich auch der Zeiger befindet, an dieser Stelle wird das nächste Zeichen dargestellt. Wir können einen Fehler an jeder Stelle im Text folgendermaßen korrigieren:





Eine Anzahl von Steuersignalen für die Videoanzeige, die in der Logik von Bild 3.6 nicht gezeigt sind, sind nichtsdestoweniger gebräuchliche Optionen. Diese enthalten:

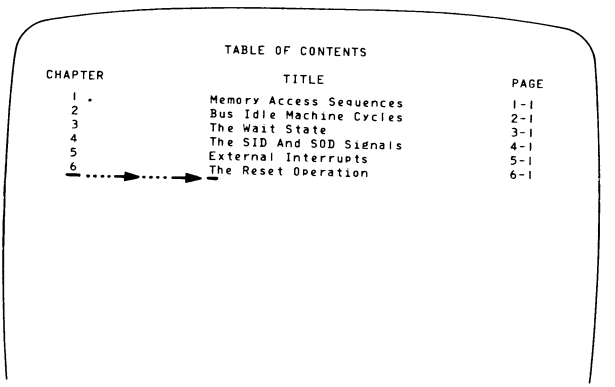
1) Leertaste (Forward spacing). Wenn wir diese Taste gedrückt halten, so bewegt sich der Zeiger in die Richtung auf das Ende der Zeile:



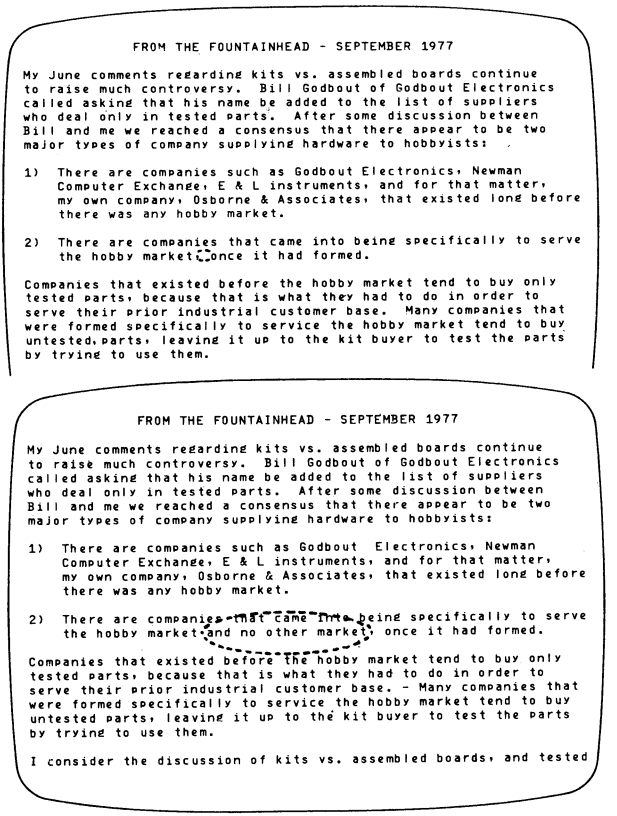
Abhängig davon, wie unsere Videoanzeige entwickelt wurde, kann der Zeiger bei Erreichen des Endes der Zeile stoppen, oder er kann sich vom Ende der Zeile zum Anfang der gleichen Zeile bewegen oder er kann sich vom Ende der Zeile zum Anfang der nächsten tieferliegenden Zeile bewegen.

2) **Rückwärtsleertaste (Backward spacing).** Wenn wir diese Taste betätigen, so bewegt sich der Zeiger einfach in der entgegengesetzten Richtung, wie es eben für die Vorwärtsleertaste gezeigt wurde.

3) **Tabulator (Tabbing).** Bei manchen Videoanzeigen lassen sich sehr einfach Tabellen setzen. Wenn man den Tabulator an der entsprechenden Stelle betätigt hat, so springt der Zeiger bis zur nächsten Tabulatorposition auf der Videoanzeige innerhalb der momentanen Zeile:



4) **Einsetzen und Löschen von Textstellen.** Videoanzeigen haben einen sehr wesentlichen Vorteil im Vergleich zu jedem Drucker. Wir können Textteile elektronisch bewegen. Viele Videoanzeigen besitzen derartige Einrichtungen und gestatten das Einsetzen oder Löschen eines bestimmten Textteiles. Das Einsetzen kann folgendermaßen gezeigt werden:



STEUERUNG DES MIKROCOMPUTERS DURCH DIE TASTATUR

Ein Zeichen, das über die Tastatur eingegeben wird, kann eine Steuerfunktion identifizieren, die nichts mit der Videoanzeige zu tun hat. Erinnern wir uns daran, wie Joe Bitburgers Programm immer die Zeichen „Escape" oder Fehlerfeststellungs-Zeichen enthielt? Dies ist ein Zeichen, das einen Teil des Programmes neuerlich startet, das heißt, daß es bei Joe bei einem Fehler eine einfache Korrektur gestattet. Bildschirmgeräte enthalten immer eine oder mehrere derartiger Tasten. Dies sind Spezialtasten, da sie keine darstellbaren Zeichen oder Bildschirmsteuersignale liefern. Manche Bildschirm-

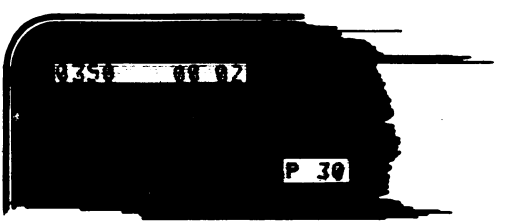
Terminals ordnen einige spezielle Steuersignale speziellen Aufgaben zu. Beispielsweise könnte es eine Taste geben, die das Terminal vom Mikrocomputer trennt. Es könnte auch eine andere Taste vorhanden sein, die den Mikrocomputer stoppt oder startet, sobald er mit dem Terminal verbunden wurde.
Wenn unser Mikrocomputer keine Frontplatte besitzt, werden die Frontplattenschalter durch spezielle Steuerzeichen ersetzt, die über die Tastatur eingegeben werden. Viele Mikrocomputer besitzen daher überhaupt keine Frontplatte und verwenden statt dessen eine Tastatur. Falls unser Mikrocomputer aber doch eine Frontplatte besitzt, werden wir häufig wahlweise eine Tastatur anstelle der Frontplattenschalter verwenden können.

ANZEIGE VON GROSS- UND KLEINBUCHSTABEN

Weitere Möglichkeiten für eine Videoanzeige bestehen in der Darstellungsmöglichkeit von Groß- und Kleinbuchstaben. Die einfachsten und preisgünstigsten Bildschirm-Terminals verwenden nur Großbuchstaben. Teurere Bildschirmterminals projizieren sowohl Groß- als auch Kleinbuchstaben. Die Tastatur besitzt eine entsprechende Umschalttaste, die wie bei einer Schreibmaschine die Umschaltung zwischen Klein- und Großbuchstaben gestattet.

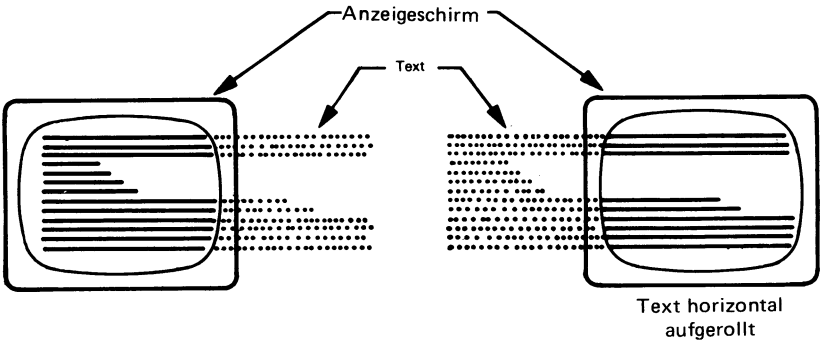
UMGEKEHRTE ANZEIGE

Manche Videoanzeigen gestatten die Darstellung auf dem Schirm umzukehren, so daß schwarze Zeichen auf einem weißen Hintergrund auf allen Teilen des Schirmes dargestellt werden:



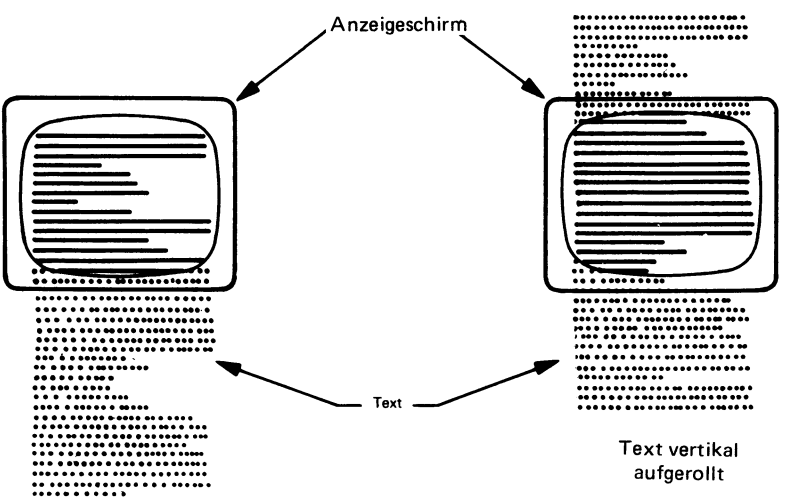
HORIZONTALER AUFROLLMODUS

Manche Anzeigen gestatten das Aufrollen des Textes horizontal oder vertikal (scrolling, scrole = Schriftrolle). Beim horizontalen Aufrollen blickt man auf Textzeilen, die länger sind als der Schirm breit ist. Man könnte den Schirm als ein Fenster auf den Zeilen ansehen. Dies kann folgendermaßen dargestellt werden:

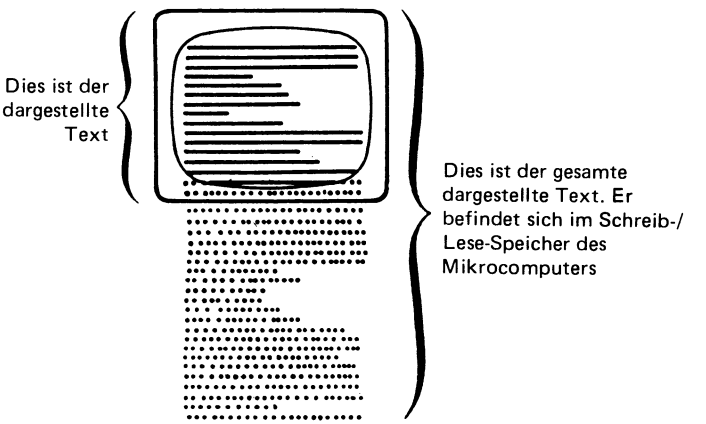


VERTIKALER AUFROLLMODUS

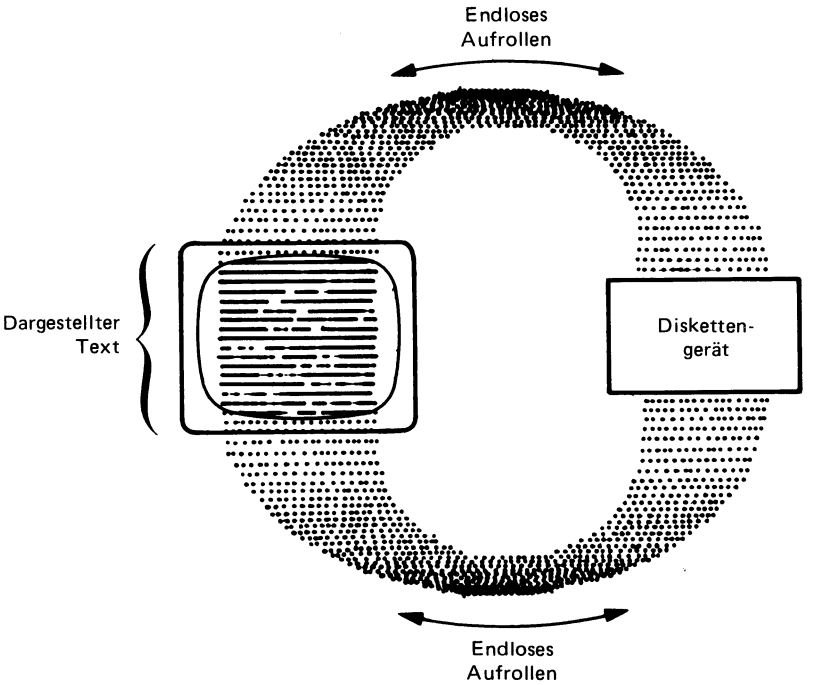
Horizontales Aufrollen ist nicht sehr gebräuchlich bei Videoanzeigen. **Der vertikale Aufrollmodus ist dagegen sehr häufig zu finden.** In diesem Fall können wir uns vorstellen, daß der Text mehr Zeilen hat als der Bildschirm anzeigen kann. Der Schirm gestattet ein Verschieben des Textes nach oben und nach unten. Dies kann folgendermaßen gezeigt werden:



Es gibt zwei verschiedene Arten, in denen Terminals das vertikale Verschieben gestatten, wobei einer der beiden Wege zu bevorzugen ist. Wir werden beide beschreiben.
Die weniger wünschenswerte Version des vertikalen Aufrollens oder Verschiebens enthält in einem lokalen Schreib-/Lesespeicher den gesamten Text, den wir darstellen können. Dies kann folgendermaßen gezeigt werden:



Bei dieser Variante des Aufrollmodus können wir nicht über den oberen Rand hinaus verschieben oder unter den unteren Rand des Textes, der momentan im Schreib-/Lesespeicher aufbewahrt ist. Wenn wir Text einfügen und hierbei ein Überlauf des verfügbaren Schreib-/Lesespeichers auftritt, dann verlieren wir einfach Informationen am oberen oder unteren Rand unseres Textes.
Wenn unser Mikrocomputer ein Diskettengerät besitzt, dann wird ein gut entwickelter Aufrollzusatz das Diskettengerät mit dem Schreib-/Lesespeicher verbinden, in dem der angezeigte Text gespeichert wird. Wenn wir nun über oder unter den Text des Schreib-/Lesespeichers hinausschreiben, werden entsprechende Programme innerhalb des Mikrocomputer-Systems automatisch einen Teil des Textes im Schreib-/Lesespeicher in die Diskette zurückspeichern und neuen Text von der Diskette in den Schreib-/Lesespeicher bringen. Dies sieht aus, als ob der Text unbegrenzt verschoben werden kann. Das kann folgendermaßen gezeigt werden:



Wenn wir Text in einem derartigen fortschrittlichen System einsetzen und zu viel in den verfügbaren Mikrocomputer-Schreib-/Lesespeicher einschreiben, dann wird der überschüssige Text einfach in die Diskette geschrieben und geht niemals verloren.
Es ist zu beachten, daß diese Möglichkeit des Aufrollmodus nichts mit dem Bildschirmterminal oder der Tastatur zu tun hat. Diese Möglichkeiten beziehen sich auf das gesamte Mikrocomputer-System und auf die Art und Weise, wie es für uns programmiert wurde.

GRAFISCHE ANZEIGEN

Grafische Anzeigen sind eine weitere nützliche Möglichkeit. Eine grafische Videoanzeige gestattet uns die Darstellung von Bildern zusätzlich zu Zeichen. Billige grafische Sichtgeräte arbeiten in Schwarz und Weiß, jedoch nicht mit Grau. Sie erzeugen Bilder aus geraden Linien, Kreisen, Punkten und festen Formen. Teurere grafische Bildschirm-Terminals gestatten uns Darstellungen wie auf einem Fernsehbildschirm und mit feineren Details.
Es gibt auch Videosichtgeräte mit farbiger Darstellung. Eine farbige Videoanzeige gibt uns zusätzliche Möglichkeiten, indem mittels Farben bestimmte Zeichen oder grafische Segmente hervorgehoben werden können.

LICHTGRIFFEL

Einige teurere Videoanzeigen gestatten uns das Zeichnen auf dem Bildschirm mit einem „Lichtgriffel" oder „Lichtstift". Die elektronische Logik hinter der Videoanzeige „erinnert" sich an die Punkte auf dem Schirm, die mit dem Lichtgriffel berührt wurden.

TASTATUREN

Sehen wir uns nun verschiedene Möglichkeiten bei den Tastaturen an.
Wie schnell kann man die Tasten auf der Tastatur anschlagen? Ist der Mikrocomputer schnell genug, um ein Zeichen zu verarbeiten, bevor wir die nächste Taste anschlagen? Im Falle des einfachen Programmes für eine Tastatur, wie in Bild 3.6 dargestellt wurde, ist die Antwort mit Sicherheit Ja. Wie wir **eingangs erfahren haben, kann die schnellste Schreibkraft (im Durchschnitt)** neun Tasten pro Sekunde anschlagen, wodurch unser Mikrocomputer die Zeit für die Ausführung von mehr als 20000 Befehlen zwischen den Anschlägen erhält. Das Programm in Bild 3.6 verwendet 100 oder 200 der 20000 Befehle, und das ist mehr als ausreichend. Jedoch neun Anschläge pro Sekunde stellen die mittlere Anschlaggeschwindigkeit für den schnellsten Maschinenschreiber dar, aber nicht die maximale Anschlaggeschwindigkeit. **Obwohl der schnellste Bedienende dieser Tastatur nicht mehr als neun Anschläge pro Sekunde eingeben kann, ist es trotzdem möglich zwei Tasten nahezu gleichzeitig zu drücken,** vorausgesetzt wir schreiben mit mehr als einem Finger.

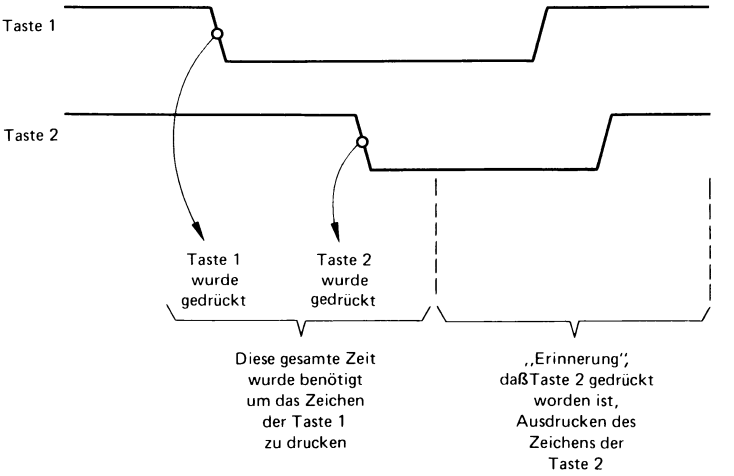
ELEKTRONISCHE VERRIEGELUNG

Zwei Möglichkeiten helfen uns, dadurch entstehende Probleme zu vermeiden: „Elektronische Verriegelung" (rollover) und „Puffer". Zuerst wollen wir die elektronische Verriegelung beschreiben. Was geschieht, wenn wir so schnell tippen, daß wir eine Taste betätigen:

Dann eine andere Taste drücken:

Dann die erste Taste auslassen:

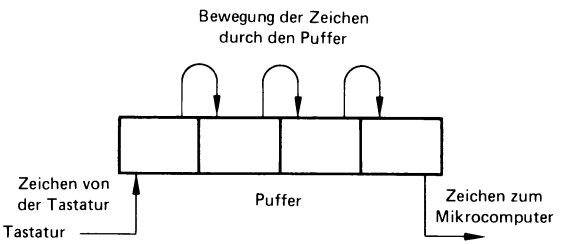
Der Moment, in dem ein Tastenanschlag aufgezeichnet wird ist sehr wichtig. Bei jeder gewöhnlichen Schreibmaschine wird beim Drücken einer Taste das dieser entsprechende Zeichen ausgedruckt. Wenn wir eine zweite Taste überlappend drücken, wie oben gezeigt ist, wird eine elektrische Schreibmaschine den zweiten Tastenanschlag sperren und es wird nichts geschehen. Bei einer mechanischen Schreibmaschine schlägt der zweite Typenhebel auf den Rücken des ersten und das zweite Zeichen wird nicht gedruckt werden.
In der Welt der Elektronik brauchen wir jedoch keine derartigen Einschränkungen. Während eine Taste noch gedrückt ist, kann die elektronische Logik das Drücken einer zweiten Taste feststellen und kann sich an den zweiten Tastenanschlag „erinnern", bis das Zeichen der ersten Taste gedruckt wurde. Dies kann folgendermaßen gezeigt werden:



Die elektronische Verriegelung ist eine äußerst wünschenswerte Möglichkeit in jeder Tastatur, da sie nichts mit der Arbeitsgeschwindigkeit eines Computers zu tun hat. Wenn wir sehr schnelle Finger besitzen, werden sich beim Schreiben sehr häufig Tastenanschläge überlappen. Wenn unsere Tastatur keine elektronische Verriegelung besitzt, so würden wir den zweiten überlappenden Tastenanschlag verlieren.

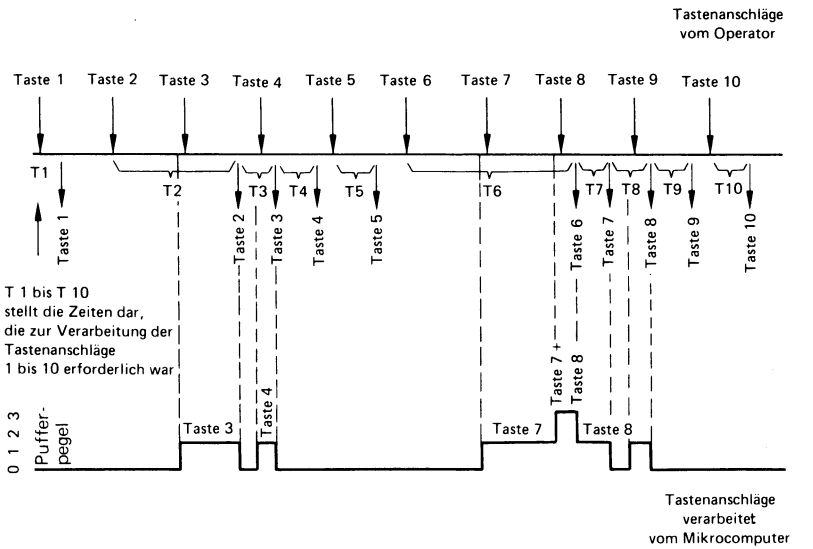
TASTATURPUFFER

Eine weitere Technik zur Verringerung von Eingabefehlern bei Tastaturen ist das Vorhandensein einiger Speicherplätze, in denen die Zeichencodes gespeichert werden, während sie auf die Übertragung zum Mikrocomputer warten. Dies kann folgendermaßen gezeigt werden:



Der oben gezeigte Speicher wird als Puffer bezeichnet. Obwohl der gezeigte Puffer vier Plätze für die Speicherung von Zeichen besitzt, kann ein realer Puffer jede beliebige Anzahl von Zeichen speichern, die normalerweise von zwei bis acht reichen. Ein Puffer gibt dem Mikrocomputer mehr Zeit zur Handhabung von Zeichen, die eine sehr lange Reaktionszeit benötigen. Das in Bild 3.6 gezeigte einfache Treiberprogramm für die Tastatur kann hier nicht mehr angewendet werden. Wenn ein Tastatur-

Treiberprogramm jedoch komplexer wird, ist es für einige Anschläge möglich, daß sie eine längere Verarbeitungszeit benötigen, als für diese zur Verfügung steht. Hier springt nun der Puffer ein. Wenn der durchschnittliche Tastenanschlag mehr als 20 000 Befehle für die Verarbeitung benötigt, wird der Puffer aufgefüllt und überlaufen, wie groß er auch immer ist. Aber wenn nur einige Tastenanschläge mehr als 20 000 Befehle für die Verarbeitung benötigen, dann wird der Puffer befriedigend arbeiten. Dies kann folgendermaßen gezeigt werden:

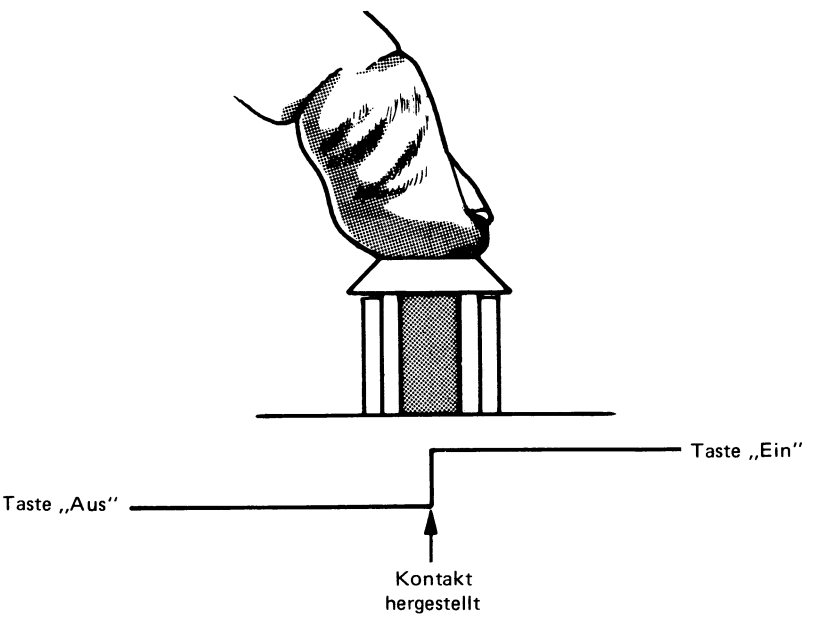


Wir wollen uns nun die obige Abbildung genauer ansehen. Die Tastenanschläge sollen in gleichmäßigen Zeitintervallen erfolgen.

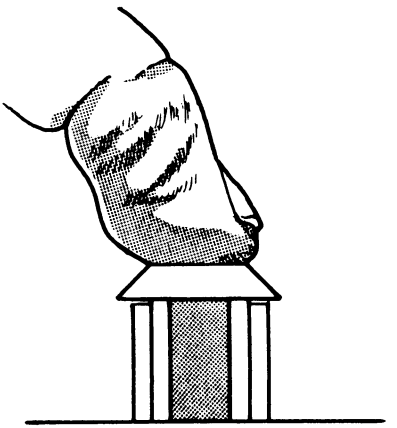
Taste 1 wird während des Zeitintervalles T1 sehr rasch verarbeitet. Der Puffer wird nicht gebraucht. Taste 2 benötigt andererseits eine erheblich längere Verarbeitungszeit. Tatsächlich wird Taste 3 eingegeben, bevor Taste 2 vollständig verarbeitet wurde. Der Puffer muß nunmehr den Code der Taste 3 aufbewahren. Sobald die Verarbeitung der Taste 2 abgeschlossen ist, wird der Code der Taste 3 aus dem Puffer genommen und verarbeitet. Der Puffer ist nun leer. Aber die Verarbeitung der Taste 3 wurde durch die Taste 2 verzögert, so daß Taste 4 eingegeben wird, bevor die Verarbeitung der Taste 3 abgeschlossen ist. Der Code, der die Taste 4 darstellt wird daher im Puffer aufbewahrt, bis die Verarbeitung der Taste 3 beendet ist.

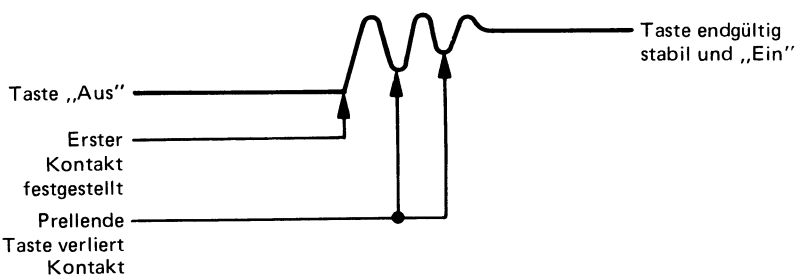
Taste 6 ist der nächste Anschlag und erfordert eine umfangreiche Verarbeitung. Tatsächlich wurde Taste 7 und Taste 8 eingegeben, bevor die Verarbeitung der Taste 6 beendet wurde. Zu diesem Zeitpunkt werden zwei Zeichencodes im Puffer aufbewahrt — Taste 7 und Taste 8. Wenn die Verarbeitung der Taste 6 vollendet ist, wird

der Code der Taste 7 aus dem Puffer genommen und läßt nun den Code der Taste 8 zurück. Taste 7 wird fertig verarbeitet, bevor Taste 9 eingegeben wird, daher ist der Puffer leer. Jedoch die Verarbeitung der Taste 8 wurde verzögert und bewirkte dadurch, daß der Code der Taste 9 für eine kurze Zeit gespeichert wurde.
Ohne das Vorhandensein eines Puffers würden die Eingaben der Taste 3, Taste 7 und Taste 8 verlorengehen.
Unsere Diskussion über mehrfache Tastenanschläge wirft ein neues, jedoch nicht offensichtliches Problem bei Tastaturen auf. Wann wird ein Tastenanschlag tatsächlich exakt festgestellt, nachdem die Taste gedrückt wurde? Die Beantwortung dieser Frage ist nicht so einfach wie sie klingt. Man könnte einfach annehmen, daß beim Drücken auf die Taste irgendeine Art von Kontakt geschlossen wird, und zu diesem Zeitpunkt die Taste als gedrückt angesehen werden kann:



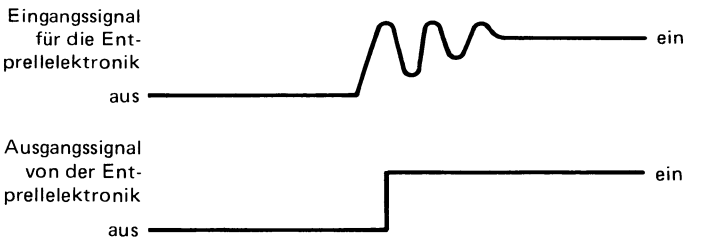
Unglücklicherweise erfolgt das Schließen von elektrischen Kontakten nicht so sauber. Sieht man sich die Vorgänge beim Schließen eines Kontaktes genauer an, so kann dies folgendermaßen aussehen:





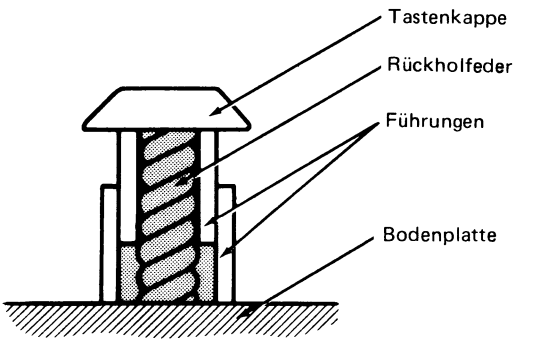
ENTPRELLEN VON TASTEN

Der sich ändernde Druck des Fingers auf einer Taste kann bei einem Kontakt zu einer unregelmäßigen Kontaktgabe führen, bis der Kontakt tatsächlich geschlossen ist. Diese Erscheinung wird als „Kontaktprellen" bezeichnet. Jede einigermaßen gute Tastatur wird daher eine entsprechende Elektronik zur Unterdrückung dieser Erscheinung besitzen. Eine entprellte Tastatur setzt daher das unsaubere Signal beim Betätigen der Tasten in ein exaktes „Aus-/Ein-"Signal um. Dieses kann folgendermaßen gezeigt werden:



Elektronische Verriegelung, Puffer und Entprellen sind elektronische Möglichkeiten, die bei den Tastaturen meist vorhanden sind. Aber mindestens ebenso wichtig sind eine Reihe von mechanischen Aspekten bei Tastaturen.

Die normale Ausführung von mechanischen Tastaturen besitzt Tasten, die etwa folgendermaßen grundsätzlich aussehen:



Wenn man auf alle Details beim Aufbau mechanischer Tastaturen eingehen würde, so könnte man ein ganzes Buch über die Entwicklung mechanischer Schalter schreiben. Für uns genügt innerhalb dieser Besprechung jedoch eine oberflächliche Betrachtung von mechanischen Schaltern. Hierzu genügt die oben gezeigte Abbildung.

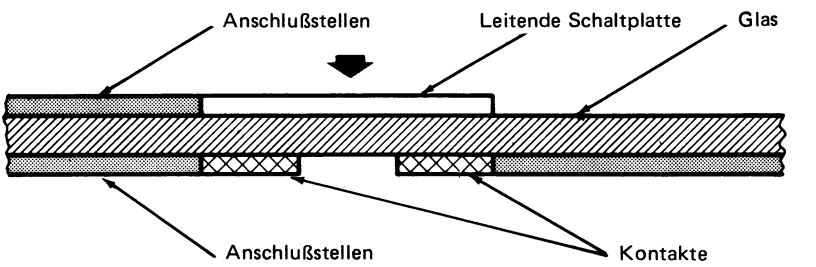
Ein mechanischer Schalter besitzt irgendeine Feder, die die Taste zurück in die „Aus"-Position drückt. Wir drücken die Feder zusammen, wenn wir den Schalter betätigen. An einem bestimmten Punkt, wenn wir den Schalter weit genug nach unten gedrückt haben, wird ein elektrischer Kontakt hergestellt – der Punkt, an dem der Schalter auf „Ein" steht.

Es muß eine relativ stabile mechanische Führung vorgesehen sein, damit beim Drücken des Schalters eine saubere Abwärtsbewegung bis zum Punkt des elektrischen Kontaktes erfolgen kann. Ein mechanischer Schalter mag als sehr einfaches Bauteil erscheinen, ist es in Wirklichkeit jedoch nicht. Beispielsweise wird niemand den Schalter genau senkrecht nach unten drücken. Stets wird durch den Winkel, mit dem unser Finger einen Schalter berührt, bewirkt, daß der Schalter nicht nur nach unten, son-

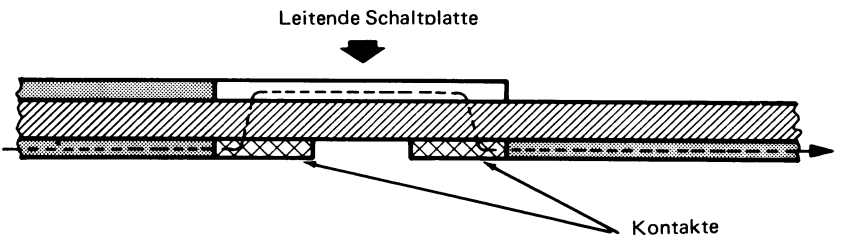
dern auch zur Seite gedrückt wird. Dementsprechend muß der Schalter gestaltet werden. **Auch beim elektrischen Kontakt gibt es eine Reihe von Problemen.** Man könnte einfach einen elektrischen Kontakt dadurch herstellen, daß an der Unterseite der Taste (oder des Tastenfußes) ein Metall mit einem anderen Metall in der Führung oder Bodenplatte zusammendrückt. Jedoch diese Art von trockenen Kontakten, wenn sie nicht besonders gut konstruiert sind, gibt nichts als Ärger. Die geringste Korrosion an der Oberfläche des Metalls reicht aus, um das sichere Funktionieren des Schalters zu verhindern. Bei billigen Schaltern kann man sich durch Besprühen mit irgendeinem Kontakt-Reinigungsmittel behelfen. Dieses Reinigungsmittel greift jedoch meistens das beim Bau des Schalters verwendete Plastik an. Bei den mechanischen Tasten gibt es verschiedene zusätzliche Möglichkeiten. Bei vielen Tasten gibt es einen hörbaren und spürbaren „Klick" beim Drücken der Taste. Dieser zeigt dem Bedienenden deutlich an, ob er die Taste wirklich richtig gedrückt hat.
Wenn die Feder einer Taste schwach wird oder bricht, so kann die Taste ausfallen. Das gleiche kann passieren, wenn die Kontakte verschmutzt sind. Dieser Fehler läßt sich verhältnismäßig leicht beheben. **Manche Tastaturen sind jedoch als geschlossene Einheiten aufgebaut, so daß beim Ausfall einer einzigen Taste die gesamte Tastatur ersetzt werden muß. Diese integrierten Tastaturen sind billiger in der Herstellung, aber offensichtlich wesentlich teurer, wenn sie ausgetauscht werden müssen.**

BERÜHRUNGSSCHALTER

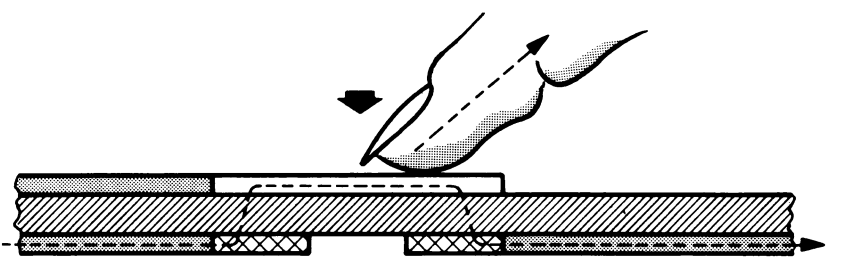
In der Zukunft ist es denkbar, daß mechanische Tastaturen durch Berührungsschalter ersetzt werden.
Berührungsschalter bestehen aus Glas oder anderen nicht leitenden Platten, auf denen elektrisch leitende Muster aufgebracht sind. Dies kann folgendermaßen dargestellt werden:



Normalerweise fließt ein sehr geringer Strom von einem der Kontakte über die oben liegende Schalterplatte und wieder zurück zum zweiten Kontakt. Dies kann folgendermaßen gezeigt werden:



Der hier fließende Strom ist äußerst gering, kann jedoch durch eine entsprechende elektronische Schaltung ohne weiteres festgestellt werden. Wenn wir nun mit unserem Finger die obere Kontaktplatte berühren, so fließt ein kleiner Strom über unseren Körper ab:



Dadurch wird der ursprünglich fließende Strom stark verändert. Eine entsprechende Logik stellt die Änderung dieses Stromes fest und setzt sie in eine entsprechende Schalterstellung um.
Außer dieser Ausführung gibt es noch andere Möglichkeiten für die Berührungs- oder Sensortasten.
Berührungstasten sind nichts neues, werden aber derzeit in elektronischen Geräten relativ selten eingesetzt. Für die Bedienung von Fernsehgeräten und Aufzügen verwendet man seit Jahren Sensortasten.
Da die Berührungstasten sehr robust, preiswert und äußerst zuverlässig sind, werden sie möglicherweise in Zukunft in Computerschaltungen mechanische Tastaturen ersetzen. Etwas ungewohnt ist es, daß die gegenwärtigen Berührungsschalter noch kein hörbares oder spürbares Geräusch beim Betätigen liefern. Auch dieses Problem läßt sich elektronisch leicht lösen.

DRUCKER

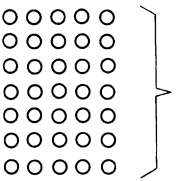
Bei den Druckern bestimmen im wesentlichen drei Faktoren den Preis: Der Druckmechanismus, Zeilenbreite (und damit Papierbreite) und Druckgeschwindigkeit.

DRUCKMECHANISMUS

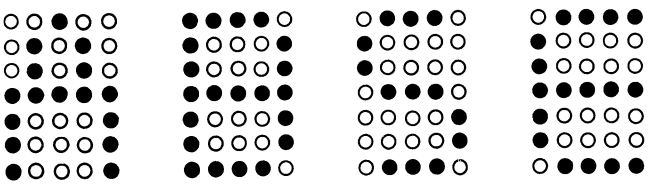
Betrachten wir zuerst den Druckmechanismus. Es gibt zahlreiche Wege, in denen Drucker Zeichen auf einem Papier erzeugen können. Wir betrachten jedoch nur die Druckmechanismen, die man gewöhnlich in preisgünstigen Druckern antrifft.

MATRIXDRUCKER

Die meisten billigen Drucker erzeugen Zeichen als eine Matrix von Punkten. Die einfachsten Matrixdrucker bilden Zeichen aus einer Matrix von 5x7 Punkten:

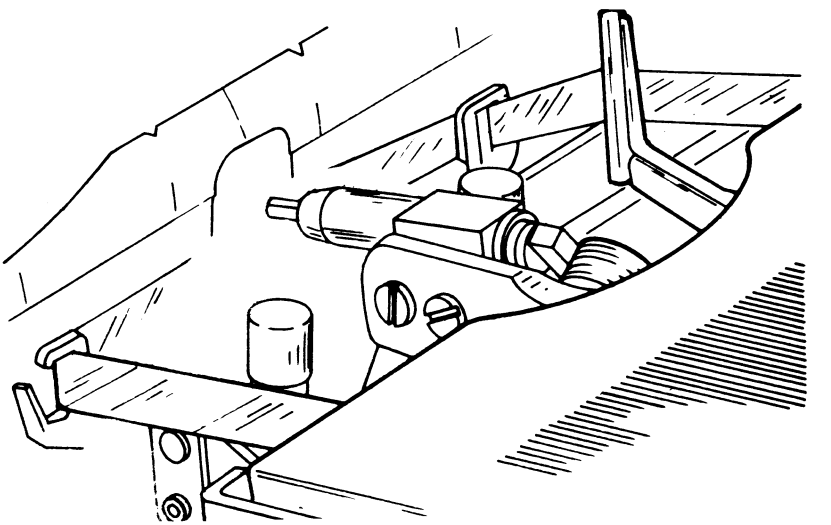


Hier sind einige Zeichen, die man aus einer 5x7-Punktmatrix bilden kann:



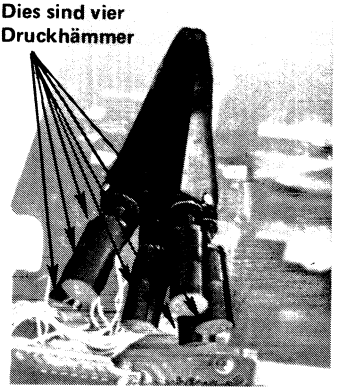
Eine 5x7-Punktmatrix ist ausreichend für Großbuchstaben, Ziffern und spezielle Zeichen, geben jedoch kein schönes Schriftbild bei Kleinbuchstaben. Eine Matrix mit 7 x 9 Punkten ergibt hier eine bessere Auflösung.

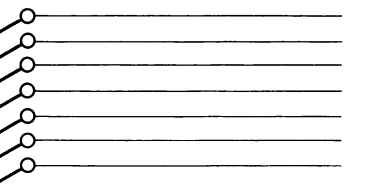
Matrixdrucker können mit einem mechanischen Anschlag oder ohne einen derartigen arbeiten. Ein schlagender Drucker schlägt, wie sein Name sagt, die Typen durch ein Farbband auf das Papier. Für diesen Zweck wird ein sehr kleiner Hammer verwendet:



Zur Bildung von Zeichen mittels einer Punktmatrix werden häufig zwei Verfahren angewendet. Eines verwendet eine vertikale Zeile von Hämmern, die sich über das Papier hinwegbewegen, Zeile für Zeile, und hierbei die Zeichen bilden:

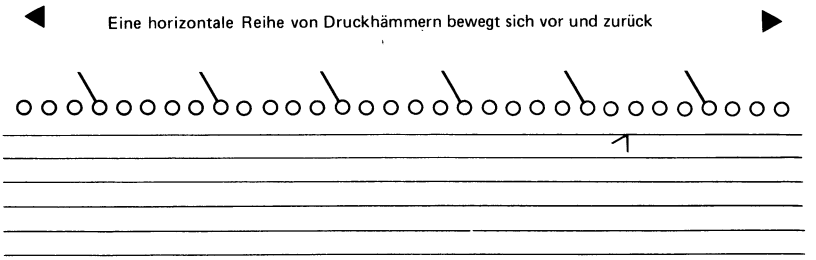
Dies sind vier Druckhämmer





7 Druckhämmer bewegen sich über das Papier und bilden eine einzelne Zeile eines Zeichens

Ein anderer Drucker besitzt eine horizontale Reihe von Hämmern, die sich vorwärts und rückwärts zur Bildung der Zeichen folgendermaßen hin- und herbewegen:



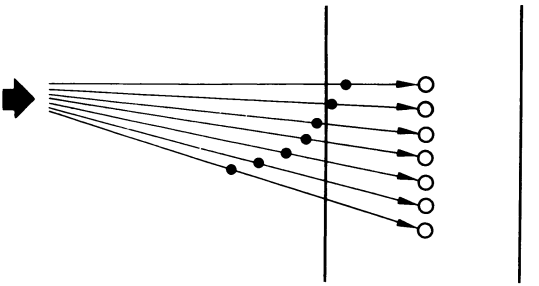
Der Vorteil der schlagenden Drucker besteht darin, daß man von jedem Druck mehrere Kopien herstellen kann, indem man Durchschlagpapier oder druckempfindliches Papier wie bei einer Schreibmaschine verwendet. Der Nachteil dieser Drucker liegt darin, daß sie relativ langsam sind, da sie mechanisch bewegte Druckhämmer benötigen. Ein durchschnittlicher Matrixdrucker druckt mit einer Geschwindigkeit von etwa 30 bis 300 Zeichen pro Sekunde.

Es gibt auch eine Anzahl von Matrixdruckern, bei denen die Zeichen auf das Papier nicht aufgeschlagen werden, sondern die verschiedene sinnreiche Verfahren verwenden, bei denen jedoch kein Anschlagen des Papiers mit einem harten Gegenstand enthalten ist. Diese nicht-schlagenden Matrixdrucker sind sehr schnell, da sie keinerlei mechanische Teile zum Drucken der Punkte benötigen. Sie können deshalb jedoch keine Mehrfachkopien drucken, da kein entsprechend harter Anschlag für Kopien vorhanden ist. **Einige äußerst preisgünstige nicht-schlagende Matrixdrucker verwenden spezielles wärmeempfindliches Papier.** Diese Drucker haben einen Druckkopf, der sich über das Papier bewegt und Wärmeimpulse an den Stellen aussendet, an denen ein Punkt gebildet werden soll.

THERMISCHE DRUCKER TINTENSTRAHLDRUCKER

Drucker, die diese Technik verwenden, werden thermische Drucker oder Thermodrucker genannt.

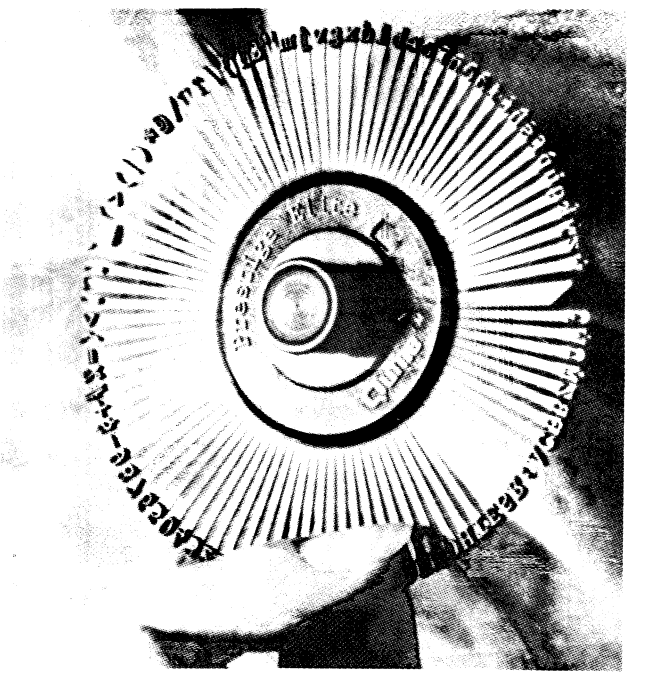
Tintenstrahldrucker sind die gebräuchlichsten Drucker, die heute in der Gruppe der nicht-schlagenden Drucker erhältlich sind. Bei einem Tintenstrahldrucker wird durch eine Düse Spezialtinte mit hoher Geschwindigkeit gedrückt. Ein vor der Düse erzeugtes Hochspannungsfeld teilt den austretenden Tintenstrahl in kleine Tröpfchen, lädt diese elektrisch auf und beschleunigt sie. Die geladenen elektrischen Tröpfchen gelangen in ein elektrisches Ablenkfeld, das ihre Flugrichtung im Sinne der darzustellenden Zeichen ändert, so daß beim Auftreffen auf dem dahinter liegenden Papier diese Zeichen „geschrieben" werden. Dies kann folgendermaßen gezeigt werden:



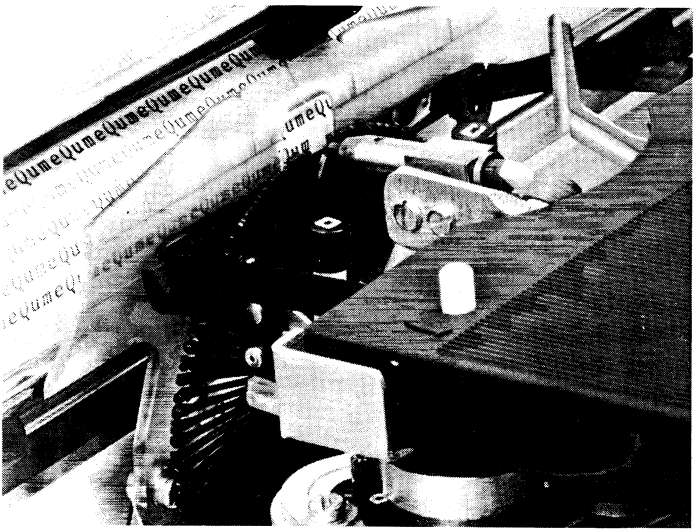
Die Tintentropfen sind außerordentlich klein und trocknen fast unmittelbar. Sie bewegen sich derart schnell, daß ein Tintenstrahldrucker Tausende von Zeichen pro Sekunde drucken kann.

DRUCKER MIT TYPENRAD

Zwei Hersteller (Diablo und Qume Corporation) stellen einen Drucker mit einem Typenrad (Daisy wheel) her. Dieser Drucker verwendet ein Druckelement, das wie ein „Gänseblümchen" mit 96 Blumenblättern aussieht. An jedem Ende dieser Blätter befindet sich ein Zeichen:



Bei diesen Druckern rotiert das Typenrad mit hoher Geschwindigkeit, während es sich über das Papier bewegt. Immer wenn das richtige Typenelement sich vor einer Zeichenposition befindet, wird ein Druckhammer betätigt, der die Type gegen ein Farbband schlägt und dadurch das Zeichen auf das Papier druckt. Dies kann folgendermaßen gezeigt werden:



Der Vorteil dieser Typenraddrucker liegt darin, daß sie ein sehr schönes und sauberes Schriftbild ergeben und sehr zuverlässig und relativ preisgünstig sind. Ein Text, der mit einem derartigen Drucker hergestellt ist, sieht ebenso gut aus wie von einer Selectric- oder Executive-Schreibmaschine. Dies kann man bei den Matrixdruckern nicht sagen. Typenraddrucker können Hunderte von Stunden ohne mechanische Fehler arbeiten, was bei Schreibmaschinen, die als Drucker eingesetzt werden, nicht der Fall ist.

Der einzige Nachteil der Typenraddrucker liegt in ihrer niedrigen Geschwindigkeit. Typische Typenraddrucker arbeiten mit Geschwindigkeiten von 30 und 45 Zeichen pro Sekunde.

Der attraktiv aussehende Text der Typenraddrucker bietet gegenüber Matrixdruckern den Vorteil, daß erstere feste Zeichen verwenden. Ein anderer etwas aufwendiger Weg zur Erzeugung fester Zeichen ist die Verwendung einer Stahlkette.

KETTENDRUCKER

Bei einem Kettendrucker rotiert eine Kette aus Stahl, die die einzelnen Zeichen trägt, mit hoher Geschwindigkeit wobei ein oder mehrere Hämmer von hinten in dem Au-

genblick gegen das Papier schlagen, in dem sich die abzudruckende Type vor der betreffenden Druckstelle befindet.
Der bekannteste Kettendrucker befindet sich im Fernschreiber Teletype Modell 40.

Alle derzeit erhältlichen preisgünstigen Drucker verwenden eine der oben beschriebenen Drucktechniken. Die billigsten Drucker verwenden die Matrixdrucktechnik mit Anschlag. Typenraddrucker und Kettendrucker sind teurer.

GEBRÄUCHLICHE DRUCKERAUSFÜHRUNGEN
LÄNGE DER DRUCKZEILE

Was man auch immer für einen Drucker verwendet, es gibt gemeinsame Möglichkeiten, die man vor der Auswahl eines Druckers wissen sollte:

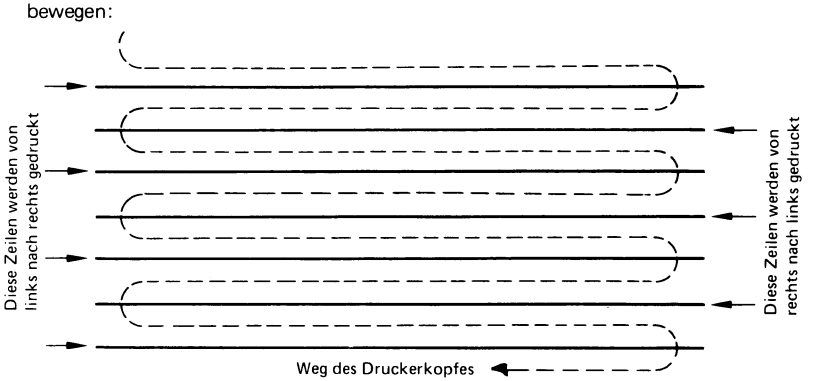
1) **Länge der Druckzeile.** Zeilen mit 80 Zeichen sind am gebräuchlichsten. Manche Drucker besitzen Zeilen mit 132 Zeichen. Sehr billige Drucker besitzen kurze Zeilen mit nur 20 Zeichen und verwenden sehr schmale Papierstreifen.

ZEICHENVORRAT EINES DRUCKERS

2) **Nur Großbuchstaben oder Groß- und Kleinbuchstaben.** Preisgünstige Drucker verwenden meist nur die großen Buchstaben des Alphabets. Teurere Drucker können sowohl Groß- wie Kleinbuchstaben verarbeiten.

ZWEIKOPFDRUCKER
DRUCKEN IN ENTGEGENGESETZTER RICHTUNG

3) **Anzahl der Druckköpfe.** Matrix-Drucker und Typenrad-Drucker, die sich horizontal über das Papier bewegen, besitzen meist teurere Versionen mit doppelter Geschwindigkeit. Die Ausführungen mit doppelter Geschwindigkeit verwenden zwei Druckköpfe, von denen jeder sich über die Hälfte der Zeile bewegt. Viele Drucker, bei denen sich der Druckkopf horizontal über das Papier bewegt, erhöhen ihre Druckgeschwindigkeit, indem sie sich vorwärts und rückwärts über das Papier bewegen:



Im Gegensatz hierzu druckt eine Schreibmaschine immer von links nach rechts, indem am Ende jeder Zeile ein Wagenrücklauf ausgeführt wird.

PROPORTIONALER ABSTAND

4) **Proportionaler Abstand.** Diese Möglichkeit ist meist nur bei Typenraddruckern erhältlich. Bei einem Text mit proportionalem Abstand entspricht der Vorschub der jeweiligen Breite des Zeichens. Der Text in diesem Buch ist zum Beispiel proportional gesetzt. Das folgende Beispiel zeigt den Unterschied:

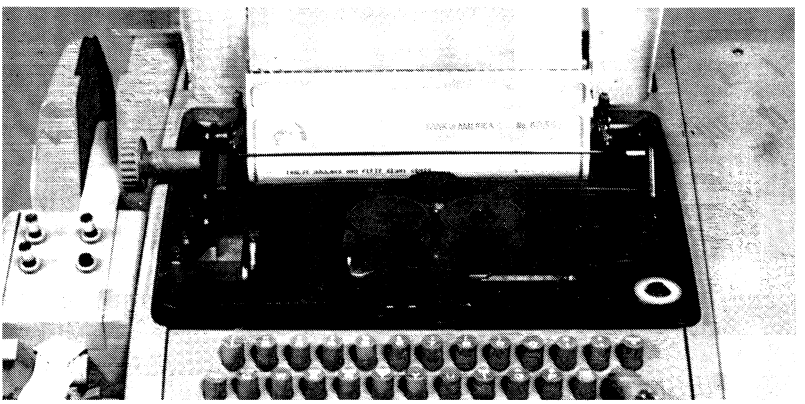
```
(2)  The ending location is an address

(2)  The ending location is an address
```

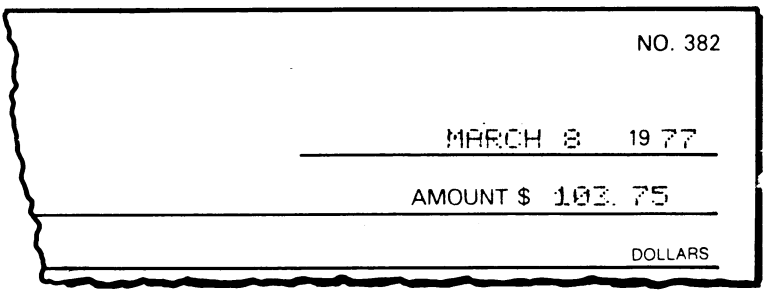
5) **Rechtsbündig.** Bei den meisten Texten bildet der linke und der rechte Rand eine gerade Linie. Während der linke gerade Rand selbstverständlich ist, hat man bei den meisten billigen Druckern und Schreibmaschinen eine ungleiche Zeilenlänge. „Rechtsbündigkeit" oder „Blocksatz" nennt man das Drucken eines Textes, bei dem die Zeilen an einer geraden vertikalen Linie beginnen und enden.

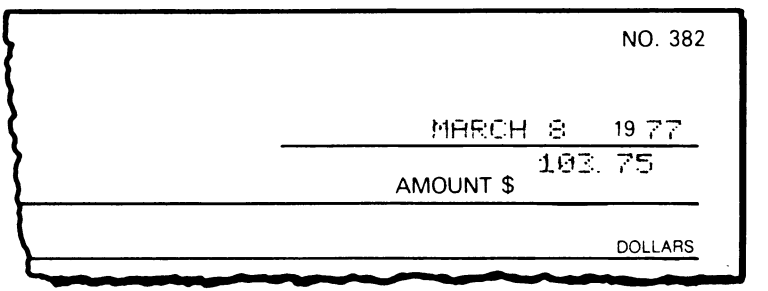
PAPIERVORSCHUB DURCH REIBUNG

6) **Stachel-Papierführung.** Wenn wir kontinuierliche Formulare ausfüllen, wie etwa die Überweisungsformulare von Joe Bitburger, können sehr rasch Probleme bei der richtigen Positionierung des Papiers entstehen. Billige Drucker schieben das Papier durch Reibung vor. Rollen an beiden Seiten des Papiers bewirken hierbei den Vorschub:



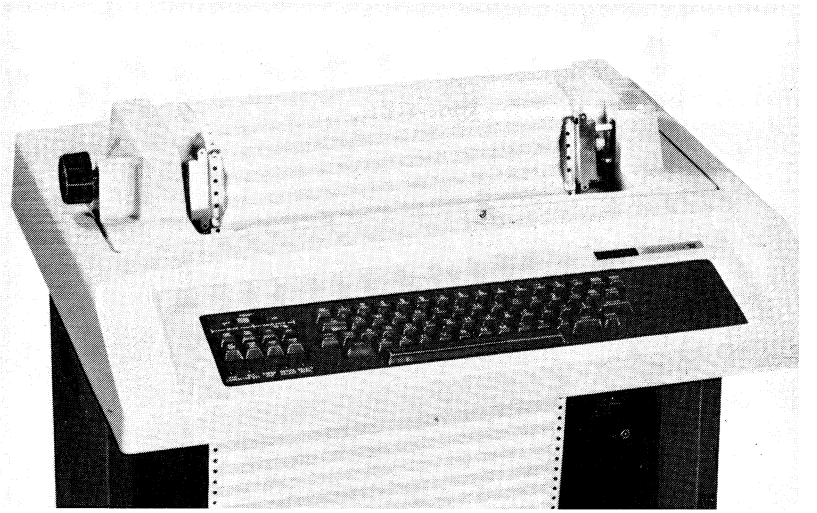
Selbst ein kleines Rutschen in diesem Reibungsvorschub, wie oben gezeigt, kann bewirken, daß sich das Papier nicht in der richtigen Lage befindet. Aufeinanderfolgende Zeilen werden dann weiter und weiter von der gewünschten Stelle bedruckt. In einem Überweisungsformular befindet sich zum Beispiel eine Zeile, in der der Geldbetrag aufscheinen muß. Wenn jede Überweisung nur um einen Millimeter verrutscht, das heißt wenn es beispielsweise sich um einen Millimeter weniger bewegt, dann findet sich nach fünf Formularen der einzudruckende Betrag bereits 5 Millimeter über seiner richtigen Stelle:





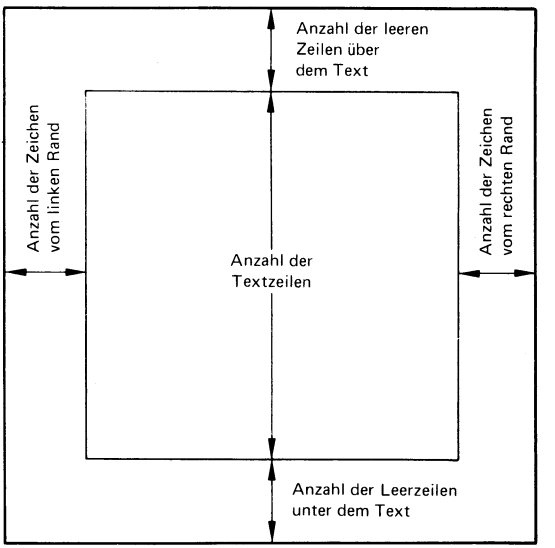
STACHELPAPIER-VORSCHUB

Bald werden die Schecks unleserlich werden. Immer wenn wir kontinuierliche Formulare verwenden, sollte unser Drucker einen Stachelpapiervorschub besitzen:



Ein derartiger Vorschub wird das Papier immer exakt bewegen, egal wie weit es sich bereits insgesamt weiterbewegt hat.

7) **Papierformat.** Einige Drucker besitzen mechanische Hilfsmittel zur Spezifizierung des Druckseitenformates. Das Druckseitenformat enthält Dinge wie die Anzahl der gedruckten Zeilen auf jedem Blatt Papier, den Abstand vom oberen und unteren Rand sowie die Abstände an der linken und rechten Seite. Dies kann folgendermaßen gezeigt werden:



Wenn wir alle beschriebenen Möglichkeiten des Druckers betrachtet haben, müssen wir uns auch noch die Robustheit des Druckers ansehen. Erinnern wir uns daran, daß ein Drucker ein mechanisches Gerät ist und es ist meistens das mechanische Gerät unseres Computersystems, das als erstes ausfällt. Wenn ein Drucker nicht entsprechend robust aufgebaut ist, nützen uns alle schönen Möglichkeiten dieses Druckers nichts, wenn er uns im Stich läßt. Bevor wir einen preisgünstigen Drucker kaufen, müssen wir uns vergewissern, daß der Hersteller nicht an der falschen Stelle gespart hat. Insbesondere müssen wir auf Plastikteile im Druckmechanismus achten. Plastik für das Gehäuse ist sehr schön, jedoch der Druckmechanismus soll zur Gänze aus Metallteilen bestehen.

MASSENSPEICHER

Wir haben drei Massenspeichergeräte besprochen: Lochstreifengeräte, Kassettengeräte und Diskettengeräte. Speicher mit starren Platten werden selten in Mikrocomputer-Systemen verwendet, da sie über die Erfordernisse eines Mikrocomputers hinausgehen, sowohl was die Geschwindigkeit des Datentransfers anlangt, als auch die Menge der gespeicherten Informationen.

LOCHSTREIFENGERÄTE

Die meisten Lochstreifenleser und Stanzer sind preiswerte Geräte mit entsprechenden Eigenschaften. Dies liegt daran, daß Kassettengeräte heute so deutlich überlegen sind, daß niemand bereit ist für ein Lochstreifengerät mehr als für ein Kassettengerät auszugeben. Daher sind die Anforderungen, die man an ein Lochstreifengerät stellen kann, sehr begrenzt.
Das Mikrocomputer-System von Heathkit verwendet einen Lochstreifenstanzer:

Es gibt auch einige äußerst billige mechanische Lochstreifenstanzer, mit denen man einen Lochstreifen von Hand stanzen kann.
Da gebrauchte Fernschreiber heutzutage bereits für DM 1200,– bis 1800,– erhältlich sind, stellt dies keinen großen Anreiz für Hersteller dar, in Zukunft Lochstreifengeräte zu produzieren. Insbesondere da Kassettengeräte ein äußerst preisgünstiges Mittel zur Speicherung großer Datenmengen darstellen. Kassettengeräte sind äußerst billig und die entsprechende Interface-Elektronik ist heute ebenfalls bereits sehr preiswert erhältlich.

KASSETTENGERÄTE

Kassettengeräte als Massenspeicher in Mikrocomputer-Systemen werden in sehr großem Umfang verwendet. Derzeit kosten die billigsten Diskettengeräte, zusammen mit der entsprechenden Interface-Logik etwa DM 2000,–. Die Preise für Diskettengeräte werden rasch weiter absinken, insbesonders wenn die verkauften Stückzahlen steigen. In der Zwischenzeit werden jedoch Kassettengeräte das wichtigste, preisgünstigste Massenspeichergerät darstellen.
Wenn wir ein Kassettengerät für unser Mikrocomputer-System auswählen, so sind zwei wesentliche Punkte zu beachten:

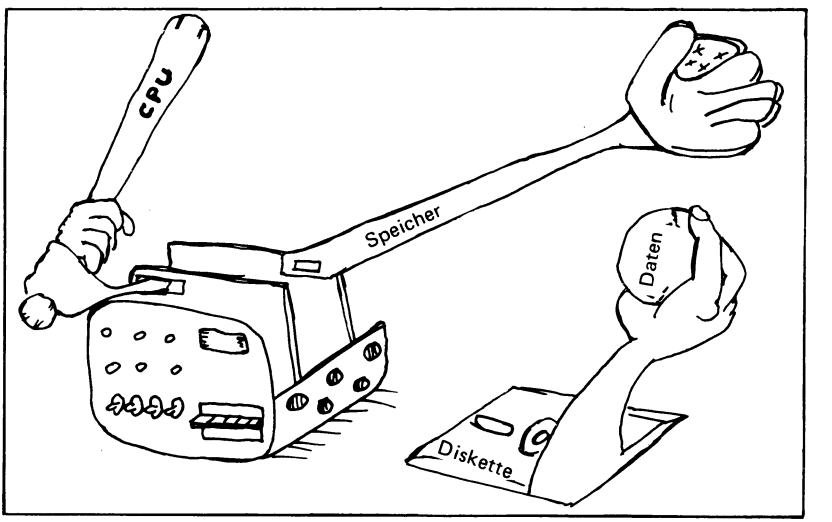
1) **Wir müssen uns vergewissern, daß der Kassettenantrieb von guter Qualität ist.** Billige Kassettenantriebe werden nur Ärger verursachen.
2) **Wir müssen uns vergewissern, daß die Interface-Logik für unser Kassettengerät Informationen in relativ kurzen Aufzeichnungen speichert und hierbei redundante Aufzeichnung verwendet.** Wir haben die Vorteile dieses Verfahrens in Kapitel 1 besprochen. Wir sollten auch sehr vorsichtig bei Kassettengeräten sein, die sehr hohe Datentransfer-Geschwindigkeiten versprechen. In der Regel steigt die Zuverlässigkeit im selben Maß wie die Transfer-Geschwindigkeit abnimmt. Ein langsamer Kassettenantrieb ist meist auch ein zuverlässiger Kassettenantrieb.

GERÄTE-KOMPATIBILITÄT

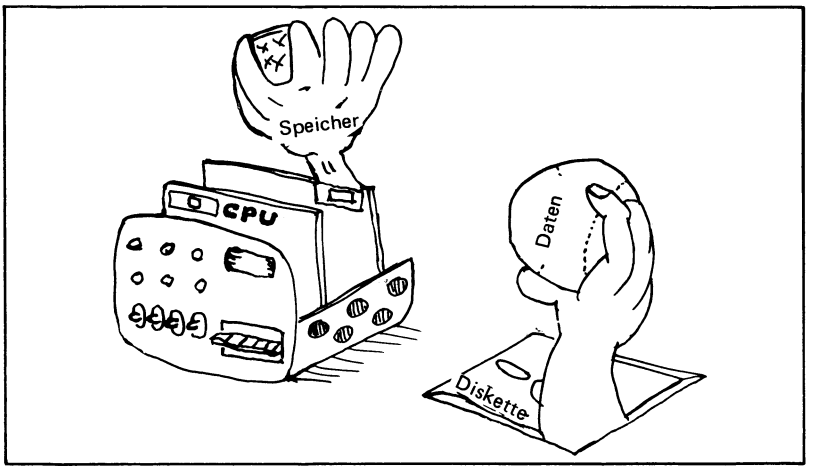
Wenn wir ein Kassettengerät für unser Mikrocomputer-System erwerben, lohnt sich eine Prüfung, inwieweit unser Kassettengerät mit anderen Kassettengeräten zum Überschreiben von Daten und Programmen zusammenpaßt. Man bezeichnet dies als Geräte-Kompatibilität (oder Geräte-Verträglichkeit).
Wenn wir ein Kassettengerät für unser Mikrocomputer-System kaufen, werden wir normalerweise auch eine Liste aller Kassettengeräte erhalten können, die mit unserem kompatibel sind. Die Kompatibilität ist eine sehr wünschenswerte Eigenschaft, da wir nur in diesem Fall Kassetten mit verschiedenen anderen Anwendern austauschen können.

DISKETTEN
(FLOPPY DISK)

Die verschiedenen Eigenschaften von Disketten-Geräten haben wir zum größten Teil bereits in Kapitel 1 besprochen. Wir müssen uns zuerst entscheiden, ob wir ein Mini-Diskettengerät oder ein Standard-Diskettengerät wollen. Wenn diese Entscheidung getroffen ist, sollten wir wieder darauf achten, wieviele Diskettengeräte kompatibel mit unserem sind. Auch das erleichtert den Programmaustausch zwischen verschiedenen Diskettengeräten.
Eine wichtige Eigenschaft, auf die wir beim Kauf eines Diskettengerätes achten sollten, ist die Art und Weise in der die Interface-Logik Transfers von Informationen zwischen dem Diskettengerät und dem Speicher ausführt. Manche Disketten-Controller führen den gesamten Datentransfer über den Mikrocomputer aus:



Dies ist ein langsamer und wenig wünschenswerter Weg zur Bewegung von Daten. Bessere Diskettengeräte können die Informationen direkt zwischen dem Schreib-/Lesespeicher und unserem Diskettengerät ausführen, indem sie den Mikrocomputer umgehen:



Dieser Weg ist wesentlich günstiger. Durch den direkten Zugriff zum Speicher kann die Interface-Logik des Diskettengerätes die Daten wesentlich rascher bewegen, wobei der Mikrocomputer frei für die Ausführung von Programmen während des Datentransfers bleibt.

Wir sollten ebenfalls prüfen, wie ein Diskettenantrieb aufgebaut ist. Erinnern wir uns daran, daß ein Diskettenantrieb ein mechanisches und nicht ein elektronisches Gerät ist. Der Antrieb für eine Diskette ist nicht nur ein mechanisches Gerät, sondern auch ein Gerät mit hoher Präzision, das imstande sein muß, den Lesekopf rasch mit höchster Präzision über die Oberfläche der Diskette zu bewegen. Wenn der Antrieb der Diskette mit billigen Plastikteilen für den Steuermechanismus aufgebaut ist, so wird dieser sicher rasch ausgeleiert sein. Da Diskettengeräte relativ teuer sind, müssen wir uns vergewissern, daß die mechanischen Steuerelemente unseres Diskettengerätes aus hochwertigen metallischen Komponenten aufgebaut sind. Plastik ist schön für das Gehäuse, aber das ist auch alles.

KAPITEL 3

FRAGEN:

1. Eine logische Einheit für die _____ - _____ kann eine Tastatur, die Frontplatte eines Mikrocomputers mit Schaltern, ein Lochstreifenleser etc. sein.
2. Eine logische Einheit für die Ausgabe von Resultaten oder Mitteilungen kann z. B. ein _____ oder eine _____ sein.
3. Diskettengeräte, Kassettengeräte und Lochstreifen stellen logische Einheiten für die _____ von Informationen dar.
4. Die einzige logische und physikalische Einheit, die durch nichts ersetzt werden kann, ist der _____ selbst.
5. Alle Sichtgeräte mit einem Bildschirm verwenden einen sogenannten _____ oder _____, damit man weiß, auf welcher Stelle des Bildschirmes das nächste Zeichen eingeschrieben wird.
6. Der Zeiger kann mit der _____ nach rechts zum Ende der Zeile und mit der _____ auf der Zeile nach links bewegt werden.
7. Einfache Videoanzeigen verwenden nur _____, während teuren Bildschirmgeräten die Darstellung von _____ und _____ möglich ist.
8. Der vertikale _____ gestattet ein Verschieben des Textes auf dem Bildschirm nach oben und unten über den Rand des Bildschirms hinaus.
9. Damit bei einer Tastatur beim gleichzeitigen Drücken zweier Tasten keine Probleme entstehen, verwendet man eine elektronische _____ und _____.
10. Da beim Betätigen einer Taste keine eindeutige Kontaktgabe erfolgt, wird ein sauberes Signal durch elektronisches _____ gebildet.
11. Preiswerte Drucker erzeugen Zeichen meist aus einer _____ von 5 x 7 Punkten und werden daher _____-Drucker genannt.
12. Schlagende Matrixdrucker arbeiten mit einer Reihe von _____ und ermöglichen somit auch die Herstellung von Kopien.
13. Nichtschlagende Matrixdrucker erzeugen die Punkte der Matrix z. B. mit Wärme-Impulsen auf _____ Papier.
14. Beim _____-Drucker wird durch eine Düse Spezialtinte gedrückt und die Flugrichtung des Strahls zur Bildung der Zeichen elektrisch abgelenkt.
15. Sehr sauberen Druck erreicht man mit _____-Druckern, bei denen das _____ mit hoher Geschwindigkeit rotiert und ein Druckhammer im richtigen Moment das entsprechende Zeichen gegen ein Farbband und damit auf das Papier schlägt.
16. Statt eines Typenrades kann man auch eine rotierende _____ aus Stahl, die die einzelnen Zeichen trägt, verwenden.
17. Die wesentlichsten Kennzeichen eines Druckers sind vor allem die _____ der Druckzeile und der _____ des Druckers.
18. Um ein Rutschen des Papiers beim Vorschub zu vermeiden, was sich bei Formularen sehr unangenehm auswirken kann, verwendet man bei Druckern meist den sogenannten _____-Papiervorschub.
19. Bei der Verwendung von Kassetten für Massenspeicher sollte darauf geachtet werden, daß ein Kassettengerät mit anderen Kassettengeräten zum Überschreiben von

Daten und Programmen zusammenpaßt, das heißt _____ ist.

20. Diskettengeräte, die den gesamten Datentransfer über den Mikrocomputer ausführen, sind wesentlich _____ als Geräte, bei denen die Übertragung _____ zwischen Schreib-/Lesespeicher und Diskettengerät erfolgt.
21. Die Mechanik der Disketten-Geräte muß äußerst präzise arbeiten und sollte daher keinesfalls aus Plastik, sondern nur aus hochwertigen _____ Komponenten aufgebaut sein.

ANTWORTEN, KAPITEL 3

1. Informationseingabe
2. Drucker, Videoanzeige
3. Massenspeicherung
4. Mikrocomputer
5. Zeiger, CURSOR
6. Leertaste, Rückwärts-Leertaste
7. Großbuchstaben, Groß- und Kleinbuchstaben
8. Aufrollmodus
9. Verriegelung, Puffer
10. Entprellen
11. Matrix, Matrix-
12. Hämmern
13. wärmeempfindlichen
14. Tintenstrahl
15. Typenrad, Typenrad
16. Kette
17. Länge, Zeichenvorrat
18. Stachel
19. kompatibel
20. langsamer, direkt
21. metallischen

Kapitel 4
WIR KOMMEN ZU DEN GRUNDLAGEN

Wir wollen nun in den Mikrocomputer „hineinschauen" und sehen wie er arbeitet. Wir können ein Mikrocomputer-System kaufen und es als solches verwenden, indem wir Programme in einer Programmiersprache (wie etwa FORTRAN oder BASIC) schreiben. Wenn wir nur das vorhaben, müssen wir nicht wissen, wie unser Mikrocomputer-System arbeitet — und die restlichen Informationen dieses Buches werden für uns völlig bedeutungslos. Dazu brauchen wir nur die ersten paar Seiten von Kapitel 5 zu lesen, in denen verschiedene Programmiersprachen besprochen werden und dann ein Buch auswählen, mit dem wir die gewünschte Programmiersprache lernen können.

Jedoch das Programmieren eines Mikrocomputer-Systems unter Verwendung einer Sprache wie FORTRAN oder BASIC ist so ähnlich wie das Fahren mit einem Bus. Um mit einem Bus zu fahren, brauchen wir nicht zu wissen wie ein Bus funktioniert. Wir brauchen nicht einmal zu wissen wie man einen Bus steuert. Wenn wir jedoch die erhöhte Flexibilität eines Privatautos wollen, so müssen wir fahren lernen. Wir können sogar lernen, wie das Auto funktioniert um uns gegebenenfalls selbst zu helfen. Ähnlich ist es bei einem Mikrocomputer. Wenn wir seine volle Leistungsfähigkeit und Möglichkeiten voll ausschöpfen wollen, müssen wir lernen wie der Mikrocomputer arbeitet.

Wenn wir uns also hierzu entschlossen haben, müssen wir zunächst einmal feststellen, welchen Grad an Kenntnissen wir überhaupt anstreben. Wir können lernen, wie ein Mikrocomputer unter Verwendung einer sehr fundamentalen Sprache auf der Ebene des Mikrocomputers zu programmieren ist. Wir können auch einen Schritt weitergehen und lernen, wie der Mikrocomputer im Detail arbeitet, um gegebenenfalls eine Störung zu beheben — und ihn zu erweitern oder zu ändern, falls er für unsere Zwecke nicht mehr ausreicht.

Im restlichen Teil dieses Buches wird angenommen, daß wir wissen wollen wie Mikrocomputer arbeiten — aber es werden keine Voraussetzungen gemacht ob wir einen Mikrocomputer reparieren oder modifizieren wollen. Immer unter der Voraussetzung, daß wir keine Kenntnisse auf dem Gebiet der Computertechnik besitzen, außer jenen, die wir beim Lesen der Kapitel 1, 2 und 3 erworben haben, beginnen wir nun einige ganz grundlegende Konzepte zu erklären. Diese Grundlagen sollen uns so weit bringen, daß wir das Buch „Einführung in die Mikrocomputer-Technik" lesen und verstehen können. Wenn man daher mehr als nur einen Mikrocomputer mittels einer höheren Programmiersprache programmieren will, so muß man den Rest des Buches lesen, unabhängig davon, was man weiter zu tun gedenkt. Erst beim Lesen weiter fortgeschrittenerer Bücher in dieser Serie können wir unterscheiden, welche Informationen wir benötigen oder nicht, abhängig von unseren Ambitionen.

Wenn wir im Rest dieses Buches grundlegende Konzepte beschreiben, so werden wir uns ständig auf den Mikrocomputer selbst beziehen, und nicht auf die physikalischen

Einheiten, die den Mikrocomputer umgeben. Dies ist zweckmäßig, da wir schließlich und endlich auf jeden Fall lernen müssen wie ein Mikrocomputer zu programmieren ist und gegebenenfalls wie man einen baut. Erst dann werden wir mehr Informationen über Disketten, Tastaturen, Anzeigen und andere physikalische Geräte oder ihre Interface-Logik brauchen. Wir werden die physikalische Einheit und ihre Interface-Logik als ein ganzes Gerät kaufen und wir werden die physikalische Einheit vom Mikrocomputer aus programmieren.

ZAHLEN UND LOGIK

Jede beliebige Logik innerhalb eines Mikrocomputer-Systems kann auf ein Netzwerk von Schaltern reduziert werden, von denen jeder entweder „Ein" oder „Aus" ist. Dieses Konzept ist für uns nicht neu, da wir es bereits zur Beschreibung von Speichern, insbesondere der Festwertspeicher verwendet haben. **Aber sehen wir uns nunmehr an, wie ein Netzwerk von Schaltern schließlich und endlich die Leistungsfähigkeit und Vielseitigkeit eines Computers bewirkt.**

BINÄRE DATEN

Betrachten wir Zahlen. Wie wir bereits früher festgestellt haben, werden Befehle, Programme und alle Arten von Daten zu einer Folge von Zahlen. **Wie können wir nun derartig viele Zahlen darstellen, wenn wir nur die Möglichkeit haben, Schalter auf „Ein" oder „Aus" zu stellen?**
Die Ziffer „0" kann durch einen „Aus"-Schalter dargestellt werden:

„1" kann durch einen „Ein"-Schalter dargestellt werden:

Was soll das? Ein Computer, der nur bis 1 zählen kann wird kaum sehr nützlich sein. **Trotzdem können in der Tat Computer nur zwei separate und bestimmte numerische Ziffern bilden:**

Null	0
Eins	1
Was dann?	

Aber der Mensch hat ein sehr ähnliches Problem. Wir besitzen auch nur zehn getrennte und bestimmte numerische Ziffern:

Null	0
Eins	1
Zwei	2
Drei	3
Vier	4
Fünf	5
Sechs	6
Sieben	7
Acht	8
Neun	9

Nun beginnen wir mit der Kombination von Ziffern!

Zehn	10
Elf	11

etc.

DEZIMALZAHLEN

Das von den Menschen im allgemeinen verwendete Zahlensystem wird als Dezimalzahlensystem bezeichnet. Das Dezimalzahlensystem ist auf der ganzen Welt verbreitet, zwischen völlig unabhängigen Stämmen und Nationen — immer wo eine Gesellschaft gelernt hat zu zählen. Höchst wahrscheinlich ist dies durch das Vorhandensein von 10 Fingern begründet, und wir haben zuerst mit unseren Fingern zählen gelernt. An sich ist nichts „Natürliches" oder „Einzigartiges" am Dezimalsystem. Es ist in der Tat sogar etwas unbeholfen, wenn man damit verschiedene Dinge tun soll. Wir werden später sehen, daß es viel elegantere Wege des Zählens gibt, wobei mit „Eleganz" die Einfachheit und Leichtigkeit bei der Ausführung arithmetischer Operationen gemeint ist.

So wie das menschliche Zählsystem einen Namen besitzt — das Dezimalsystem — so hat auch das Zählsystem der Computer einen Namen — es ist das Binärsystem (oder Dualsystem).

Sehen wir uns nun das Binärsystem im Detail an.

DAS BINÄRE ZAHLENSYSTEM

Das binäre Zahlensystem besitzt nur zwei getrennte und bestimmte Ziffern: 0 und 1. Um Zahlen auszudrücken, die größer sind als 1, folgen wir dem Beispiel der Dezimalzahlen — und verwenden mehr als eine Ziffer. Betrachten wir die Zahl 2. Im Binärsystem wird sie durch die Ziffern 10 dargestellt:

DEZIMAL		BINÄR	
Null	0	Null	0
Eins	1	Eins	1
Zwei	2	Zwei	10
Drei	3		
Vier	4		
Fünf	5		
Sechs	6		
Sieben	7		
Acht	8		
Neun	9		
Zehn	10		

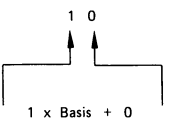
Sowohl im Dezimal- wie im Binärsystem stellt die aus zwei Ziffern bestehende Kombination „10" eine Zahl dar, die um Eins größer als die größte einstellige Zahl ist. Im Falle der Dezimalzahlen ist die größte einstellige Zahl 9. Daher stellt 10 Eins mehr als 9 dar, und das ist Zehn. Im binären Zahlensystem ist die größte einstellige Zahl 1. Deshalb stellt 10 um Eins mehr als Eins dar, und das ist Zwei.

Der numerische Wert der Ziffernkombination „10" ist sehr wichtig. Diese Ziffernkombination hat den Wert Zehn im Dezimalsystem, von dem das Dezimalsystem seinen Namen bekommen hat. Im Binärsystem hat ganz ähnlich die Ziffernkombination „10" den Wert Zwei, von dem das Wort „Binär" kommt.

BASIS DES ZAHLENSYSTEMS

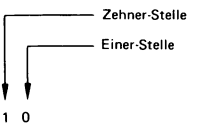
Diese Werte, Zehn für das Dezimalsystem und Zwei für das Binärsystem, werden die „Basis" des Zahlensystems genannt.

Die Basiszahl, das heißt der Wert, der der Ziffernkombination „10" zugeordnet ist, wird für Dezimal- oder Binärzahlen wie folgt interpretiert:



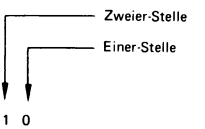
EINER-STELLE
ZEHNER-STELLE

Die Stellen einer zweistelligen Dezimalzahl werden als die „Einer"-Stelle und als die „Zehner"-Stelle bezeichnet:

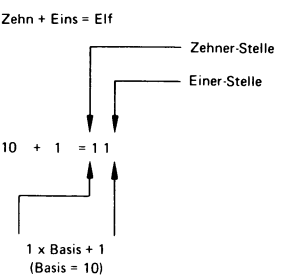


ZWEIER-STELLE

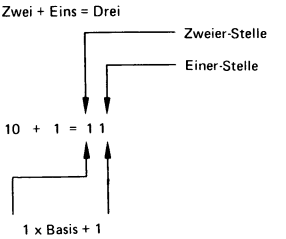
Für eine zweistellige Binärzahl werden die Stellen als „Einer"-Stelle und als die „Zweier"-Stelle bezeichnet:



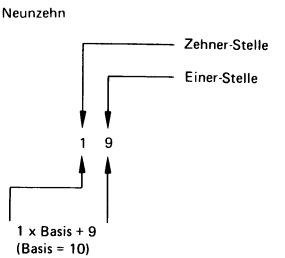
Um die Zahl Drei im Binärsystem darzustellen, können wir weiterhin eine Parallele mit unserem Dezimalsystem ziehen. **Die nächste Dezimalzahl nach der dezimalen Zehn wird durch Addition von 1 folgendermaßen gebildet:**



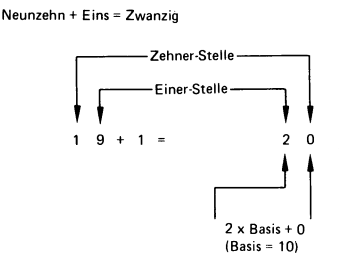
Ähnlich gelangen wir von Zwei zu Drei bei Verwendung von Binärzahlen durch Addition von 1 zum binären Zwei:



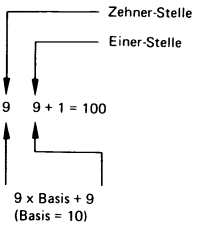
Was geschieht, wenn wir die Zahl 4 im Binärsystem bilden wollen? Die Ähnlichkeit mit dem Dezimalsystem ist nicht so offensichtlich. Wenn wir beim dezimalen 11 weitermachen, haben wir noch eine Weile Zeit, bis Probleme auftreten. Wir können die Addition von 1 so lange fortsetzen, bis wir die Dezimalzahl Neunzehn (19) erreichen:



Dann gelangen wir zur dezimalen Zwanzig (20):



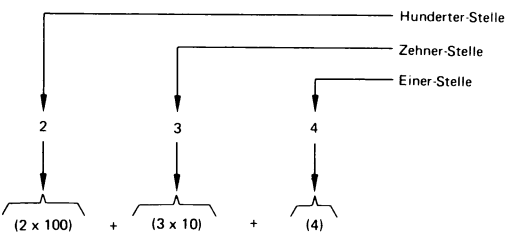
Erst wenn wir die Dezimalzahl 99 erreichen, müssen wir eine dritte Stelle hinzufügen und kommen so zur Zahl 100:



Im Binärsystem haben wir nach der Binärzahl 11 (die dem dezimalen Drei entspricht) ein Problem. Wenn wir 1 zu 11 addieren, können wir nicht zu 12 gelangen, da die Ziffer 2 im Binärsystem nicht existiert. Weiter können wir auch nicht von 11 zu 20 gelangen, da wir wieder die 2 verwenden würden, die im Binärsystem nicht vorhanden ist. **Im Binärsystem folgt daher der Zahl 11 die Zahl 100:**

Null	0
Eins	1
Zwei	10
Drei	11
Vier	100

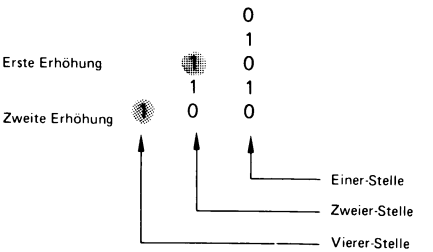
Überprüfen wir daher die Bedeutung von dreistelligen Zahlen.
Wenn wir die Zahl 234 sehen, so interpretieren wir diese automatisch als Zweihundertundvierunddreißig:



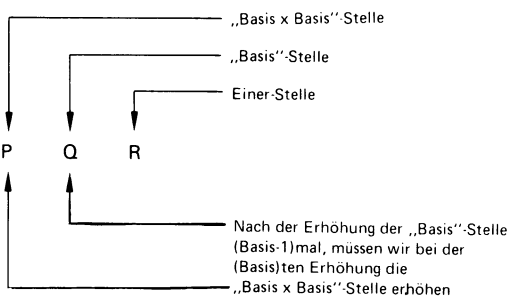
Aber hier gibt es eine besondere Wertigkeit der „Hunderter"-Stelle, so wie wir es bei der „Zehner"-Stelle gesehen haben. Wir können die Zehnerstelle neunmal erhöhen. Bei der zehnten Erhöhung müssen wir die Hunderterstelle verwenden. Betrachten wir die zehn Erhöhungen der Ziffer, beginnend mit der Dezimalzahl Drei:

				3	Drei
Erste	Erhöhung		1	3	Dreizehn
Zweite	Erhöhung		2	3	Dreiundzwanzig
Dritte	Erhöhung		3	3	Dreiunddreißig
Vierte	Erhöhung		4	3	Dreiundvierzig
Fünfte	Erhöhung		5	3	Dreiundfünfzig
Sechste	Erhöhung		6	3	Dreiundsechzig
Siebte	Erhöhung		7	3	Dreiundsiebzig
Achte	Erhöhung		8	3	Dreiundachtzig
Neunte	Erhöhung		9	3	Dreiundneunzig
Zehnte	Erhöhung	1	0	3	Einhundertunddrei

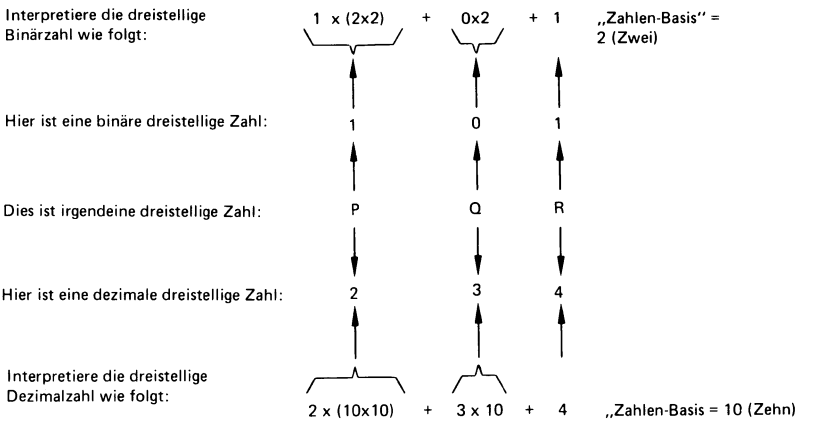
Im Falle der Binärzahlen können wir die „Zweier"-Stelle nur einmal erhöhen. Bei der zweiten Erhöhung müssen wir eine „Vierer"-Stelle schaffen:



Und so sieht die Regel aus: Die Anzahl der möglichen Erhöhungen einer Stelle ist gleich um Eins weniger als die Basis des Zahlensystems. Dann müssen wir die nächsthöhere Stelle erhöhen. Wir können daher in einem Dezimalsystem jede Stelle neunmal erhöhen (0 bis 9). Dann müssen wir die nächstnöhere Dezimalstelle erhöhen. In einer Binärzahl können wir nur einmal erhöhen (0 bis 1), dann müssen wir die nächsthöhere binäre Stelle erhöhen. Daher können Stellen folgendermaßen dargestellt werden:

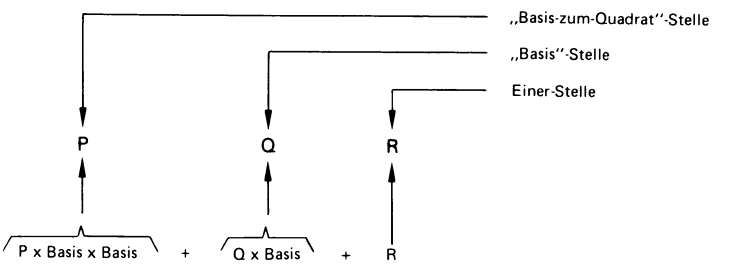


In der obigen Abbildung stellen P, Q und R die Stellen irgendeines Zahlensystems dar. Ersetzen wir „Basis" durch 2 oder 10, so stellt die Abbildung „binäre" oder „dezimale" Zahlen dar.
Die zweite Stelle einer mehrstelligen Zahl wird zu einem „Zahlenbasis"-Multiplikator innerhalb einer Gleichung, die uns den Wert von mehrstelligen Zahlen angibt. Ähnlich wird eine dritte Stelle ein Multiplikator für die mit sich selbst multiplizierte Basis. Dies kann folgendermaßen gezeigt werden:



QUADRAT EINER ZAHL

Eine Zahl, die mit sich selbst multipliziert wird, wird als das „Quadrat" der Zahl bezeichnet. **Wir können daher jede dreistellige Zahl mit folgender allgemeingültigen Gleichung angeben:**



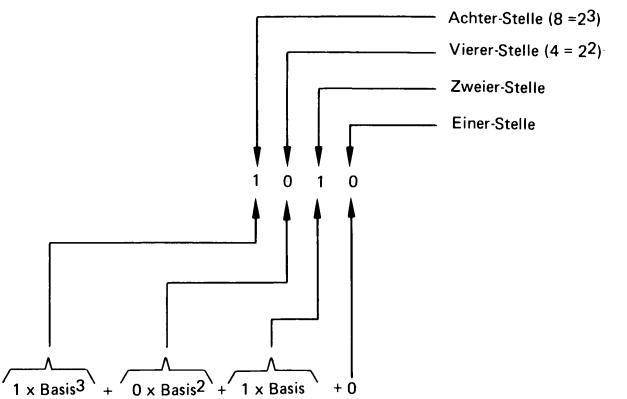
Für „Basis x Basis" verwenden wir das Symbol Basis^2. Daher stellt Basis^2 das Quadrat der Zahl dar, die durch „Basis" dargestellt wird.

TAUSENDER-STELLE
KUBUS EINER ZAHL

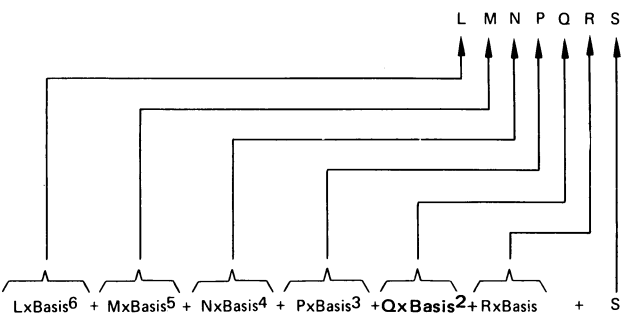
Wir können dieselbe Schlußfolgerung für große Zahlen anwenden.
In der Dezimalarithmetik identifiziert eine vierte Stelle die Tausender und wird als „Tausender"-Stelle bezeichnet. Beispielsweise stellt 2345 den Wert Zweitausenddreihundert-und-fünfundvierzig dar. Eintausend ist 10 x 10 x 10, und ist gleichbedeutend mit „Zahlenbasis" x „Zahlenbasis" x „Zahlenbasis". Dies ist der „Kubus" der Zahlenbasis. Für „Basis x Basis x Basis" verwenden wir das Symbol Basis^3. Daher stellt Basis^3 den Kubus von „Basis" dar.

ACHTER-STELLE

Eine vierstellige Binärzahl wird folgendermaßen interpretiert:



Wir können nunmehr mehrstellige Dezimal- oder Binärzahlen wie folgt definieren:

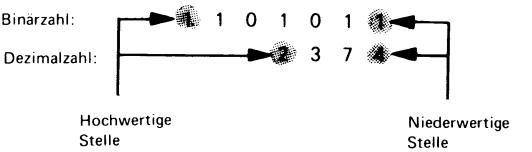


Die obige allgemeine Definition gilt für eine siebenstellige Zahl. Jede andere Anzahl von Stellen kann durch Addition von Stellen am linken Ende der Zahl erfolgen.
Jedesmal wenn wir Basis^n mit Basis multiplizieren erhalten wir Basis^{n+1}. Daher hat „Basis" bis „Basis^6" folgende Werte:

Binär		Dezimal
Zwei	Basis	Zehn
Vier	Basis^2	Einhundert
Acht	Basis^3	Eintausend
Sechzehn	Basis^4	Zehntausend
Zweiunddreißig	Basis^5	Einhunderttausend
Vierundsechzig	Basis^6	Eine Million

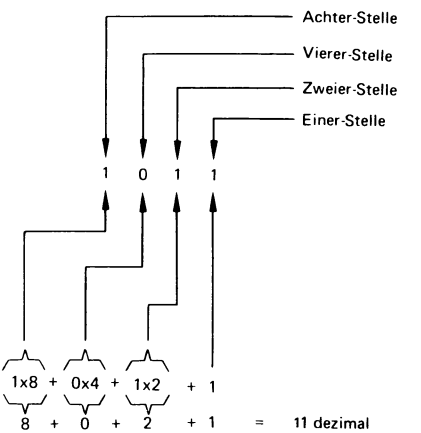
HOCHWERTIGE STELLE
NIEDRIGWERTIGE STELLE

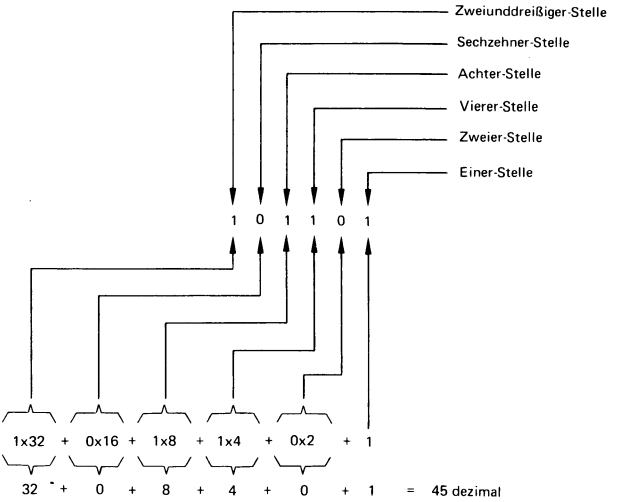
In jeder mehrstelligen Zahl bezeichnen wir die Einer-Stelle, das heißt die am weitesten rechts stehende Stelle, als die „niedrigwertige" Stelle. Die am weitesten links stehende Stelle wird die „hochwertige" Stelle genannt. Dies kann folgendermaßen gezeigt werden:



UMWANDLUNG BINÄR IN DEZIMAL

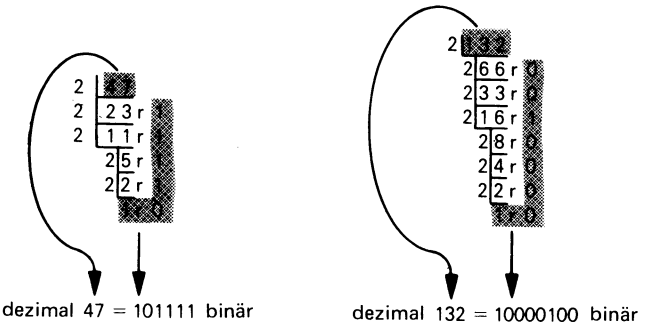
Wir können die allgemeine Darstellung einer mehrstelligen Zahl zur Umwandlung jeder beliebigen Binärzahl in ihr dezimales Äquivalent verwenden.
Hier sind einige Beispiele von mehrstelligen Binärzahlen, mit denen gezeigt wird, wie man aus diesen ihr dezimales Äquivalent berechnet:





UMWANDLUNG DEZIMAL IN BINÄR

Es gibt ein sehr einfaches Verfahren zur Umwandlung jeder beliebigen Dezimalzahl in ihr binäres Äquivalent. Wir wollen zuerst diese Technik einfach darstellen, dann werden wir erklären wie sie arbeitet.
Die Technik ist folgendermaßen definiert: Zur Umwandlung einer Dezimalzahl in ihr binäres Äquivalent ist die Dezimalzahl so oft durch 2 zu dividieren, bis von der Zahl nichts mehr übrigbleibt. Hier sind zwei Beispiele:



Nun wollen wir diese Umwandlungstechnik erklären.
Die oben gezeigten Schritte bilden das mehrstellige binäre Äquivalent der Dezimalzahl. Die niedrigstwertige (das heißt die am weitesten rechts stehende) binäre Stelle wird als erstes gebildet. Diese Stelle ist der Rest, sobald wir wissen wieviele Zweier-Stellen es gibt.
Wir wollen das Symbol NNN zur Darstellung einer beliebigen Dezimalzahl verwenden. Was geschieht, wenn wir NNN durch 2 teilen? Wir bekommen die Hälfte von NNN plus einem Rest von 0 oder 1. Wir wollen das Symbol PPP zur Hälfte von NNN ver-

wenden. In dem allgemeinen Fall dient es zur Darstellung wie NNN durch 2 dividiert wird:

```
2 | NNN
  PPP        Rest 0 oder 1
```

Hier sind einige spezielle Fälle:

```
2 | 421        (NNN = 421)
  210 Rest 1   (PPP = 210)
2 |  36        (NNN = 36)
   18 Rest 0   (PPP = 18)
2 |   7        (NNN = 7)
    3 Rest 1   (PPP = 3)
```

In jeder der oben gezeigten Darstellungen hat die Dezimalzahl NNN insgesamt PPP Zweier-Stellen, plus 0 oder 1:

```
NNN =  PPP x 2 + Rest
421 =  210 x 2 + 1
 36 =   18 x 2 + 0
  7 =    3 x 2 + 1
```

Um festzustellen wieviele Zweier-Stellen es gibt (PPP) teilen wir die Dezimalzahl (NNN) durch Zwei. Der Rest (0 oder 1) ist die Einer-Stelle.
Was ist nun mit PPP? Es ist eine Dezimalzahl. Wenn zufällig PPP gleich 0 oder1 ist, dann ist es auch eine gültige Binärzahl. Jede größere Zahl ist keine gültige Binärzahl. Wenn die Anzahl der im ersten obigen Schritt (PPP) berechneten Zweier-Stellen größer als 1 ist, dann bedeutet dies, daß einige Vierer-Stellen in der Binärzahl enthalten sind. Um zu berechnen, wieviele Vierer-Stellen es gibt, könnte man einfach die anfängliche Dezimalzahl durch Vier dividieren:

```
4 | NNN
  QQQ Rest 3, 2, 1
  oder 0
```

Sehen wir uns wieder obige Beispiele an:

```
NNN =  QQQ x 4 + Rest
421 =  105 x 4 + 1
 36 =    9 x 4 + 0
  7 =    1 x 4 + 3
```

Man beachte aber, daß QQQ die Anzahl der Vierer-Stellen, die Hälfte von PPP, die Anzahl der Zweier-Stellen sein muß. Anders ausgedrückt, die Division von NNN durch 4 ist das gleiche wie die Division der Hälfte von NNN durch 2:

4 | NNN		ist das gleiche wie	2 | NNN	
QQQ	Rest 3, 2, 1 oder 0		2 | PPP	Rest 1 oder 0
			QQQ	Rest 1 oder 0
4 | 421		ist das gleiche wie	2 | 421	
105	Rest 1		2 | 210	Rest 1
			105	Rest 0
4 | 36		ist das gleiche wie	2 | 36	
9	Rest 0		2 | 18	Rest 0
			9	Rest 0
4 | 7		ist das gleiche wie	2 | 7	
1	Rest 3		2 | 3	Rest 1
			1	Rest 1

Der Vorteil der zweimaligen Division durch Zwei ist, daß der Rest immer 0 oder 1 ist – gültige Binärzahlen. Sehen wir uns die oben gezeigten Werte für den Rest an:

1 ist das gleiche wie 01 (binär)
0 ist das gleiche wie 00 (binär)
3 ist das gleiche wie 11 (binär)

Dezimal (7_{10}) wurde zum binären 111_2. Das heißt mit anderen Worten, dezimal sieben ist gleich einer Vierer-Stelle, plus einer Zweier-Stelle plus einer Einer-Stelle:

$$7_{10} = 4_{10} + 2_{10} + 1$$

Bei den Dezimalzahlen 36_{10} und 421_{10} muß in der gleichen Weise fortgesetzt werden, um die entsprechenden binären Stellen zu schaffen, da 9_{10} und 105_{10} keine gültigen binären Zahlen sind. Man setzt einfach die Division durch 2 fort. Hier ist die vollständige Umwandlung für 36:

2 | 36	
2 | 18	Rest 0 (keine Einer-Stelle)
2 | 9	Rest 0 (keine Zweier-Stelle)
2 | 4	Rest 1 (eine Vierer-Stelle)
2 | 2	Rest 0 (keine Achter-Stelle)
1	Rest 0 (keine Sechzehner-Stelle)

eine Zweiunddreißiger-Stelle

Daher: $36_{10} = 32_{10} + 4_{10} = 100100_2$

DEZIMALE SCHREIBWEISE

In den obigen Abbildungen haben wir eine neue Art von Kurzschrift eingeführt, die in den Computerbüchern allgemein verwendet wird. **Dezimalzahlen werden durch den Index 10 am Ende der Zahl identifiziert:**

Dezimal 4713 wird dargestellt als 4713_{10}

BINÄRE SCHREIBWEISE

Binärzahlen werden durch den Index 2 am Ende der Zahl identifiziert:

Binär 11010 wird dargestellt als 11010_2

Da wir fortlaufend die Dezimalzahl durch 2 dividieren, kann der Rest nur 0 oder 1 sein – und der Rest sagt uns, wieviele Einer-Stellen, Zweier-Stellen, Vierer-Stellen und so weiter in dem binären Äquivalent der Dezimalzahl vorhanden sind. Wenn QQQ gleich 2 oder mehr ist, dann gibt es mehr als 0 oder 1 Vierer-Stellen und wir dividieren die Vierer-Stellen (QQQ) durch 2 um zu bestimmen, wieviele Achterstellen vorhanden sind. Der Rest, wenn wir die Vierer-Stellen (QQQ) durch 2 dividieren, sagt uns, ob es 0 oder 1 Vierer-Stellen gibt. Wenn es mehr als eine Achter-Stelle gibt, dann kommen wir zu den Sechzehner-Stellen. Und wenn es mehr als eine Sechzehner-Stelle gibt, so kommen wir zur Zweiunddreißiger-Stelle usw.

Tabelle 4.1 faßt alle möglichen vierstelligen Binärzahlen zusammen und gibt ihr dezimales Äquivalent an.

Sehen wir uns Tabelle 4.1 an, so ist daraus sehr leicht zu ersehen, wie wir Schalter zur Darstellung von Zahlen jeder beliebigen Größe verwenden können. Eine 0 wird ein „Aus"-Schalter, während eine 1 zu einem „Ein"-Schalter wird. Durch einfaches Vergrößern der Anzahl der verwendeten Schalter zur Darstellung einer Zahl, können wir die Größe der Zahlen unbegrenzt erweitern. **Tabelle 4.2 zeigt uns die größte Zahl, die wir im binären Zahlensystem beim Erhöhen der Anzahl der Stellen in der Zahl darstellen können.**

Dezimal-zahlen	Binär-zahlen
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Tabelle 4.1: Alle vierstelligen Binärzahlen und ihre Dezimaldarstellungen.

Anzahl der Binärstellen	Maximaler Binärwert	Dezimales Äquivalent
1	1	1
2	11	3
3	111	7
4	1111	15
5	11111	31
6	111111	63
7	1111111	127
8	11111111	255
9	111111111	511
10	1111111111	1023
11	11111111111	2047
12	111111111111	4095
13	1111111111111	8191
14	11111111111111	16383
15	111111111111111	32767
16	1111111111111111	65535

Tabelle 4.2:
Die größte Zahl, die mit Binärzahlen mit 1 bis 16 Stellen dargestellt werden kann.

Man erkennt, daß bei Hinzufügen eines neuen Schalters (oder einer binären Stelle) die maximal darstellbare Zahlengröße verdoppelt wird.

BITS, „NIBBLES" UND BYTES

BIT

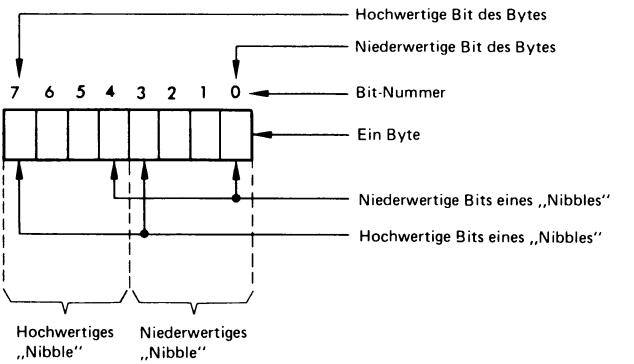
Ein Bit ist eine Abkürzung für Binary digit . Daher kann ein Bit den Wert von 0 oder 1 haben.

Byte

Obwohl Zahlen aus jeder beliebigen Anzahl von binären Ziffern oder Bits gebildet werden können, wie in Tabelle 4.2. gezeigt wurde, gibt es bestimmte Anzahlen von Bits, denen wir häufig begegnen werden. Wir werden sehr oft mit Kombinationen aus 8 Bits zu tun haben. **Eine 8-Bit-Einheit wird als ein Byte bezeichnet.** Es gibt einige unbedeutende Computer, die das Wort „Byte" zur Beschreibung anderer Anzahlen von Bits (am häufigsten 6 Bits) verwenden, jedoch in der Welt der Mikrocomputer ist das Byte immer eine 8-Bit-Einheit. Ein Byte kann daher Zahlen im Bereich von 0 bis 255 darstellen.

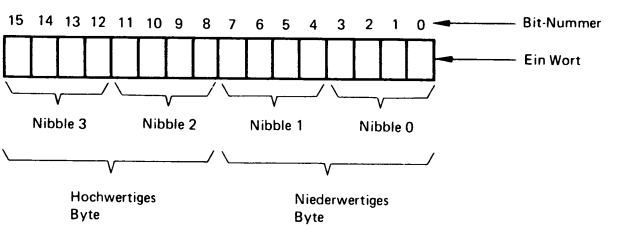
„NIBBLE"

4-Bit-Einheiten werden manchmal als „Nibbles" bezeichnet. Daher besteht ein Byte aus zwei „Nibbles".



WORT

Einheiten aus 16 Bits werden manchmal Worte genannt. Daher besteht ein Wort aus zwei Bytes oder 4 „Nibbles:"

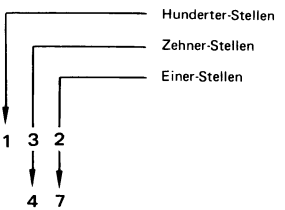


BINÄRE ARITHMETIK

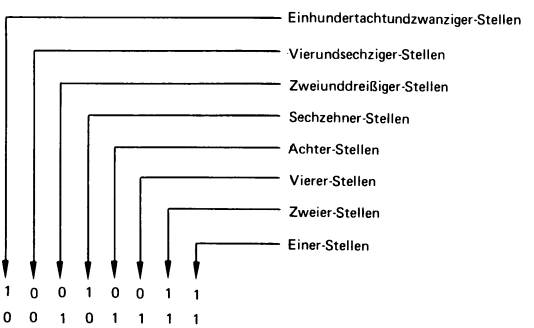
Sehen wir uns nun die Parallelen zwischen der Arithmetik mit Binärzahlen und der Arithmetik mit Dezimalzahlen an.

BINÄRE ADDITION

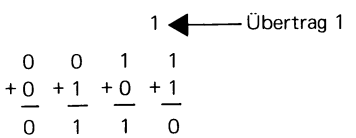
Wenn wir eine dezimale Addition ausführen, so ordnen wir die Stellen der beiden zu addierenden Zahlen folgendermaßen untereinander an:



Genau das gleiche machen wir bei einer binären Addition:



Wir addieren die Zahlen, Stelle für Stelle, indem wir mit der niedrigwertigen (Einer-) Stelle beginnen. Die binäre Addition ist jedoch ziemlich einfach, verglichen mit der dezimalen Addition. Wenn wir zwei Binärziffern addieren, gibt es nur vier Möglichkeiten:



Bei Dezimalzahlen haben wir 100 Möglichkeiten, von denen 45 einen Übertrag erzeugen.

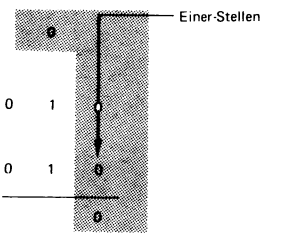
Sehen wir uns einige Beispiele für binäre Additionen an, zuerst den sehr einfachen Fall von 2+2=4. Bei der Verwendung von Binärzahlen erhalten wir folgendes:

Dezimal	Binär
0	000
1	001
2	010
3	011
4	100

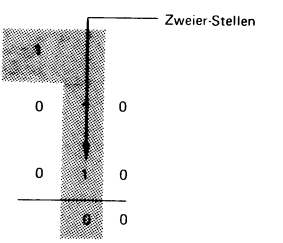
Dezimal	Binär
2	010
+ 2	+ 010
= 4	= 100

Hier ist eine Erklärung, Stelle-für-Stelle, der binären Addition:

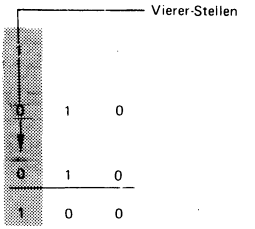
1) Die Einer-Stellen sind beide 0. Die Addition ergibt 0 und keinen Übertrag:



2) Die Zweier-Stellen sind beide 1 und es liegt kein Übertrag von der Einer-Stelle vor. Die Addition der beiden Einer-Bits ergibt 0 und einen Übertrag:



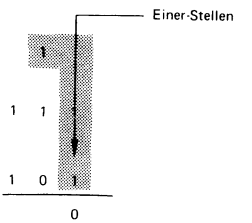
3) Die Vierer-Stellen sind beide 0, es ist jedoch ein Übertrag von den Zweier-Stellen vorhanden. Die beiden Nullen ergeben 0, jedoch die Addition von 1 (dem Übertrag) zu 0 ergibt 1, ohne neuerlichen Übertrag:



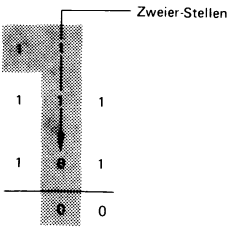
Nun sehen wir uns ein etwas komplexeres Beispiel einer binären Addition an: 7+5=12. Dies kann folgendermaßen gezeigt werden:

Dezimal	Binär
7	111
+ 5	+ 101
= 12	= 1100

Die Einer-Stellen sind beide 1. Ihre Summe ergibt 0 und erzeugt einen Übertrag:



Die Zweier-Stellen sind 1 und 0. Es liegt auch noch ein Übertrag von der Einer-Stelle vor. Die ist gleichbedeutend der Addition von drei Bits: 1, 1 und 0, das 0 und einen Übertrag ergibt:



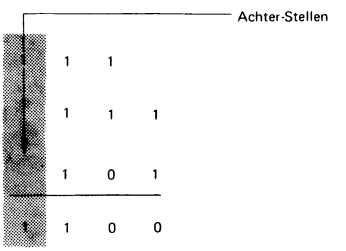
Die Vierer-Stellen sind beide 1, es liegt jedoch noch ein Übertrag von der Zweier-Stelle vor. Die Addition der drei 1-Bits ist sehr einfach, wenn wir sie in zwei Schritten ausführen. Zunächst addieren wir die zwei 1-Bits:

$$\begin{array}{r} 1 \\ 1 \\ \hline 10 \end{array}$$

Dies ergibt 0 und einen Übertrag. Nun addieren wir das dritte 1-Bit zu dem Resultat:

$$\begin{array}{r} 10 \\ 1 \\ \hline 11 \end{array}$$

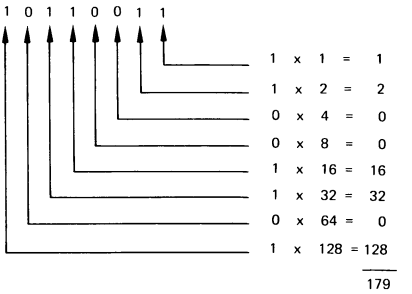
Wir haben also 1 mit einem Übertrag.
Die Achter-Stellen sind beide 0, es liegt jedoch ein Übertrag vor. Die Achter-Stellen summieren sich daher zu 1:



Die Addition von 132 und 47, die wir früher bei der Beschreibung der Mikrocomputer-Logik gezeigt haben, sieht binär folgendermaßen aus:

Dezimal	Binär
132	10000100
+ 47	00101111
= 179	10110011

Die Umwandlung von dezimal in binär für 132 und 47 wurde bereits früher beschrieben. **Wir können prüfen, ob die binäre Summe tatsächlich äquivalent der Dezimalzahl 179 wie folgt ist:**



Wenn wir bis zu diesem Punkt die binäre Addition nicht verstanden haben, so sollten wir weitere Beispiele üben. Für jedes Beispiel schreiben wir den dezimalen Augenden, Addenden und die Summe nieder. Dann bilden wir die binären Äquivalente des dezimalen Augenden und Addenden. Dann summieren wir die binären Äquivalente und wandeln die Summe zurück in einen Dezimalwert. Wenn wir nicht den richtigen Dezimalwert erhalten, so haben wir einen Fehler gemacht und müssen den Rechenvorgang nochmals überprüfen.

BINÄRE SUBTRAKTION UND NEGATIVE ZAHLEN

Die binäre Subtraktion ist wesentlich einfacher als die dezimale Subtraktion, da wir nur vier Möglichkeiten haben. Wir können 0 von 0 subtrahieren, das ergibt 0:

$$\begin{array}{r} 0 \\ -\,0 \\ \hline 0 \end{array}$$

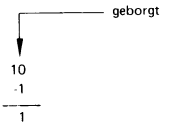
Wir können 1 von 1 subtrahieren, das wiederum ergibt 0:

$$\begin{array}{r} 1 \\ -\,1 \\ \hline 0 \end{array}$$

Wenn wir 0 von 1 subtrahieren, so ist das Ergebnis 1:

$$\begin{array}{r} 1 \\ -\,0 \\ \hline 1 \end{array}$$

Aber was geschieht, wenn wir 1 von 0 subtrahieren? Genau das gleiche wie bei einer dezimalen Subtraktion, müssen wir bei einer binären Subtraktion von dem nächsthöheren Bit wie folgt „borgen":



10 ist die binäre Darstellung von 2. In der obigen Darstellung wird 1 von 2 subtrahiert und ergibt das Ergebnis 1. Wenn kein höheres Bit vorhanden ist, von dem geborgt werden kann, so ist das Ergebnis –1:

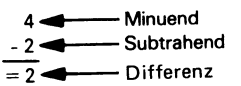
$$\begin{array}{r} 0 \\ -\,1 \\ \hline -\,1 \end{array}$$

Die Erweiterung der Subtraktion für mehrstellige binäre Zahlen ist ebenso einfach wie die Erweiterung der Subtraktion auf mehrstellige Dezimalzahlen: Hier ist das binäre Äquivalent von 4–2=2:

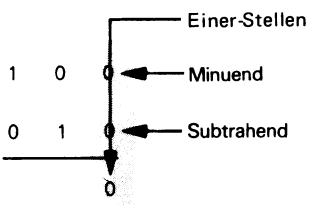
$$\begin{array}{r} 100 \\ 010 \\ \hline 010 \end{array}$$

SUBTRAHEND
MINUEND

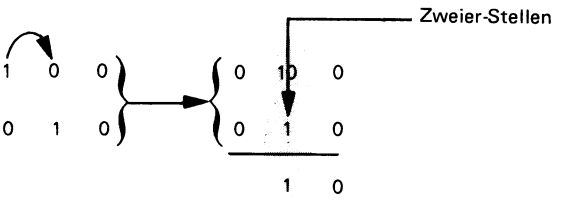
Wenn wir zwei Zahlen voneinander subtrahieren, so subtrahieren wir einen Subtrahenden von einem Minuenden. Dies kann folgendermaßen gezeigt werden:



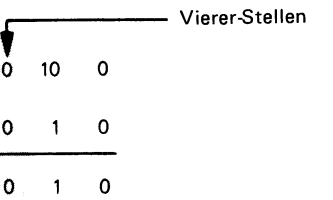
Sehen wir uns die binäre Subtraktion Vier minus Zwei an, so sind die Einer-Stellen beide 0. Daher ist die Differenz 0:



Im Falle der Zweier-Stellen müssen wir 1 von 0 subtrahieren. Wir borgen daher 1 von der Viererstelle des Minuenden und erhalten eine Differenz von 1:



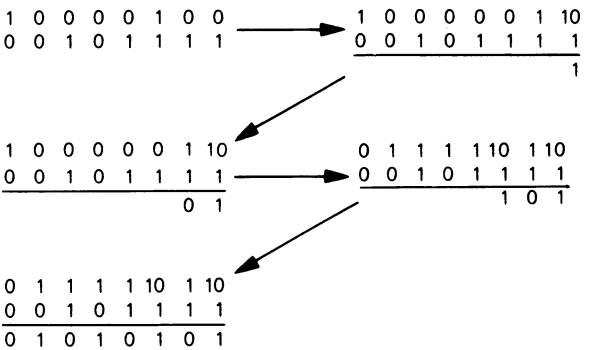
Die Vierer-Stelle des Minuenden ist nun 0. Die 1, die sich dort befunden hat, wurde von der Zweier-Stelle des Minuenden geborgt. Daher wird in der Vierer-Stelle 0 von 0 subtrahiert, und ergibt eine Differenz von 0:



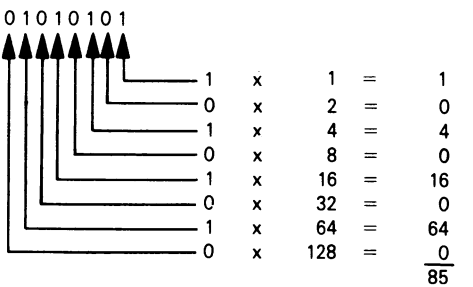
Wir betrachten ein komplexeres Beispiel in dem wir binär die Subtraktion $132_{10} - 47_{10} = 85_{10}$ ausführen:

Dezimal	Binär	
	111	„Entleihungen"
11		
132	10000100	Minuend
- 47	- 00101111	Subtrahend
= 85	01010101	Differenz

Nachfolgend ist die binäre Subtraktion Schritt für Schritt dargestellt:



Wir können das binäre Resultat überprüfen, indem wir sein dezimales Äquivalent bilden:



Es mag nun überraschend sein, daß die meisten Computer nicht subtrahieren können. Sie können nur addieren. Glücklicherweise ist dies kein Problem. Die Subtraktion einer Zahl von einer anderen ist gleichbedeutend mit der Addition des negativen Wertes dieser Zahl. Dies kann folgendermaßen gezeigt werden:

$$4 + -(2) = +(-2)$$

Die obige Lösung mag vielleicht etwas lächerlich aussehen, ist es in der Tat jedoch nicht. Es sieht nur deshalb lächerlich aus, weil wir Dezimalzahlen gewohnt sind. Betrachten wir die Subtraktion 9 minus 3. Wir könnten dies als 9–3 schreiben oder wir könnten es als 9+(0–3) schreiben. In der Tat ist 9+(0–3) komplizierter als 9–3. Um 9+(0–3) zu lösen, müssen wir 3 von 0 subtrahieren, bevor wir das Ergebnis zu 9 addieren. Aber wenn wir 3 von 0 subtrahieren, so erhalten wir –3, das uns zur Subtraktion 3 von 9 zurückführt – was wir auch an erster Stelle getan haben.

Die Subtraktion in der Welt unserer Schalter und binären Zahlen kann nicht gelöst werden, bis wir nicht einen Weg für die Darstellung negativer Zahlen gefunden haben. Ein Schalter ist ein Bauteil mit zwei Zuständen, und wir können nicht einfach einen neuen Zustand des Schalters hinzufügen, der ein negatives Vorzeichen darstellt:

0 1 –

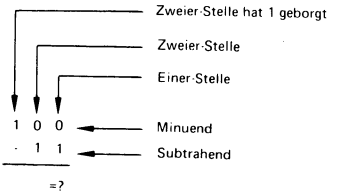
Bis wir kein Hilfsmittel zur Darstellung negativer Zahlen gefunden haben, können wir nicht einmal versuchen, das binäre Äquivalent von 9+(0–3) darzustellen.
Um irgend ein Verfahren zur Darstellung negativer Binärzahlen zu finden, wollen wir zuerst mit dem einfachen Fall von Zahlen im Bereich von 0 bis –7 beginnen. In ihrer positiven Form werden diese Zahlen durch drei binäre Ziffern wie folgt dargestellt:

Dezimal	Binäres Äquivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

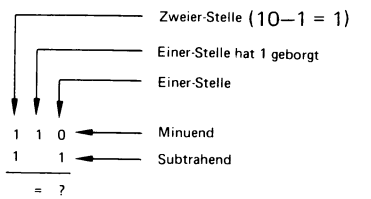
Wir können nicht vollkommen willkürlich eine Methode zur Darstellung negativer Zahlen auswählen. Unsere Forderung besteht darin, daß wir imstande sind zu subtrahieren, indem wir die negative Darstellung der Zahl addieren.
Ein logischer Weg zum Finden der binären Darstellung von negativen Zahlen wäre der Versuch, die positive Zahl von 0 zu subtrahieren. Betrachten wir +3. Die binäre Darstellung hiervon ist 11_2. Sehen wir uns einmal an, was geschieht wenn wir 11_2 von 00 subtrahieren:

```
00 <------ Minuend
-11 <----- Subtrahend
-----
 =?
```

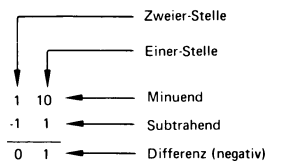
Beginnend mit der Einerstelle wollen wir 1 von 0 subtrahieren. Das ist unmöglich; also versuchen wir 1 von der Zweier-Stelle des Minuenden zu borgen – der aber ebenfalls 0 ist. Wenn wir annehmen, daß wir 1 von der Vierer-Stelle des Minuenden borgen können (links von der Zweier-Stelle), dann erhalten wir folgendes:



Nun kann die Einer-Stelle des Minuenden 1 von der Zweier-Stelle des Minuenden borgen:



Nun können wir die Subtraktion mit Erfolg ausführen:



Die Differenz in der Einerstelle wird als $10_2 - 1_2 = 1_2$ berechnet. Dies entspricht der dezimalen Subtraktion $2_{10} - 1_{10} = 1_{10}$.
Die Differenz in der Zweier-Stelle ist einfach 1 — 1 = 0.
Wir haben bei der Subtraktion 3 von 0 unter Verwendung der binären Arithmetik Erfolg gehabt, müssen aber dieses etwas oberflächliche Vorgehen noch genauer begründen. Wir haben 1 von der nächsthöheren Stelle des Minuenden geborgt (die Vierer-Stelle in diesem Fall) auch wenn eine derartige Stelle gar nicht existierte.
Auch ein weiteres Problem muß nun gelöst werden. Die gezeigten zwei Bits 01 stellen den Wert —3 dar. Sie stellen aber auch den Wert +1 dar.
Diese beiden Probleme können wir mit der bisher behandelten binären Zählweise nicht lösen. **Um eine Subtraktion zu handhaben — wobei zwangsläufig negative Zahlen entstehen können — müssen wir die bisher angenommenen Regeln für das binäre Zählen erweitern.** Aber diese neuen Regeln müssen logisch aufgebaut sein und zahlenmäßig mit den Erfordernissen der positiven Binärzahlen in Einklang stehen sowie die Erfordernisse der binären Arithmetik erfüllen.

Glücklicherweise gibt es eine einfache Lösung. Sehen wir uns einige Dezimalzahlen mit Vorzeichen an:

+10	+123	+47	+83742
−10	−123	−47	−83742

Das einzige neue Kennzeichen, das bei den oben dargestellten Zahlen eingeführt wurde, ist das Vorzeichen. Es gibt ein Pluszeichen (+) und ein Minuszeichen (−). Es wäre denkbar, daß man Binärzahlen mit Vorzeichen folgendermaßen darstellt:

+1011	+1110101	+110	+1011010
−1011	−1110101	−110	−1011010

VORZEICHEN VON BINÄRZAHLEN

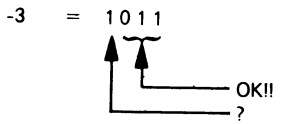
Wir haben nun ein Pluszeichen (+) oder ein Minuszeichen (−), das der Kette von Bits vorausgeht. Aber wir können keine Plus- und Minus-Zeichen als getrennte Gebilde innerhalb der Computer-Logik darstellen. Erinnern wir uns daran, daß die Computer-Logik aus nichts anderem besteht als aus Schaltern mit zwei Zuständen. Wir müssen daher nun auch diese beiden Schalterstellungen zur Darstellung des Plus- und Minuszeichens verwenden. **Wenn wir einen „Aus"-Schalter zur Darstellung eines Pluszeichens, und einen „Ein"-Schalter zur Darstellung des Minuszeichens verwenden, dann wird das Pluszeichen (+) und die Ziffer Null (0) beide durch einen „Aus"-Schalter dargestellt.** Das Minuszeichen (−) und die Ziffer Eins (1) werden beide durch einen „Ein"-Schalter dargestellt. **Unsere Binärzahlen mit Vorzeichen können nun folgendermaßen geschrieben werden:**

01011	01110101	0110	01011010	Positive Zahlen
11011	11110101	1110	11011010	Negative Zahlen

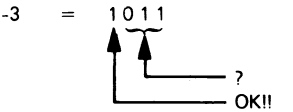
Unglücklicherweise kann man mit dem oben dargestellten Verfahren zur Darstellung negativer Zahlen nicht arbeiten. Betrachten wir das einfache Beispiel 5 − 3 = 2. Erinnern wir uns an das Vorhergesagte, daß wir für eine Subtraktion imstande sein müssen, die negative Darstellung einer Zahl zu addieren. Daher wird 5 − 3 = 2 zu 5 + (−3) = 2. Wird das funktionieren? Wir wollen es versuchen:

Wenn + 5 = 0101 + 3 = 0011 und −3 = 1011
Dann wird + 5 + (−3) = 0101
+ 1011
10000

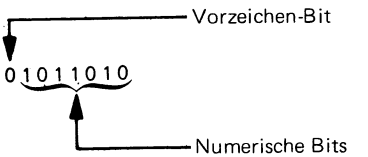
Es funktioniert nicht. Entweder arbeitet die angenommene Methode für die Darstellung des Vorzeichens nicht:



Oder das Verfahren, das wir zur Darstellung des numerischen Teils der negativen Zahl gewählt haben, arbeitet nicht:



Um herauszufinden was hier nicht richtig ist, wollen wir uns die negativen Zahlen noch sorgfältiger ansehen.
Die einzige Möglichkeit mit der wir erkennen können, ob eine binäre Stelle das Vorzeichen einer Zahl oder eine Ziffer innerhalb einer Zahl darstellt, ist das Betrachten der Position der binären Ziffer. **Das am weitesten links stehende Bit muß als Vorzeichen-Bit interpretiert werden:**

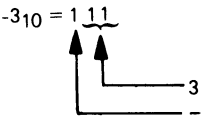


Wie können wir nun den Unterschied zwischen einer Binärzahl mit Vorzeichen und einer Binärzahl ohne Vorzeichen, die um eine Stelle länger ist, erkennen? Die Antwort ist, man kann den Unterschied nicht durch eine einfache Betrachtung feststellen. Wir müssen einfach wissen, womit wir es zu tun haben. Dieses Erfordernis zur Interpretierung von Zahlen ist natürlich neu für uns. Wir haben die Interpretation von Zahlen in Kapitel 2 besprochen, als wir das Programm zur Zahlung von Rechnungen von Joe Bitburger beschrieben haben. Im voraus zu wissen, ob eine Binärzahl eine Zahl mit Vorzeichen oder ohne Vorzeichen ist, ist das gleiche, als ob wir zwischen einem Geldbetrag oder der Kontonummer auf einem Scheck zu unterscheiden haben. Diese Art der Interpretation von Zahlen ist ein ständiges Erfordernis beim Arbeiten mit Mikrocomputern. Wir haben zwischen numerischen Daten mit und ohne Vorzeichen zu unterscheiden sowie viele zahlreiche Möglichkeiten, in denen eine Sequenz von Bits (BInary digiT) zu interpretieren wäre. Dies erhöht sowohl die Leistungsfähigkeit als auch die Komplexität der Computerprogrammierung.
Wenn wir nun zu den Binärzahlen mit Vorzeichen zurückkehren, so wie wir sie definiert haben, so wollen wir nun das Vorzeichen-Bit kritisch betrachten. Bei der Wahl des „Aus"-Schalters zur Darstellung entweder eines Pluszeichens oder eines 0-Bits haben Binärzahlen mit positiven Vorzeichen die gleiche numerische Interpretation wie Binärzahlen ohne Vorzeichen. Dies kann folgendermaßen gezeigt werden:

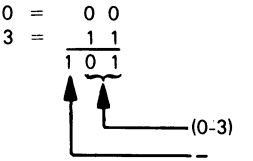
Dezimalzahl		Binärzahl
3_{10}	=	11_2
$+ 3_{10}$	=	011_2

Wenn wir von einer positiven Zahl ohne Vorzeichen zu einer positiven Zahl mit Vorzeichen gehen, so führen wir links eine 0 hinzu, wodurch die Größe der Zahl in keiner Weise verändert wird. Das bedeutet, daß unsere Darstellung des Vorzeichens befriedigend ist. Wie sieht es mit den numerischen Bits aus? Wir haben eine von zwei Wahlmöglichkeiten:

1) Wir können eine 1, die das negative Vorzeichen darstellt, links vor das hochwertige Bit der positiven Zahl stellen:

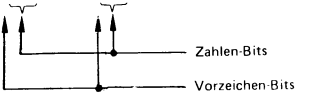


2) Andererseits können wir das binäre Ziffernmuster nehmen, das durch die Subtraktion der positiven Zahl von 0 gebildet wurde, so wie wir es bei der Subtraktion 3 von 0 gemacht haben, und wir können die Eins links vor das höchstwertige numerische Bit stellen:



Wie wir gesehen haben, funktioniert das Verfahren 1), das wir automatisch ausgewählt haben nicht (wie wir durch den Versuch der Subtraktion 3 weniger 5 gezeigt haben). Das Verfahren 2) ist der einfachste Weg. Dieses Verfahren würden wir zur Darstellung von Zahlen im Bereich von 0 bis –7 mittels vier binärer Stellen verwenden:

Dezimalzahlen		Binärzahlen	
Positiv	Negativ	Positiv	Negativ
0	0	0000	0000
1	1	0001	1111
2	2	0010	1110
3	3	0011	1101
4	4	0100	1100
5	5	0101	1011
6	6	0110	1010
7	7	0111	1001



ZWEIER-KOMPLEMENT
BILDUNG DES ZWEIERKOMPLEMENTS EINER ZAHL
EINER-KOMPLEMENT

Die vorher gezeigte Darstellung negativer Zahlen wird als das Zweier-Komplement einer Zahl bezeichnet.
Um das Zweier-Komplement einer Zahl zu bilden, müssen wir nicht die positive Zahl von 0 subtrahieren, dann ein hochwertiges (am meisten links stehendes) 1-Bit zur Darstellung des Minuszeichens addieren. Es gibt ein einfacheres Verfahren. Wir bilden zuerst das Einer-Komplement der Zahl, indem wir jedes Bit (BInary digiT) umdrehen, beziehungsweise invertieren, das heißt die 0-Bits durch 1-Bits zu ersetzen und die 1-Bits durch 0-Bits. Hier sind einige Beispiele von Binärzahlen und ihrem Einer-Komplement:

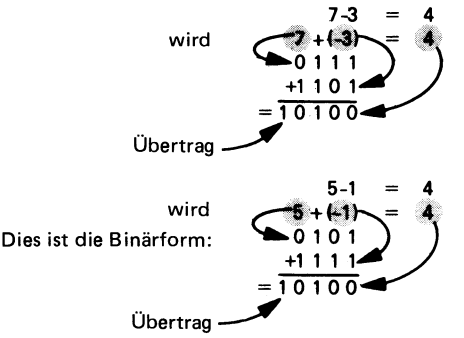
Binärzahl:	101	11101	101010
Einer-Komplement:	010	00010	010101

Wir addieren nun 1 zum Einerkomplement der Zahl und haben so das Zweier-Komplement der Zahl. Hier sind einige Beispiele, die wir mit den oben gezeigten negativen Binärzahlen vergleichen können:

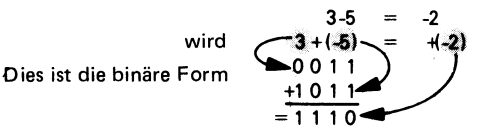
+3	=	0011
Einer-Komplement	=	1100
Zweier-Komplement	=	1101 = −3
+5	=	0101
Einer-Komplement	=	1010
Zweier-Komplement	=	1011 = −5

Nun wollen wir sehen, ob diese binäre Darstellung der negativen Zahlen gültig ist. Wenn sie gültig ist, dann **müssen wir imstande sein eine Zahl zu subtrahieren, indem wir ihr Zweier-Komplement, das heißt ihre negative Darstellung, addieren.** Betrachten wir die Zahlen im Bereich von 1 bis 7. Wenn wir uns versichern, daß wir immer eine kleinere Zahl von einer größeren Zahl subtrahieren, so haben wir hier **einige Beispiele, die beweisen, daß wir eine gültige Darstellung für negative Zahlen entwickelt haben:**

Dezimal	Binär
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111



Das Resultat ist gültig, aber es liegt ein Übertrag vor. Immer wenn eine positive Differenz bei einer Subtraktion gebildet wird, gibt es einen Übertrag von 1.
Was geschieht, wenn wir eine größere Zahl von einer kleineren Zahl subtrahieren? Es geschieht genau das gleiche wie bei einer dezimalen Subtraktion. **Wir erhalten eine negative Zahl. Bei der Subtraktion 5 von 3 geschieht beispielsweise folgendes:**



Wir können sagen, daß das Resultat negativ ist, da die hochwertige Stelle des binären Ergebnisses 1 ist. Erinnern wir uns daran, daß bei positiven und negativen Binärzahlen das hochwertige Bit das Vorzeichen der Zahl darstellt. Ist das hochwertige Bit 0, so ist die Zahl positiv. Wenn das hochwertige Bit 1 ist, wie oben, so ist die Zahl negativ.
Es ist sehr einfach das positive Äquivalent einer negativen Zahl zu bilden. Wir bilden einfach das Zweier-Komplement der negativen Zahl und kommen so zur positiven Zahl zurück. Hier sind einige Beispiele von positiven Zahlen, ihres negativen Äquivalents in Form des Zweier-Komplements und die Wiederherstellung der ursprünglichen positiven Zahl:

+7	=	0111
Einer-Komplement	=	1000
Zweier-Komplement	=	1001 = −7
Einer-Komplement	=	0110
Zweier-Komplement	=	0111 = +7
+4	=	0100
Einer-Komplement	=	1011
Zweier-Komplement	=	1100 = −4
Einer-Komplement	=	0011
Zweier-Komplement	=	0100 = +4

Es besteht überhaupt kein Problem, wenn man das Zweier-Komplement einer Zahl zweimal bildet und die ursprüngliche Zahl zurückerhält. Auch in der Welt der dezimalen Arithmetik ergeben zwei negative Vorzeichen ein positives Vorzeichen. Beispielsweise ist −(−2) gleich +2. Wenn daher die binäre Darstellung, die wir für negative Zahlen entwickelt haben, gültig ist, dann muß das Zweier-Komplement einer negativen Zahl wieder die positive Zahl liefern und das tut es auch.

Wir haben ein sehr elegantes Verfahren zur Handhabung negativer Zahlen und zur Subtraktion von Binärzahlen entwickelt. Aber erinnern wir uns daran, daß Computer noch nicht von vorneherein die Fähigkeit besitzen mit negativen Zahlen zu arbeiten. Sobald wir einen Computer verwenden, müssen wir uns daran erinnern, ob ein binäres Ziffernmuster nur positive Zahlen oder positive und negative Zahlen darstellt.

BINÄRE MULTIPLIKATION UND DIVISION

Wir wollen binäre Multiplikation oder Division nicht detailliert besprechen. Diese Informationen werden nur dann für uns nützlich sein, wenn wir erfahrene Mikrocomputer-Anwender sind. Es gibt jedoch eine sehr interessante Erscheinung, die mit der Multiplikation und Division von Binärzahlen verknüpft ist.

Um eine Binärzahl mit zwei zu multiplizieren, verschieben wir jedes Bit um eine Stelle nach links. Hier ist ein Beispiel:

$$27_{10} = 1B_{16} = 00011011_2$$
$$27_{10} \times 2 = 54_{10} = 36_{16} = 00110110_2$$

Die Verschiebung jedes Bits einer Binärzahl eine Stelle nach rechts ist gleichbedeutend mit einer Division durch Zwei. Hier ist ein Beispiel:

$$36_{10} = 24_{16} = 00100100_2$$
$$18_{10} = 12_{16} = 00010010_2$$

In Wirklichkeit ist gar nichts Überraschendes bei der Verschiebung von Binärziffern zur Multiplikation oder Division durch Zwei. Wir können das gleiche mit Dezimalzahlen machen. Um eine Dezimalzahl mit Zehn zu multiplizieren verschieben wir jede Ziffer eine Stelle nach links:

$$374 \times 10 = 3740$$

Die Verschiebung von Dezimalziffern eine Stelle nach rechts bedeutet eine Division der Zahl durch Zehn:

$$\frac{26730}{10} = 2673$$

Es gibt eine Vielzahl von Verfahren, die man sich zur Multiplikation und Division von Binärzahlen ausgedacht hat. Einige dieser Verfahren sind in der Buchserie „Programming for Logic Design" beschrieben. Wenn wir einen Mikrocomputer in einer höheren Sprache programmieren, so brauchen wir uns niemals um das genaue Verfahren zu kümmern, mit dem binäre Zahlen multipliziert werden, der Kompilierer nimmt uns diese Arbeit ab. (Verschiedene Möglichkeiten von Programmiersprachen werden am Anfang des Kapitels 5 diskutiert.) Warum plagt man sich dann überhaupt mit der Assemblersprache? Die Antwort ist wieder, um eine bessere Kontrolle unseres Programmes zu haben. Es gibt beispielsweise viele verschiedene Programme, die man für die Ausführung binärer Multiplikationen oder Divisionen schreiben kann. Die verschiedenen Multiplikations- und Divisionsprogramme werden uns alle das gleiche Ergebnis liefern. Aber einige arbeiten äußerst schnell, wobei sie jedoch viel Speicherplatz zur Speicherung des Programmes benötigen, während andere zwar langsamer rechnen, jedoch relativ kurze Programme besitzen. Wenn wir eine höhere Programmiersprache verwenden, dann müssen wir hinnehmen, welche Art von Multiplikations- oder Divisionsprogramm diese Sprache verwendet. Wenn wir unser eigenes Programm schreiben, so können wir das Multiplikations- oder Divisionsverfahren auswählen, das entweder am schnellsten arbeitet oder ein Verfahren, das am wenigsten Speicherplatz benötigt.

OKTAL- UND HEXADEZIMALZAHLEN

Man muß schon fast ein mathematisches Genie sein, um mehrstellige Dezimalzahlen und Binärzahlen ineinander umzuwandeln. Diese Umwandlung ist nicht einfach. Das gilt aber nur für den Menschen, es ist für den Mikrocomputer kein Problem. Ein Mikrocomputer arbeitet zur Gänze mit Binärziffern. Er weiß nicht einmal, daß es Dezimalzahlen gibt. Für den Menschen ist es jedoch sehr schwierig mit Binärzahlen zu arbeiten. **Die Manipulation von Binärzahlen ist unbequem, zeitraubend und führt leicht zu Fehlern.** So leicht es einerseits ist, eine 0 in eine 1 umzuwandeln, so leicht kann dies auch irrtümlich geschehen. **Daher hat man Zahlensysteme eingeführt, die kompakter als Binärsysteme sind, aber trotzdem eine einfache Beziehung zu den Binärziffern besitzen. Die beiden am häufigsten verwendeten Systeme sind das „oktale" und das „hexadezimale".**
Jede Oktalziffer stellt genau drei binäre Ziffern wie folgt dar:

Binär	000	001	010	011	100	101	110	111
Oktal	0	1	2	3	4	5	6	7

Das Wort „oktal" stammt von der Zahl 8. Oktalzahlen sind Zahlen mit einer „Basis 8". Daher ist die Zahl 10, die zehn im Dezimalsystem bedeutet, gleich 8 im Oktalsystem.

Wenn wir uns die oben abgebildeten 8 Oktalziffern ansehen, so werden wir uns fragen, welcher Unterschied zwischen Oktal- und Dezimalziffern besteht. Es gibt keinen im Bereich zwischen 0 bis 7, jedoch die Oktalziffern haben keine 8 oder 9. Daher haben mehrstellige Oktal- und Dezimalzahlen nicht dieselben Werte. Dies kann folgendermaßen gezeigt werden:

Dezimal	Oktal
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14

Oktalzahlen werden durch eine tiefgestellte 8 wie folgt gekennzeichnet:

$$11_{10} = 13_8$$

11 (dezimal) = 13 (oktal)

Die Umwandlung von Binärzahlen in ihr oktales Äquivalent ist sehr einfach.
Wir teilen einfach die Binärzahlen in Gruppen zu drei Binärziffern und ersetzen jede Gruppe von drei Binärziffern durch sein oktales Ziffern-Äquivalent. Dies kann folgendermaßen gezeigt werden:

Binär:	110	101	110	111
Oktal:	6	5	6	7

Daher ist $110101110111_2 = 6567_8$
Umgekehrt, wenn wir eine Oktalzahl in ihr binäres Äquivalent umwandeln wollen, ersetzen wir einfach jede oktale Ziffer durch ihr aus drei Binärziffern bestehendes Äquivalent. Dies kann folgendermaßen gezeigt werden:

Oktal	2	5	7	4
Binär	010	101	111	100

$$2574_8 = 010101111100_2$$

Jede hexadezimale Zahl stellt genau vier Binärziffern wie folgt dar:

Binär	0000	0001	0010	0011	0100	0101	0110	0111
Hexadezimal	0	1	2	3	4	5	6	7
Dezimal	0	1	2	3	4	5	6	7
Binär	1000	1001	1010	1011	1100	1101	1110	1111
Hexadezimal	8	9	A	B	C	D	E	F
Dezimal	8	9	10	11	12	13	14	15

Das Wort hexadezimal stammt von der griechischen Bezeichnung für die Zahl Sechzehn. Hexadezimalzahlen sind Zahlen mit der Basis Sechzehn. Die Zahl 10, die Zehn im Dezimalsystem bedeutet und Acht in einem Oktalsystem, ist daher Sechzehn in einem hexadezimalen System. Dies wirft ein neues Problem auf. **Wenn 10 Sechzehn im Hexadezimalsystem bedeutet, dann muß es sechzehn einzelne numerische Ziffern im hexadezimalen System geben, so wie es zehn einzelne numerische Ziffern im Dezimalsystem gibt. Die sechs zusätzlichen hexadezimalen einzelnen numerischen Ziffern werden durch die Buchstaben A, B, C, D, E und F,** wie oben gezeigt, dargestellt. Wir müssen daher zwischen den Buchstaben A bis F zur Darstellung von hexadezimalen Ziffern oder Buchstaben des Alphabets unterscheiden. Wenn dies auch wie eine unnötige Komplikation aussieht, so werden wir finden, daß dies niemals ein besonderes Problem darstellt. Wenn wir uns irgendwelche Daten ansehen, wissen wir automatisch, ob wir Zahlen oder Text vor uns haben. Es wird niemals eine Verwechslung geben.

Wir müssen uns völlig darüber im klaren sein, daß Oktal- und Hexadezimalsysteme sehr nützlich für den Menschen sind, jedoch Computer nicht imstande sind mit diesen Zahlensystemen zu arbeiten, und wir diese daher nur zum Schreiben verwenden. **Computer können nur binäre Daten erkennen.**

Wenn aber Mikrocomputer oktale und hexadezimale Zahlensysteme nicht verstehen, ist es dann wirklich leichter, daß wir diese Zahlensysteme erlernen, anstatt bei Dezimalzahlen zu bleiben und mit Dezimal-Binärumwandlungen zu arbeiten? Die Antwort ist ja, wir tun gut daran das oktale und hexadezimale System zu lernen. Wir wollen diesen Punkt mit einem Beispiel erläutern. Die Dezimalzahl 2735 hat folgendes binäres, oktales und hexadezimales Äquivalent:

	A	A	F		Hexadezimal
Dezimal 2735 =	1010	1010	1111		Binär
	5	2	5	7	Oktal

Wir haben die Standardverfahren beschrieben, die man zur Bildung von Binärziffern aus Dezimalziffern und umgekehrt verwenden kann. Aber diese Verfahren sind zeitraubend, schwerfällig und es ist nicht leicht mit ihnen zu arbeiten, egal wie gut wir Binär- und Dezimalzahlen verstehen. Andererseits können wir sehr einfach eine Umwandlung von oktalen oder hexadezimalen Zahlen in ihre binären Äquivalente ausführen. Sobald wir es gelernt haben in Oktal- und Hexadezimal-Schreibweise zu denken, und das ist wirklich sehr einfach, so haben wir den Vorteil einer kurzen Schreibweise für Daten und einen wirklich einfachen Umwandlungsvorgang, um zu oder von binären Äquivalenten zu gelangen. Man kann dies etwa mit den gleichen Argumenten begründen, die zur Einführung des metrischen Maßsystemes auf der ganzen Welt geführt hat. Es gibt keinen grundlegenden Unterschied zwischen dem Messen einer Entfernung in Kilometern und Metern oder in Meilen und Yards. Ähnlich gibt es keinen grundlegenden Unterschied zwischen dezimaler oder hexadezimaler Zählweise. Aber im einen Fall sind die Umwandlungen sehr schwerfällig, während sie im anderen Fall denkbar einfach sind.

Tabelle 4.3 faßt die Zahlen im Bereich von Null bis Sechzehn zusammen und zeigt ihre binäre, dezimale, oktale und hexadezimale Darstellung:

Hexadezimal	Dezimal	Oktal	Binär
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111
10	16	20	10000

Tabelle 4.6: Zahlensysteme

Den einzigen Gesichtspunkt der oktalen und hexadezimalen Zahlen, den wir beachten müssen, ist die Umwandlung zwischen diesen Zahlen und Binär- und Dezimalzahlen. Die Addition und Subtraktion unter Verwendung oktaler und hexadezimaler Zahlen ist identisch mit der Addition und Subtraktion bei Verwendung von Dezimalzahlen – wobei wir natürlich beachten müssen, daß jedes Zahlensystem einen eigenen Satz von numerischen Ziffern besitzt.

OKTAL–HEXADEZIMAL–UMWANDLUNGEN

Wenn wir Oktal- in Hexadezimalzahlen oder umgekehrt umwandeln wollen, so besteht die einfachste Möglichkeit darin, daß wir einen binären Zwischenschritt einschalten, da Binärzahlen einen Zusammenhang Ziffer für Ziffer sowohl für Oktal- wie Hexadezimalzahlen besitzen. Betrachten wir beispielsweise die Hexadezimalzahl $3C2F_{16}$. Wir können das oktale Äquivalent unter Verwendung der Tabelle 4.3 wie folgt bilden:

Hexadezimal	3	C	2	F
Binär	0011	1100	0010	1111
Oktal	0 3	6	0	5 7

Daher $3C2F_{16} = 36057_8$

Wir können die Oktalzahl 23754 in ihr hexadezimales Äquivalent wie folgt umwandeln:

Oktal	2	3	7	5	4
Binär	010	011	111	101	100
Hexadezimal	2	7	E	C	

Daher $23754_8 = 27EC_{16}$

UMWANDLUNGEN DEZIMAL–OKTAL UND DEZIMAL–HEXADEZIMAL

Die Umwandlung einer Dezimalzahl in ihr oktales oder hexadezimales Äquivalent benutzt die gleiche Logik, die wir bereits für die Bildung des binären Äquivalents einer Dezimalzahl beschrieben haben.

Betrachten wir zuerst die Umwandlung einer Dezimalzahl in ihr hexadezimales Äquivalent. Bei dieser Umwandlung beginnen wir mit der niedrigstwertigen (das heißt der am meisten rechts stehenden) Stelle. Wenn unsere Hexadezimalzahl eine zweistellige Zahl ist, dann würde sie aus Sechzehn (die Basis) multipliziert mit einer festen Ziffer (R) mit einem Rest von S bestehen.

$$RS_{16} = R_{10} \times 16_{10} + S_{10}$$

Um herauszufinden, worin der Rest S besteht, dividieren wir die Dezimalzahl durch die Basis (Sechzehn) wie folgt:

$$16 \overline{)\,NNN}$$

R Rest S

NNN ist eine Dezimalzahl. Sie ist gleich der Hexadezimalzahl RS_{16}. Hier ist ein reales numerisches Beispiel:

$$16 \overline{)\,124}$$

7 Rest 12

Daher $124 = 7 \times 16 + 12$

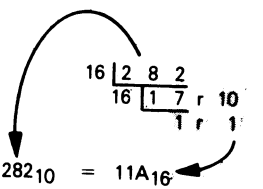
deshalb $124_{10} = 7C_{16}$

Daher sind wir imstande die Ziffern R und S zu bilden und wir haben eine vollständige Umwandlung dezimal in hexadezimal. Dezimal 124 ist gleich hexadezimal 7C.

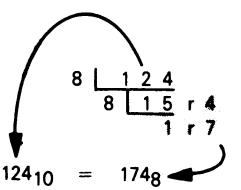
Nun betrachten wir die größere Dezimalzahl 282_{10}. Wenn wir 282_{10} durch 16_{10} dividieren, so erhalten wir:

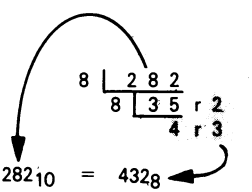
$$16 \mid 282$$
$$17 \text{ Rest } 10$$

Der Rest (S) ist dezimal 10, wie wir aus Tabelle 4.3 entnehmen können und hat das hexadezimale Äquivalent A_{16}. So weit so gut. Jedoch R, der Multiplikator für die Basis, besitzt das dezimale Äquivalent 17. Wie wir aus Tabelle 4.3 entnehmen können, gibt es hier keine einzelne hexadezimale Ziffer, die den Dezimalwert 17 darstellt. Um das hexadezimale Äquivalent zu bilden, müssen wir daher zum nächsten Schritt gehen und 17_{10} durch 16_{10} dividieren:

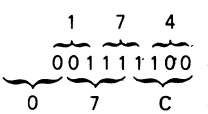


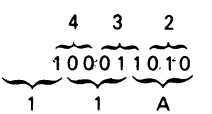
Nun wollen wir die gleichen beiden Dezimalzahlen, 124_{10} und 282_{10}, in ihr oktales Äquivalent umwandeln. Da Oktalzahlen die Basis Acht besitzen, müssen wir wiederholt die Dezimalzahl durch 8_{10} dividieren, um das oktale Äquivalent zu bilden. Dies kann folgendermaßen dargestellt werden:





Nun können wir uns vergewissern, ob unsere Umwandlungen richtig sind, indem wir die oktalen und hexadezimalen Äquivalente von 124_{10} und 282_{10} über ihre binären Zwischenwerte prüfen:





$124_{10} = 174_8 = 7C_{16}$ $\qquad$ $282_{10} = 432_8 = 11A_{16}$

Die Oktal- und Hexadezimalumwandlungen sind in der Tat richtig.

Die Umwandlung von Oktal- und Hexadezimalzahlen in ihr dezimales Äquivalent ist sehr einfach, wenn wir uns daran erinnern, was die verschiedenen Ziffern einer Oktal- oder Hexadezimalzahl darstellen. Um die Dinge einfach zu halten, betrachten wir vierstellige Zahlen. Die dezimale Darstellung jeder vierstelligen Zahl kann durch folgende Gleichung definiert werden:

$$\text{Dezimalwert} = P \times (\text{Basis})^3 + Q \times (\text{Basis})^2 + R \times \text{Basis} + S$$

Eine Zahl mit mehr als vier Stellen würde einfach Ausdrücke links in der Gleichung mit höheren Potenzen der Basis besitzen. Nun kann die allgemeine Gleichung für vierstellige Zahlen für die speziellen Fälle der Oktal- und Hexadezimalzahlen wie folgt umgeschrieben werden:

$$\text{Dezimalwert} = P \times 8^3 + Q \times 8^2 + R \times 8 + S \text{ für Oktalzahlen}$$
$$\text{Dezimalwert} = P \times 16^3 + Q \times 16^2 + R \times 16 + S \text{ für Hexadezimalzahlen}$$

Um eine Oktal- oder Hexadezimalzahl in ihr dezimales Äquivalent umzuwandeln, multiplizieren wir jede Ziffer der Oktal- oder Hexadezimalzahl mit dem entsprechenden Basis-Multiplikator. Hier sind einige Beispiele:

$$\begin{aligned}
2473_8 &= (2\text{x}\,8^3 + 4\text{x}8^2 + 7\text{x}8 + 3)_{10} \\
&= (2\text{x}512 + 4\text{x}64 + 7\text{x}8 + 3)_{10} \\
&= (1024 + 256 + 56 + 3)_{10} \\
&= 1339_{10} \\
149A_{16} &= (1\text{x}16^3 + 4\text{x}16^2 + 9\text{x}16 + 10)_{10} \\
&= (1\text{x}4096 + 4\text{x}256 + 9\text{x}16 + 10)_{10} \\
&= (4096 + 1024 + 144 + 10) \\
&= 5274_{10}
\end{aligned}$$

ZEICHENCODES

Die Schalter mit zwei Zuständen, die jeder Mikrocomputer zur Bildung von Binärziffern verwendet, muß auch für die Darstellung von Buchstaben des Alphabets und jedem anderen Zeichen, das dargestellt, gedruckt oder sonstwie gehandhabt wird, verwendet werden. **Wie wir am Beginn dieses Kapitels festgestellt haben, besteht die Computerlogik aus nichts weiter als einer Anordnung von Schaltern mit zwei Zuständen, so daß wir bei der Darstellung von Zeichen keine andere Möglichkeit haben als Schalter (und damit Binärziffernmuster) zu verwenden, um auch Zeichen darzustellen.**

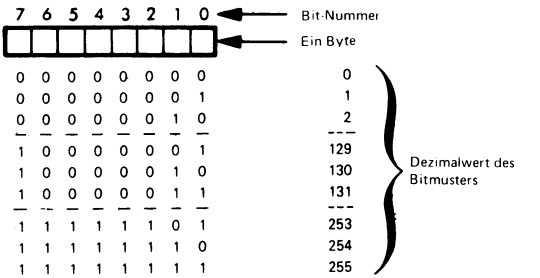
Um mit einer vernünftigen Zeichencodiertechnik zu arbeiten, müssen wir zwei Probleme näher betrachten:

1) **Gibt es irgendeine „natürliche" Methode zur Darstellung von Zeichen, so wie es für die Darstellung binärer Daten der Fall ist?**
2) **Wie können wir zwischen einem Muster aus Binärziffern, die ein Zeichen darstellen, und einem Muster aus Binärziffern, das Zahlen oder andere Informationen darstellt, unterscheiden?**

Es gibt kein „natürliches" Verfahren zur Darstellung von Zeichen unter Verwendung von Binärcodes, und jeder Binärcode den wir bilden, könnte auch als Binärzahl interpretiert werden. Wieder wird vom Programmierer verlangt, daß er im voraus weiß, was eine numerische binäre Ziffernfolge darstellt.

Und wieder können wir sicher sein, daß die mehrfache Verwendung von binären Ziffern niemals zusätzliche Probleme schafft.

Die verschiedenen Codes zur Darstellung von Zeichen verwenden alle ein Byte (8 binäre Ziffern) zur Darstellung eines einzigen Zeichens. Ein Byte hat 256 verschiedene Kombinationsmöglichkeiten von 8 binären Ziffern:



Daher kann ein Byte interpretiert werden als:

1) Eine positive Zahl mit einem Dezimalwert von 0 bis +255.
2) Eine Zahl mit Vorzeichen mit einem Dezimalwert von −128 bis +127.
3) Ein Byte einer aus mehreren Bytes bestehenden Zahl mit oder ohne Vorzeichen.
4) Ein Zeichencode.

ASCII

Das bekannteste Codierschema, das zur Darstellung von Zeichen verwendet wird ist als der „American Standard Code for Information Interchange", allgemein abgekürzt mit den Buchstaben ASCII, bekannt. Der vollständige ASCII-Code für alle druckbaren Zeichen ist im Anhang A angeführt.

Nur die ASCII-Zeichencodes für die numerischen Ziffern 0 bis 9 besitzen einen logischen Zusammenhang. Wenn wir uns diese Zeichencodes ansehen, werden wir finden, daß die vier niederwertigen Ziffern genau gleich dem mit dem Zeichen verbundenen numerischen Wert sind:

Binärcode	Hexadezimales Äquivalent	ASCII-Zeichen
00110000	30	0
00110001	31	1
00110010	32	2
00110011	33	3
00110100	34	4
00110101	35	5
00110110	36	6
00110111	37	7
00111000	38	8
00111001	39	9

Wir werden bemerken, daß im Anhang A zur Darstellung jedes Zeichencodes im Binärmuster dessen hexadezimales Äquivalent verwendet wird. Dadurch wird der Zeichencode wesentlich leichter lesbar. **Wir können das numerische Äquivalent eines Textes unter Verwendung von Hexadezimalzahlen wie folgt zeigen:**

```
T h i s   i s   t h e   n u m e r i c
54 68 69 73 20 69 73 20 74 68 65 20 6E 75 6D 65 72 69 63 0D

e q u i v a l e n t   o f   a
65 71 75 69 76 61 6C 65 6E 74 20 6F 66 20 61 0D

t e x t   s t r i n g
74 65 78 74 20 73 74 72 69 6E 67 2E
```

Jeder Buchstabe des Textes hat unterhalb die beiden Hexadezimalziffern, die den ASCII-Code für den entsprechenden Buchstaben, wie er im Anhang A definiert wurde, darstellt. Wir sehen, daß zwischen den Worten der Hexadezimalcode 20_{16} aufscheint. Das ist der Code für einen Zwischenraum. Der Code $0D_{16}$ stellt einen Wagenrücklauf dar.

KETTE (REIHE)

Das Wort „Kette" oder „Reihe" (engl. string) wird häufig zur Beschreibung einer Folge von Zeichen verwendet, die über ihre numerischen Codes gespeichert sind. Daher wurde auch im obigen Text dieser als „text string" bezeichnet.

In einem Computer wird unser Text als eine Folge von Bits (binary digits) gespeichert. Wenn wir diesen Text drucken wollen, so holen wir die Bits in der richtigen Reihenfolge heraus und übertragen sie auf eine Anzeige oder einen Drucker. Die der Anzeige oder dem Drucker zugehörige Logik interpretiert die Binärdaten, wobei sie annimmt, daß sie Zeichen darstellen.

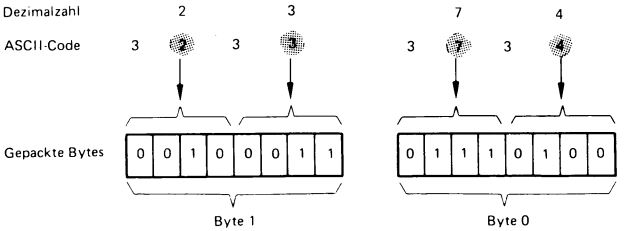
Wenn wir den Text über eine Tastatur eingeben, dann wird jedesmal beim Drücken einer Taste der zugehörige binäre Zifferncode zum Mikrocomputer gesendet – der diesen Code im entsprechenden Speicher aufbewahrt.

Wir können die Zeichencodes modifizieren, indem wir sie wie binäre Daten behandeln. Angenommen wir hätten zum Beispiel eine große Anzahl von numerischen Daten, die wir in einem Speicher aufbewahren wollen, und wir hätten keine alphabetischen Daten. Wenn wir uns wieder die ASCII-Tabelle im Anhang A ansehen, so werden wir finden, daß alle Dezimalziffern die gleichen vier hochwertigen Bits besitzen:

Der ASCII-Code für die Dezimalzahl N ist $3N_{16}$.

GEPACKTE BYTES

Wir können eine Menge Speicherplatz sparen, indem wir Dezimalziffern packen, zwei pro Byte wie folgt:



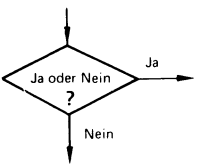
Durch die Ausführung der in Bild 4.1 dargestellten Operationen an unseren Zeichencodes, können wir die ASCII-Zeichen wieder herstellen.

COMPUTERLOGIK UND BOOLESCHE OPERATIONEN

Ein Mikrocomputer verwendet sehr wenig seiner Zeit für arithmetische Operationen. In der Tat gibt es viele Programme, die keinerlei arithmetische Operationen enthalten. Ein Computer wird seine Zeit meist für die Ausführung „logischer" Operationen aufwenden.

STATUS-FLAGS

Wenn wir die in Kapitel 2 gezeigten Flußdiagramme für die Programmlogik untersuchen, so werden wir häufig folgenden Entscheidungsschritt sehen:



Ein gebräuchliches Verfahren zur Handhabung einfacher Zweiweg-Entscheidungslogik, wie oben gezeigt, besteht darin, daß man dem Mikrocomputer einige spezielle Schaltungen zur Verfügung stellt, die man „Status-Flags" nennt (wörtl. Zustands-„Flaggen" oder Kennzeichen). Vorgänge, die diesem logischen Schritt vorausgehen, müssen einen dieser Schalter auf „Aus" oder „Ein" stellen. Die Logik, die eine Entscheidung in zwei Richtungen trifft, wird dann zu einem einzigen Befehl der folgendermaßen dargestellt werden kann:

„Wenn das Flag „Ein" ist, verzweige zu Befehl x.
Wenn das Flag „Aus" ist, gehe zum nächsten Befehl weiter."

Status-Flags stellen eine der einfachsten Formen der Mikrocomputer-Logik dar. Verschiedene Mikrocomputer haben verschiedene Anzahlen und Arten von Status-Flags, die wir jedoch hier im einzelnen nicht zu besprechen brauchen.

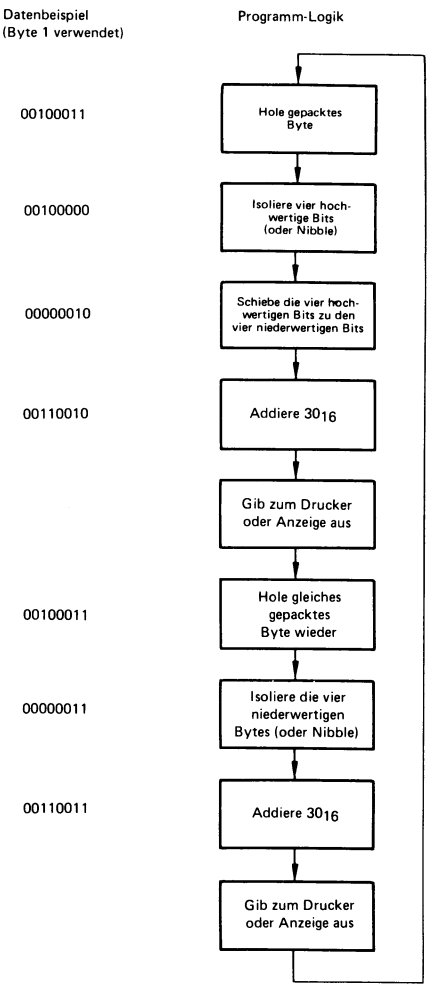
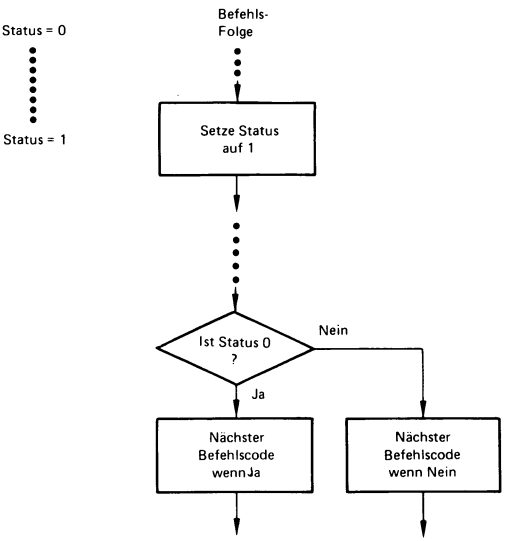


Bild 4.1 Zerlegung gepackter Byte und ASCII-Code-Bildungslogik

Um das Prinzip eines Status-Flags zu verstehen, brauchen wir nur daran zu denken, daß es ein Zweiweg-Schalter ist, der Befehle in unserem Programm entweder auf „Ein" oder „Aus" schalten kann. Nachfolgende Befehle in unserem Programm überprüfen den Schalter um festzustellen, welcher der beiden Wege unserer Programmlogik genommen werden soll. Dies kann folgendermaßen gezeigt werden:



LOGISCHE OPERATOREN

LOGISCHE OPERATOREN

Die meiste Computerlogik wird durch vier „logische Operatoren" gebildet. An diesen „logischen Operatoren" ist nichts Mysteriöses. Addition, Subtraktion, Multiplikation und Division sind „arithmetische Operatoren". Ein logischer Operator nimmt die eingegebenen Daten, führt mit ihnen etwas nicht-arithmetisches aus und bildet ein Resultat – genau wie ein arithmetischer Operator Dateneingaben nimmt, mit ihnen irgend etwas arithmetisches tut und ein Resultat bildet.
Es gibt vier logische Operatoren: Sie sind der NICHT-, UND-, ODER- und der Exklusiv-ODER-Operator. Wir wollen jeden logischen Operator einzeln besprechen.

DER NICHT-OPERATOR

Der NICHT-Operator ist am einfachsten zu verstehen. Dieser Operator sagt einfach: Bringe einen Schalter in seine entgegengesetzte Stellung, das heißt, wenn er „Ein" ist, schalte auf „Aus" und wenn er „Aus" ist, schalte ihn auf „Ein". **Wenn wir uns die Auswirkung des NICHT-Operators auf ein Bit (binäre Ziffer) ansehen, so wandelt er eine 1 in eine 0 oder eine 0 in eine 1.** Dies kann folgendermaßen gezeigt werden:

NICHT 0 = 1
NICHT 1 = 0
NICHT 101101 = 010010

Häufig wird ein Querstrich über einer Zahl anstelle des NICHT verwendet. Dies kann folgendermaßen gezeigt werden:

$\overline{0} = 1$
$\overline{1} = 0$
$\overline{101101} = 010010$

EINER-KOMPLEMENT

Der NICHT-Operator bildet das Einer-Komplement einer Zahl. Erinnern wir uns daran, daß der erste Schritt bei der Bildung des Zweier-Komplements einer Zahl die Bildung eines Einer-Komplements der Zahl ist.

DER UND-OPERATOR

Der UND-Operator prüft zwei Schalter, ob beide „Ein" sind. UND-Operationen können folgendermaßen definiert werden:

0 UND 0 = 0
0 UND 1 = 0
1 UND 0 = 0
1 UND 1 = 1

Sehr häufig wird ein Punkt (.) anstelle des Wortes UND verwendet. Die vier oben gezeigten UND-Operationen können daher folgendermaßen neu geschrieben werden:

$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \quad 1 = 1$

Der Logik der UND-Operationen begegnen wir häufig in unserem Alltagsleben. Ich habe beispielsweise zwei kleine Söhne, Jan und Paul, die beide bereits trotz ihrer Jugend eine Menge argumentieren können. Wenn wir in einem Supermarkt einkaufen gehen, und die beiden Jungen nur eine Zuckerstange kaufen können, dann wird diese Zuckerstange auf Grund einer UND-Operation ausgewählt. Das heißt, wir kaufen die Zuckerstange X nur wenn Jan eine Zuckerstange X UND Paul eine Zuckerstange X will:

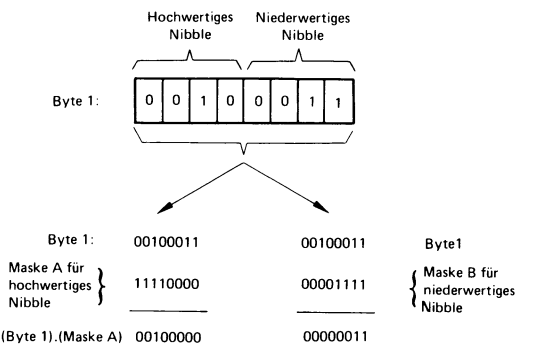
Jan will Zuckerstange A und
Paul will Zuckerstange X. $A \cdot X = 0$
Es wurde keine Zuckerstange ausgewählt.

Jan will eine Zuckerstange X und
Paul will eine Zuckerstange X $X \cdot X = 1$
Wir kaufen die Zuckerstange X.

Wenn entweder Jan oder Paul keine Zuckerstange will, dann wird auf Grund der UND-Logik keine Zuckerstange ausgewählt.

BIT-MASKE

Wir haben also eine Anwendung eines Mikrocomputers für die UND-Operation gesehen. Erinnern wir uns an die Erklärung, wie Zeichencodes aus numerischen Ziffern je zwei pro Byte gespeichert werden können. **In Bild 4.1 könnten wir eine 8-Bit-Maske und eine UND-Operation zur Isolation eines oder der anderen numerischen „Nibbles" verwenden.** Dies kann für Byte 1 in Bild 4.1 wie folgt gezeigt werden:



Wenn ein 0-Bit in eine Folge von Datenbits eingefügt werden muß, so liefern wir ein 0-Bit in einer UND-Maske. Wo auch immer die UND-Maske ein 1-Bit hat, läßt sie die Datenbits unverändert passieren. Nehmen wir beispielsweise an, daß wir die Bits 2, 3, 4 und 5 eines Bytes bewahren wollen, andererseits aber wollen, daß die Bits 0 und 1 und die Bits 6 und 7 den Wert 0 haben sollen. Wir würden dann unsere binären Daten mit der Maske 00111100 UNDieren. Dies kann folgendermaßen gezeigt werden:

	7	6	5	4	3	2	1	0 ◄ Bit-Nr.
Binäre Daten:	X	X	X	X	X	X	X	X
Maske:	0	0	1	1	1	1	0	0
Daten-UND-Maske:	0	0	X	X	X	X	0	0

DER ODER-OPERATOR

Die logische ODER-Operation dient zum Prüfen jedes „Ein"-Schalters. Die ODER-Operation kann folgendermaßen definiert werden:

0 ODER 0 = 0
0 ODER 1 = 1
1 ODER 0 = 1
1 ODER 1 = 1

Das Plus-Zeichen (+) wird häufig anstelle von ODER verwendet. Die vier oben gezeigten ODER-Operationen können daher folgendermaßen neu geschrieben werden:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

Die Tatsache, daß ein Pluszeichen zur Darstellung des logischen ODER-Operators verwendet wird, könnte uns vielleicht verwirren, ist jedoch irrelevant für einen Computer. Es gibt bestimmte Befehle mit einem eigenen unabhängigen Befehlscode, der eine Additionsoperation oder eine logische ODER-Operation darstellt. Das Pluszeichen, das zur Darstellung der Addition oder einer logischen ODER-Operation verwendet wird, wird nur auf gedrucktem oder geschriebenem Material aufscheinen. Einer ODER-Operation können wir wieder in unserem Alltagsleben begegnen. Wenn wir beispielsweise unseren Vorrat an Speiseeis ergänzen wollen, so werden wir das Eis X kaufen wenn entweder Jan oder Paul es möchte:

Möchte Jan Zuckerstange X?		Möchte Paul Zuckerstange X?		Wird Zuckerstange X gekauft?
Nein (=0)	+	Nein (=0)	=	Nein (=0)
Nein (=0)	+	Ja (=1)	=	Ja (=1)
Ja (=1)	+	Nein (=0)	=	Ja (=1)
Ja (=1)	+	Ja (=1)	≙	Ja (=1)

Wir können den ODER-Operator in unserem Beispiel für Zeichen für Dezimalziffern (Bild 4.1) verwenden, um die vier hochwertigen Bits des ASCII-Zeichencodes zu liefern. Dies kann folgendermaßen gezeigt werden:

	7	6	5	4	3	2	1	0 ◀—Bit-Nr.
Isoliertes Nibble:	0	0	0	0	0	0	1	1
ODER-Maske für die vier hochwertigen Bits:	0	0	1	1	0	0	0	0
	0	0	1	1	0	0	1	1

Überall dort wo ein 1-Bit in eine Folge von Datenbits eingesetzt werden muß, sehen wir ein 1-Bit in einer ODER-Maske vor. Wo immer die ODER-Maske ein 0-Bit besitzt, gehen die Daten ungeändert durch. Dies kann folgendermaßen gezeigt werden:

	7	6	5	4	3	2	1	0 ◄— Bit-Nr.
Binäre-Daten:	X	X	X	X	X	X	X	X
Willkürliche Maske:	0	0	1	1	1	1	0	0
Daten-ODER-Maske:	0	0	X	X	X	X	0	0

Daher kann der UND-Operator als „Lösch"-Maske verwendet werden, während der ODER-Operator als „Einfügungs"-Maske eingesetzt werden könnte.

DER XOR-OPERATOR

Der letzte logische Operator, den wir beschreiben wollen ist das Exklusiv-ODER oder XOR. Das Exklusiv-ODER prüft auf Unterschiede und Änderungen. Es könnte folgendermaßen definiert werden:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Das ⊕ -Symbol wird häufig anstelle der Buchstaben XOR verwendet. Daher können die vier oben gezeigten XOR-Operationen wie folgt neu geschrieben werden:

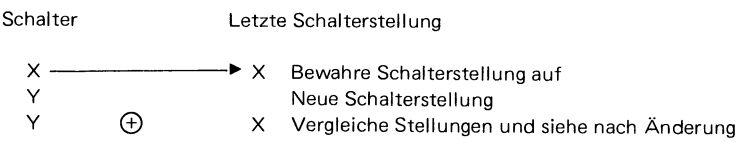
0 ⊕ 0 = 0
0 ⊕ 1 = 1
1 ⊕ 0 = 1
1 ⊕ 1 = 0

Das Exklusiv-ODER ist ebenso Bestandteil unseres täglichen Lebens. Es identifiziert unterschiedliche Meinungen. Beispielsweise würde ein Streit entstehen, wenn Jan ja und Paul nein sagt:

Jans Wunsch		Pauls Wunsch		Streit?
Nein (= 0)	⊕	Nein (= 0)	=	Nein (= 0)
Nein (= 0)	⊕	Ja (= 1)	=	Ja (= 1)
Ja (= 1)	⊕	Nein (= 0)	=	Ja (= 1)
Ja (= 1)	⊕	Ja (= 1)	=	Nein (= 0)

In der Computerlogik verwendet man die Exklusiv-ODER-Operation zur Prüfung auf Änderungen eines Zustandes. Nehmen wir beispielsweise an, daß das Wissen, ob ein Schalter „Ein" oder „Aus" ist, unzureichend ist. Wir müssen auch wissen, ob der Schalter ein- oder ausgeschaltet wurde, seitdem wir ihn zum letzten Mal geprüft

haben. Wir können die Bedingung jedes Schalters bei jeder Prüfung aufbewahren und dann die Schalterstellung mit dieser aufbewahrten Bedingung wie folgt vergleichen:



Daher können wir bei der Ausführung einer Exklusiv-ODER-Operation an den Schalterbedingungen und der vorausgehenden Stellungen herausfinden, ob sich die Schalterstellungen seit unserer letzten Prüfung geändert haben.

KAPITEL 4

FRAGEN:

1. Im Binärsystem gibt es nur zwei bestimmte Ziffern, nämlich___und___.
2. Den Wert 2 für das Binärsystem oder 10 für das Dezimalsystem, nennt man die _____des jeweiligen Zahlensystems.
3. Der Binärzahl 101101 entspricht die Zahl_____im Dezimalsystem.
4. Das binäre Äquivalent der Dezimalzahl 63_{10} ist gleich_____.
5. Die kleinste Darstellungs-Einheit für Binärdaten nennt man ein_____.
6. Eine Kombination aus 8 Bits wird in der Computertechnik im allgemeinen ein____ genannt.
7. Bei einer binären Addition ist:
   0 + 0 = ___
   1 + 0 = ___
   1 + 1 = ___
8. Die Addition der beiden Binärzahlen 1111_2 und 0001_2 ergibt ___.
9. Wenn Binärzahlen mit einem Vorzeichen versehen werden, so bedeutet ein Pluszeichen die Ziffer___und ein Minuszeichen die Ziffer___.
10. Die Subtraktion im Binärsystem wird durch die Bildung des sogenannten _____ durchgeführt.
11. Um das Zweierkomplement einer binären Zahl zu erhalten, wird zunächst das____ _____dieser Zahl gebildet, zu dem dann___addiert wird.
12. Das Einerkomplement der Zahl 1010_2 ist_____.
13. Das Zweierkomplement derselben Zahl ist dann_____.
14. Wenn wir also die Differenz $1012 - 010_2$ bilden wollen, so erhalten wir _____.
15. Das Einmaleins der binären Arithmetik ist besonders „schwer" zu erlernen, es ist:
    1 x 0 = ___
    0 x 1 = ___
    1 x 1 = ___
16. Die Multiplikation zweier Binärzahlen unterscheidet sich nicht von der uns bekannten Multiplikation von Dezimalzahlen. So ist z.B.
    1110_2 x 101_2 = _____.
17. Die Multiplikation einer Binärzahl mit 2 bedeutet einfach die Verschiebung jedes Bits um eine Stelle nach_____. Daher ergibt sich beispielsweise bei der Multiplikation von 011_2 mit 2 das Ergebnis_____.
18. Die Verschiebung jedes Bits einer Binärzahl eine Stelle nach rechts ist gleichbedeutend mit einer_____durch 2.
19. Da Binärzahlen manchmal sehr unhandlich werden können, faßt man drei binäre Ziffern zu_____-Ziffern oder vier Binärziffern zu_____-Ziffern zusammen.
20. Hexadezimalzahlen verwenden die Ziffern 0, 1, 2, 3, ... 9, und weiter _____.
21. Die Binärzahl 0110 1111_2 lautet daher in hexadezimaler Schreibweise_____.
22. Ein Byte, das aus 8 binären Ziffern besteht, kann auf verschiedene Art und Weise interpretiert werden und zwar als:
    1. Eine positive Zahl von _____.

2. Eine Zahl mit Vorzeichen von _____.
3. Ein_____.

23. Das bekannteste Codierschema zur Darstellung von Zeichen ist der allgemein anerkannte_____-Code (gesprochen meist „Aski").
24. Um einem Computer bei der Ausführung eines Programmes an einem bestimmten Punkt des Programmes eine Entscheidung zu ermöglichen, verwendet man sogenannte _____.
25. Die meiste Computerlogik wird durch sogenannte _____gebildet. Die wichtigsten hiervon sind NICHT,_____,_____,_____und Exclusiv-ODER.
26. Beim NICHT-Operator bedeutet einfach
    NICHT 0 = ____
    NICHT 1 = ____
    Der NICHT-Operator bildet das_____einer Zahl.
27. Beim UND-Operator ergibt sich nur dann 1 als Resultat, wenn beide oder mehrere "Eingänge"_____sind. In allen anderen Fällen liefert er _____.
28. Die ODER-Operation ergibt immer dann 1, wenn wenigstens einer der beiden "Eingänge"_____oder beide Eingänge _____sind.
29. Die Exclusiv-ODER-Operation ergibt nur dann 1, wenn beide "Eingänge" einen_____Wert besitzen.

ANTWORTEN, KAPITEL 4

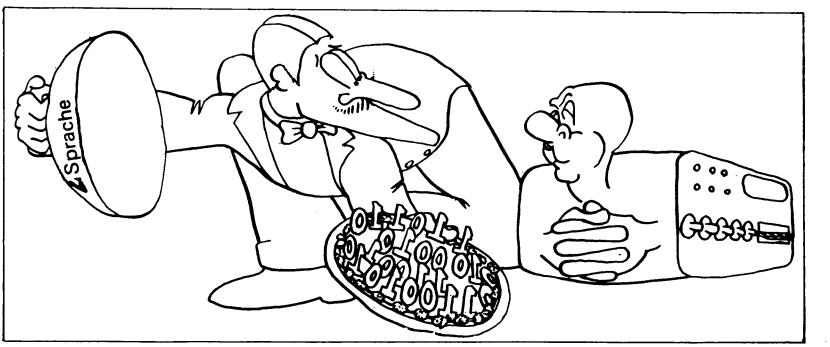
1. 0,1
2. Basis
3. 45_{10}
4. 111111_2
5. Bit
6. Byte
7. 0, 1, 10
8. 10000_2
9. 0,1
10. Zweier-Komplement
11. Einer-Komplement, 1
12. 0101_2
13. 0110_2
14. 011_2
15. 0, 0, 1
16. 1000110_2
17. links, 110_2
18. Division
19. Oktal, Hexadezimal
20. A, B, C, D, E, F
21. $6F_{16}$
22. 0 bis $+255_{10}$, -128_{10} bis $+127_{10}$, Zeichencode
23. ASCII
24. Status-Flags
25. logische Operationen, UND, ODER
26. 1,0, Einer-Komplement
27. 1, 0
28. 1, 1
29. unterschiedlichen

Kapitel 5
DAS INNENLEBEN EINES MIKROCOMPUTERS

Wir haben nun in den Kapiteln 1, 2 und 3 die allgemeinen Mikrocomputer-Konzepte besprochen. In Kapitel 4 betrachteten wir das andere Extrem, indem wir uns mit den Grundlagen befaßten, aus denen jede Computer-Funktion gebildet werden kann. Es ist nun an der Zeit eine Brücke zwischen fundamentalen Konzepten und dem Endprodukt, dem Mikrocomputer-System, zu bilden. Diese Brücke wollen wir in den nächsten beiden Kapiteln bauen. In diesem Kapitel werden wir uns den Mikrocomputer selbst ansehen, seine verschiedenen Komponenten voneinander trennen und erforschen. Kapitel 6 zeigt uns, wie wir die grundlegenden digitalen Logikkonzepte von Kapitel 4 zur Bildung der Komponenten eines Mikrocomputer-Systems, das wir in diesem Kapitel beschrieben haben, verwenden können.
Um die funktionellen Komponenten eines Mikrocomputer-Systems zu überprüfen, wollen wir uns die Art und Weise ansehen, in der ein Mikrocomputer unter Verwendung einer Programmiersprache programmiert werden kann.

PROGRAMMIERSPRACHEN

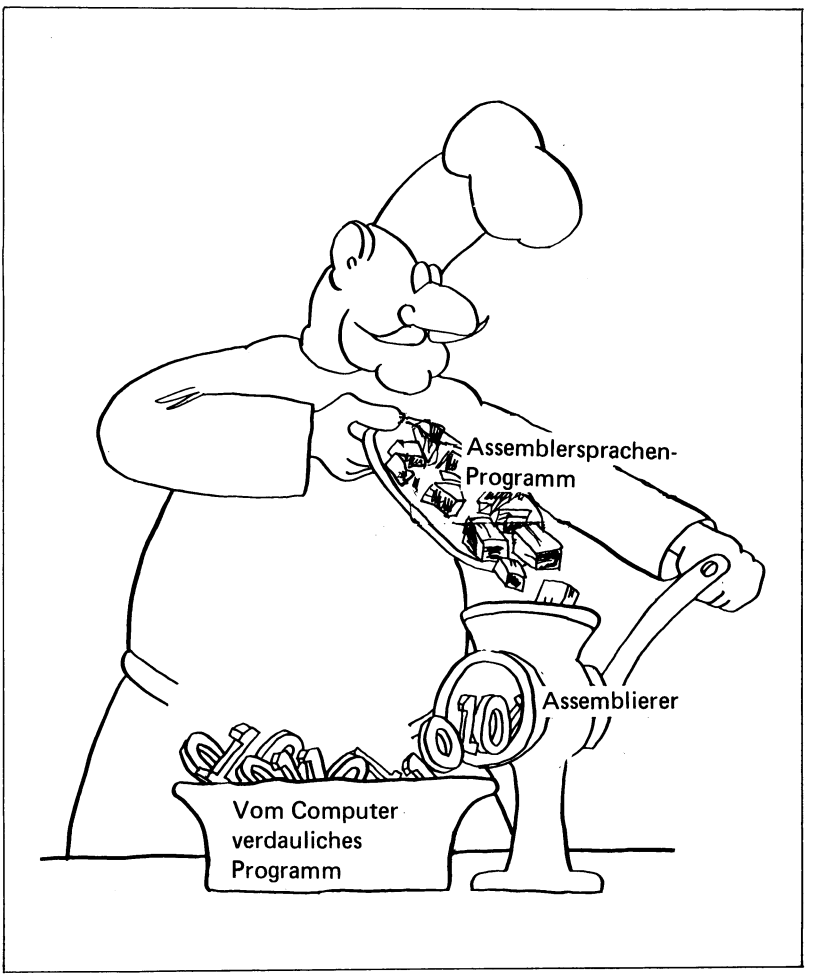
Neben den „höheren" Sprachen wie etwa BASIC, FORTRAN und COBOL gibt es auch „fundamentale" Programmiersprachen, die als „Assemblersprache" bezeichnet werden.
Beim Betrachten jeder Programmiersprache wird einem klar, daß eine Programmiersprache vor allem zur Bequemlichkeit des Programmierers dient. Eine Programmiersprache ist ein künstliches Gebilde, das entwickelt wurde um einem Programmierer das Leben zu erleichtern. **Welche Sprache auch immer wir als die für uns am besten geeignete ausgewählt haben, der Computer verlangt immer, daß er das Programm als eine Folge von Zahlen empfängt:**



Der Computer wird sich nun immer selbst darum kümmern, das Programm von der vom Programmierer geschriebenen Form in die Form umzuwandeln, die er verstehen und ausführen kann. Um diese Umwandlung zu bewerkstelligen, führt der Computer ein anderes Programm aus — ein Programm, das irgend jemand für uns geschrieben hat.

ASSEMBLIERER

Ein Programm, das ein „Assemblierer" genannt wird, wandelt das Programm, das wir in Assemblersprache geschrieben haben, in ein Programm um, das der Computer verstehen und ausführen kann:

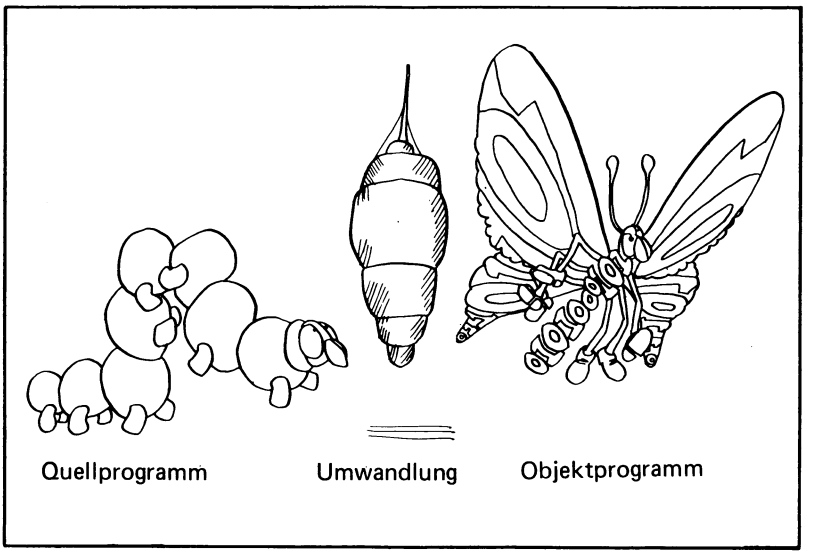


KOMPILIERER

Ein Programm, das als "Kompilierer" bezeichnet wird, führt die gleiche Umwandlung von Programmen aus, die wir jedoch in einer höheren Programmiersprache geschrieben haben.
Assemblierer und Kompilierer behandeln unser Programm als Daten. Sie lesen Daten (unser Programm) und wandeln sie in eine andere Form von Daten (die vom Computer ausführbare Version unseres Programmes) um.

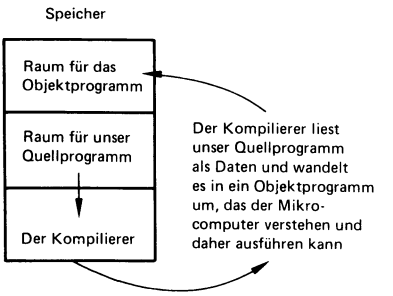
QUELLPROGRAMM
OBJEKTPROGRAMM

Wir bezeichnen ein Programm in einer vom Menschen lesbaren Form als ein "Quell-Programm". Das heißt, ein Quell-Programm ist ein in einer Programmiersprache geschriebenes Programm. Sobald das Programm in eine vom Computer lesbare Form umgewandelt wurde, wird es ein "Objekt-Programm" (oder Maschinen-Programm) genannt. Ein Objekt-Programm ist nichts anderes als eine Folge von Zahlen. Dies kann folgendermaßen gezeigt werden:

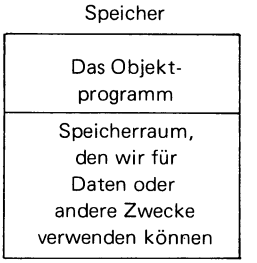


Daher lesen Assemblierer und Kompilierer Daten (unser Quellprogramm) und wandeln diese in eine andere Form von Daten (ein Objekt-Programm) um. In Wirklichkeit gibt es zwei Arten von Kompilierern. Eine Type der Kompilierer nimmt unser Programm, wandelt es in eine vom Computer lesbare Form um und bewahrt diese vom Computer lesbare Form auf. Anschließend wird die vom Computer lesbare Form in den Speicher zur Ausführung geladen. Diese kann folgendermaßen gezeigt werden:

Schritt 1 – Der Kompilier-Schritt

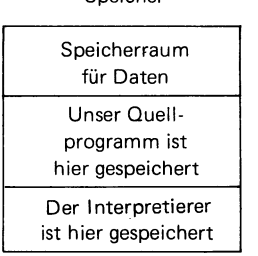


Schritt 2 – Der Ausführungs-Schritt

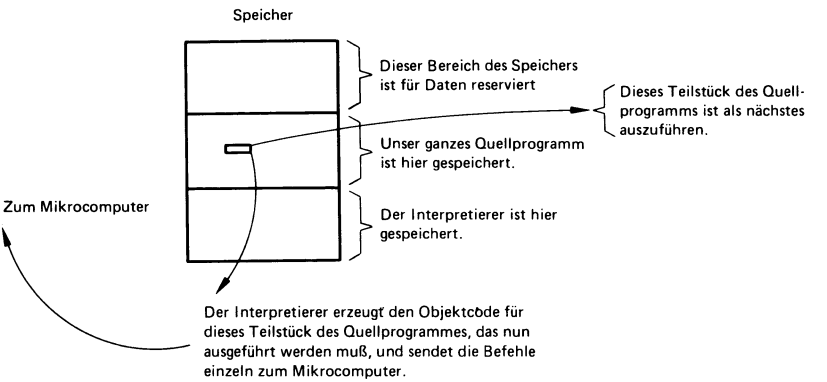


INTERPRETIERER

Eine andere Type von Kompilierern bewahrt niemals die vom Computer lesbare Form unseres Programmes (das heißt das Objektprogramm) auf. Diese Art von Kompilierer wird ein "Interpretierer" genannt. Wenn wir einen Interpretierer verwenden, so liegt unser ganzes Quellprogramm im Speicher, zusammen mit dem Interpretierer, solange wie das Quellprogramm ausgeführt wird. Dies kann folgendermaßen gezeigt werden:



Der Interpretierer wandelt unser Quellprogramm wie verlangt in den Objekt-Code um. Dies kann folgendermaßen dargestellt werden:



Obige Abbildung zeigt, daß ein Bereich des Speichers für unser ganzes Quellprogramm reserviert wurde. Wir könnten vielleicht fälschlicherweise denken, daß der reservierte Teil des Speichers für unser Quellprogramm die Größe des ausführbaren Quellprogramms nach oben begrenzt. In der Tat können wir viel größere Programme ausführen, sobald das größere Programm in Blöcke aufgeteilt werden kann, wenn kein einziger Block den für das Quellprogramm verfügbaren Speicherraum überschreitet.
Kompilierer und Interpretierer sind selbst Objektprogramme, die irgendjemand für uns geschrieben hat.
Wir können den Unterschied zwischen einem Kompilierer und einem Interpretierer in nicht-technischen Ausdrücken erklären, wenn wir an die Art und Weise denken, in der ein Schauspieler seine Rollen in einem Stück lernen könnte. Stellen wir uns das Quellprogramm als das Manuskript des Schauspielers vor. Objektprogramm-Befehle, die zum Mikrocomputer gehen, sind äquivalent mit dem Schauspieler, der seine Rollen einem Auditorium vorträgt. Wenn der Schauspieler seinen ganzen Teil lernt, dann das Manuskript wegwirft und seine Rollen spielt, so hat er das gleiche getan, wie es bei der Kompilierung eines Quellprogrammes geschieht. Aber nehmen wir an, der Schauspieler lernt nicht seinen ganzen Teil. Nehmen wir an, der Schauspieler behält das Manuskript und hat einen Souffleur, der ihm seine Rollen einzeln angibt, unter Verwendung von Soufflier-Tafeln. Er spielt seine Rolle nun in der Art eines Interpretierers.
BASIC ist eine der populärsten höheren Programmiersprachen für Mikrocomputer. Sie ist ebenfalls eine Interpretierer-Sprache.
Zusammengefaßt können wir die meisten Programmiersprachen in "höhere" Sprachen und "Assembler"-Sprachen einteilen. Höhere Sprachen werden in Objektprogramme durch Kompilierer und Interpretierer übersetzt. Assemblersprachen werden in den Objekt-Code durch einen Assemblierer umgewandelt.
Der prinzipielle Unterschied zwischen höheren Sprachen und der Assembler-Sprache ist die Tatsache, daß höhere Sprachen zur Darstellung von Problemen entwickelt

wurden, während Assemblersprachen zur Darstellung des Computers geschaffen wurden. Daher sieht ein Computer ein Quellprogramm in höherer Sprache als etwas sehr Fremdartiges an, und ein Kompilierer hat die umfangreiche Aufgabe, das Quellprogramm in ein Objektprogramm umzuwandeln. Im Gegensatz hierzu kann ein Programm in Assemblersprache sehr leicht in ein Objektprogramm umgewandelt werden. Ein Assemblierer ist daher ein relativ einfaches Programm. Wir wollen nun höhere Sprachen und die Assemblersprache vergleichen, um den Unterschied zwischen diesen beiden klarer herauszuarbeiten.

EIN VERGLEICH ZWISCHEN HÖHEREN SPRACHEN UND DER ASSEMBLERSPRACHE

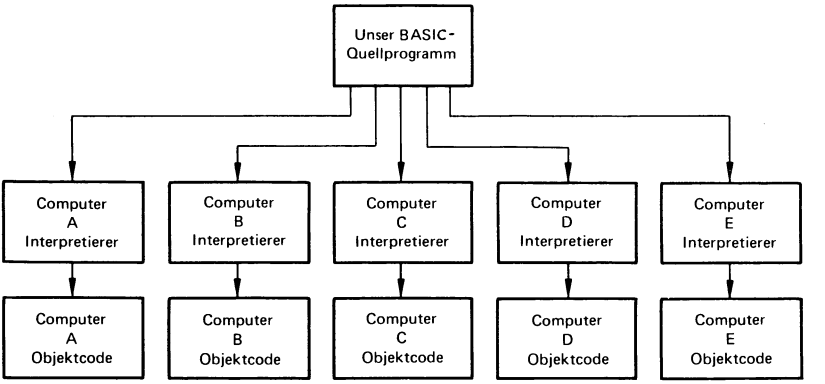
Wir wollen uns zuerst die Vorteile der höheren Sprachen ansehen.
Höhere Sprachen sind einfacher in ihrer Anwendung als Assemblersprachen. Dies liegt darin begründet, daß höhere Sprachen eher das Problem als den Computer darstellen. Beispielsweise würde eine einfache Addition in seiner selbstverständlichen Form unter Verwendung einer höheren Sprache folgendermaßen geschrieben:

```
SUM = VAL1 + VAL2
```

VAL1 und VAL2 sind Bezeichnungen, die wir einem Augenden und einem Addenden zuordnen — die jeden beliebigen Wert haben können. SUM ist die Bezeichnung, die wir der Summe zuordnen.
Die Assemblersprache zeigt uns eine Definition unseres Computers — in einer vom Menschen lesbaren Form. Daher würde die oben gezeigte Addition in Assemblersprache wie folgt dargestellt:

```
LXI   H,VAL1
LDA   VAL2
ADD   A,M
STA   SUM
```

VAL1 und VAL2 sind nicht länger Zeichen, die wir dem Augenden und Addenden zuordnen. VAL1 und VAL2 sind nun Adressen — sie identifizieren Speicherplätze, in denen der Augend und der Addend gespeichert sind. Daher müssen der Augend und der Addend jeder klein genug sein um in einen Speicherplatz zu passen. Ähnlich ist SUM die Adresse des Speicherplatzes, in den die Summe gespeichert wird — vorausgesetzt sie wird in ein Speicherwort passen.
Die Definition der Addition in der Assemblersprache ist keinesfalls selbstverständlich. Ein anderer wesentlicher Vorteil der höheren Sprachen liegt darin, daß sie "problemorientiert" sind. Mit "problemorientiert" meinen wir, daß die Sprache nicht im Hinblick auf irgendeinen bestimmten Computer entwickelt wurde. **Wenn wir daher ein Programm in einer höheren Sprache schreiben, so können wir dieses Quellprogramm in höherer Sprache in ein Objektprogramm umwandeln, das auf jedem Computer laufen wird — vorausgesetzt der Computer hat einen Kompilierer (oder Interpretierer) für höhere Sprachen.** Angenommen wir schreiben beispielsweise ein Programm in BASIC. Wir können dieses BASIC-Programm auf unserem Computer ausführen. Und ebenso können alle unsere Freunde unser Programm auf ihren vollkommen unterschiedlichen Computern laufen lassen — vorausgesetzt ihre Computer haben ebenfalls BASIC-Interpretierer. Dies kann folgendermaßen dargestellt werden:



Assemblersprache ist andererseits eine menschliche Darstellung des von uns verwendeten Computers. Daher hat jeder einzelne Computer und Mikroprozessor seine eigene eindeutige Assemblersprache. Und ein Programm, das in der Assemblersprache eines Computers oder Mikroprozessors geschrieben wurde, ist völlig ungeeignet für jeden anderen Computer oder Mikroprozessor. Wenn wir ein Quellprogramm in Assemblersprache für unseren Mikroprozessor schreiben, so werden nur Leute mit einem Mikrocomputer, der unseren Mikroprozessor enthält, imstande sein unser Quellprogramm zu assemblieren und ablaufen zu lassen.

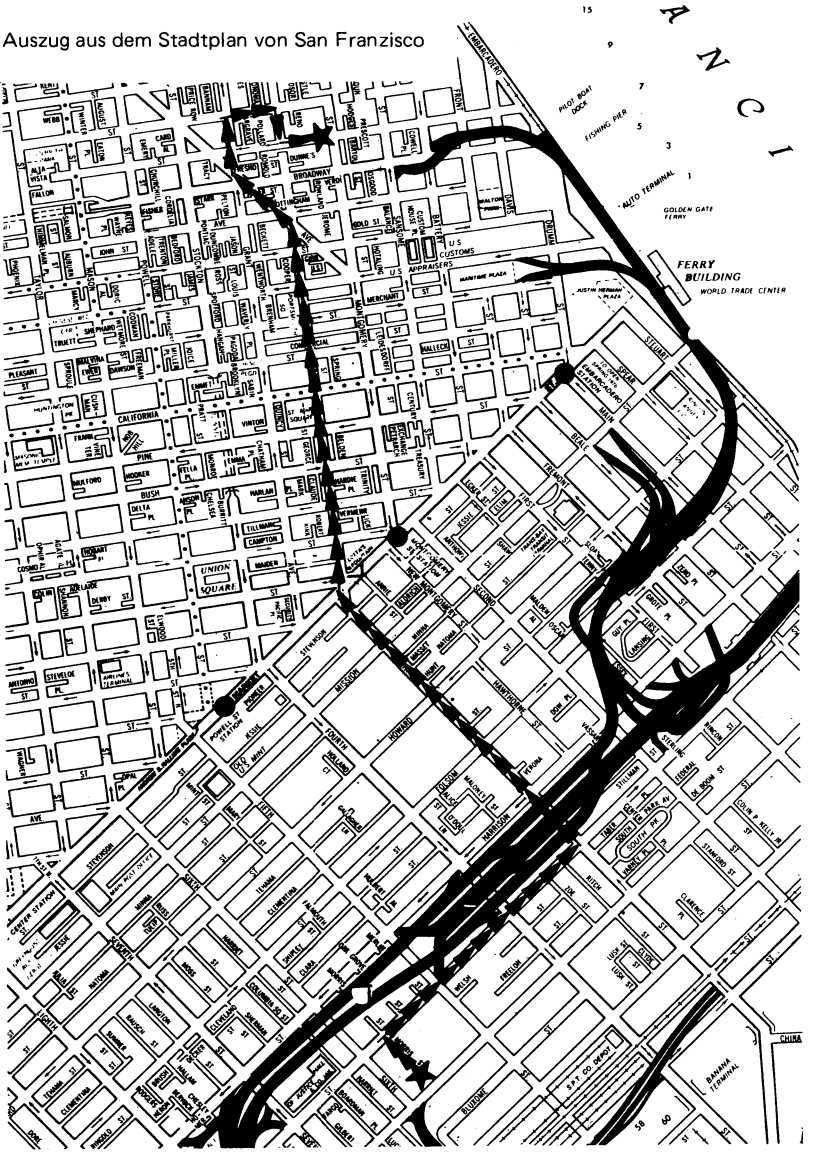
Theoretisch wäre es möglich, ein Programm sehr ähnlich einem Kompilierer zu schreiben, das ein Quellprogramm, das in der Assemblersprache eines bestimmten Mikroprozessors geschrieben wurde, aufnimmt und es in ein Objektprogramm für einen anderen Mikroprozessor umwandelt. In Wirklichkeit machen dies sehr wenige Leute, da mit der Assemblersprache eines anderen Mikroprozessors ebenso fremd und schwer umzugehen ist wie mit einer höheren Sprache.

Warum plagt sich dann überhaupt jemand mit der Assemblersprache, trotz all der Vorteile, die die Programmierung in einer höheren Sprache bietet? Assemblersprache bietet auch Vorteile.

Die Assemblersprache erzeugt vor allem wesentlich kürzere Objektprogramme als höhere Sprachen. Der Grund hierfür ist, daß die Assemblersprache für jeden Mikroprozessor oder Computer speziell für diesen Mikroprozessor oder Computer entwickelt wurde. Tatsächlich ist ein von einem Kompilierer geschaffenes Objektprogramm aus einem Quellprogramm in höherer Programmiersprache gewöhnlich zwei- bis viermal so lang wie das gleiche Objektprogramm, das durch einen Assemblierer aus einem Quellprogramm in Assemblersprache gebildet wurde. Der Grund hierfür ist, daß der Kompilierer in Wirklichkeit ein Assemblersprachen-Programm zur Darstellung des Problems schreiben muß, wie es in der höheren Sprache definiert wurde. Aber wo ein menschlicher Programmierer ein Programm schreiben kann und hierbei menschliche Entscheidungen verwendet, muß der Kompilierer diese Aufgabe mit festen Regeln erledigen.

Betrachten wir einen Fall aus dem täglichen Leben. Wir müssen jemandem zeigen, wie er von einem Punkt in einer Stadt zu einem anderen gelangt. Wenn wir den genauen Ausgangsort und Bestimmungsort wissen, dann können wir eine äußerst direkte Route definieren:

Auszug aus dem Stadtplan von San Franzisco



Nun versuchen wir einen Satz universeller Befehle zu schaffen, die wir aneinanderhängen können um die Route zu definieren, die zwischen zwei Punkten einer Stadt zu fahren ist. Wenn diese Befehle durch eine Maschine zu interpretieren sind, kann nichts der Fantasie überlassen werden. Es muß daher eine bestimmte Anzahl von Befehlen wie etwa folgende vorhanden sein:

Biege nach links ab
Biege nach rechts ab
Prüfe ob eine Einbahnstraße
Prüfe ob eine Sackgasse
Prüfe für eine Wendung um 45°
etc.

Wir können keine Befehle einschließen, bei denen angenommen wird, daß wir wissen ob eine Einbahnstraße vorliegt oder nicht, da Einbahnstraßen häufig geändert werden. Wir können auch keine Befehle einschließen, die einfach die Anzahl der Häuserblocks definieren, an denen wir gerade vorbeifahren müssen, da derartige Straßen gesperrt sein könnten. Oder in Städten mit sehr steilen Hügeln wie etwa San Francisco kann eine Straße, die fortlaufend aussieht, in Wirklichkeit durch 30 Meter tiefe Abgründe in irgendeinem Punkt geteilt sein.
Sobald wir beginnen, uns einen Satz universeller Richtungsanweisungen auszudenken, die unbestimmte Möglichkeiten berücksichtigen, haben wir eine Vorstellung des Problems, das bei einem Kompilierer auftritt. Der Kompilierer weiß nicht, welche Besonderheit bei irgendeinem speziellen Computer vorliegt, er muß daher Programme erzeugen, die auch die ausgefallendsten Möglichkeiten berücksichtigen.
Bei höheren Programmiersprachen gibt es ein weiteres Problem. Der Kompilierer, der ein Quellprogramm in höherer Sprache in ein Objektprogramm umsetzt ist selbst ein großes Programm. Ein Kompilierer-Programm kann achtmal so lang wie ein Assemblerprogramm sein. **Daher kann unser Mikrocomputer-System eine höhere Sprache nur dann verwenden, wenn unser Mikrocomputer-System auch ausreichenden Speicherraum zum Aufbewahren des Kompilierers besitzt.**
Wenn wir einen Interpretierer haben, dann muß dieser Interpretierer immer im Speicher liegen, zusammen mit dem auszuführenden Programm. Dieser Unterschied zwischen einem Kompilierer und einem Interpretierer wurde bereits früher in diesem Kapitel gezeigt.
Die Tatsache, daß Quellprogramme in einer höheren Sprache längere Objektprogramme erzeugen, bedeutet auch, daß das Objektprogramm mehr Zeit für die Ausführung benötigt, da hier mehr Befehle abzuarbeiten sind. Wenn bei unserer Anwendung bestimmte Anforderungen an die Geschwindigkeit vorliegen, können wir eine Beschleunigung um den Faktor 2 oder mehr erreichen, indem wir einfach unser Programm neu in Assemblersprache schreiben.
Auch einige der Vorteile von höheren Programmiersprachen sind nur scheinbar. **Beispielsweise sollen höhere Sprachen auch übertragbar sein.** Mit anderen Worten, ein Programm in höherer Sprache kann kompiliert und von vielen verschiedenen Mikroprozessoren ausgeführt werden. Das ist jedoch nicht immer richtig. **Häufig werden wir finden, daß kleine Unterschiede in der Art und Weise bestehen, in der der Kompilierer eines Computers das Aussehen des Quellprogrammes erwartet, verglichen mit dem nächsten.** Jedoch auch im ungünstigsten Fall würden die Änderungen, die an dem Quellprogramm in der höheren Sprache zu machen sind, wenn man zu einem neuen

Mikroprozessor oder Computer übergeht, geringfügig sein, verglichen mit den Problemen, die mit einem vollkommenen Neuschreiben des Programms in der neuen Assemblersprache des Mikroprozessors oder Computers verbunden sind.

Was können wir also daraus folgern?

Wenn wir die Absicht haben einen Mikrocomputer nur als Hilfsmittel für die Ausführung von Programmen zu verwenden, so sollten wir so rasch wie möglich eine höhere Programmiersprache verwenden. Wenn wir jedoch andererseits vorhaben, tiefer in die Mikrocomputer-Technik einzudringen, einen eigenen Mikrocomputer zu bauen, ihn abzuändern, oder uns sonst irgendwie mit seinen Komponenten zu befassen, dann sollten wir so rasch wie möglich die Assemblersprache lernen, und wir werden wahrscheinlich auch bei der Assemblersprache bleiben.

DIE FUNKTIONSLOGIK DES MIKROCOMPUTERS

Das von uns geschaffene Objektprogramm bestimmt die Funktionen, die von der Logik unseres Mikrocomputers ausgeführt werden.

Bild 5.1 zeigt die Funktionslogik eines Mikrocomputers. Diese Logik wollen wir nun besprechen.

Egal was der Mikrocomputer auch tut – letztlich besteht die Aufgabe aus diesen drei Schritten:

1) **Einbringen der Daten in den Mikrocomputer.**
2) **Modifizierung (oder Verarbeitung) der Daten.**
3) **Zurücksenden der modifizierten Daten aus dem Mikrocomputer.**

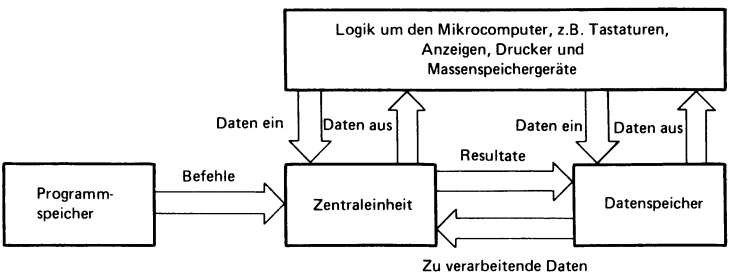
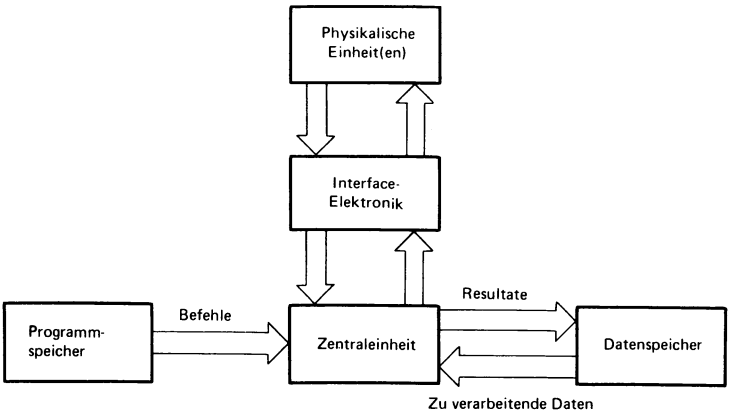


Bild 5.1 Mikrocomputer-Funktionslogik

Die Logik um den Mikrocomputer (die aus physikalischen Einheiten besteht, die wir früher in diesem Buch beschrieben haben) wird zur Eingabe der Informationen, zum Empfang der Resultate und zur Speicherung größerer Datenmengen verwendet. Die Daten, die gerade verarbeitet werden, sind im Datenspeicher aufbewahrt, der, wie wir uns aus Kapitel 2 erinnern, ein Schreib-/Lesespeicher mit schnellem Zugriff ist. **Daher werden die oben gezeigten Schritte 1) und 3) von der in Bild 5.2 gezeigten schraffierten Mikrocomputerlogik gehandhabt.**

Physikalische Einheiten, wie wir uns erinnern, senden Informationen zu und vom Mikrocomputer über entsprechende Interface-Logik. Mit Bezug auf Bild 5.1 kann dies folgendermaßen gezeigt werden:



ZENTRAL-EINHEIT

Die tatsächlich ausgeführten Operationen an Daten werden durch die Logik innerhalb der Zentraleinheit (CPU = Central Processing Unit) erledigt. Diese Operationen werden durch eine Folge von Befehlen definiert, die zusammengenommen das Programm darstellen. Das Programm wird im Programmspeicher aufbewahrt. **Daher wird Schritt 2) der obigen 3 Schritte von der in Bild 5.3 schraffierten Mikrocomputer-Logik ausgeführt.**

PROGRAMM-SPEICHER

Der Programmspeicher kann ein Festwertspeicher oder ein Schreib-/Lesespeicher sein. Der Programmspeicher kann ein Festwertspeicher sein, da die vom Programm im Programmspeicher aufbewahrten Befehle zur Zentraleinheit gesendet werden. Es werden jedoch gewöhnlich Befehle nicht von der Zentraleinheit zum Programmspeicher übertragen. Programmspeicher müssen nicht unbedingt Festwertspeicher sein. Es ist in Mikrocomputer-Systemen allgemein üblich, Programme von Daten zu trennen, wie in Bild 5.1 gezeigt wird, und in vielen industriellen Mikrocomputer-Anwendungen werden Programme in Festwertspeichern aufbewahrt um zu sichern, daß das Programm niemals zufällig geändert wird oder verlorengeht.

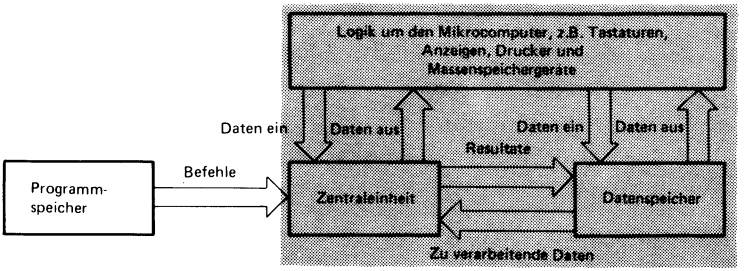


Bild 5.2 Mikrocomputer-Funktionslogik, die bei der Bewegung und Speicherung von Daten beteiligt ist

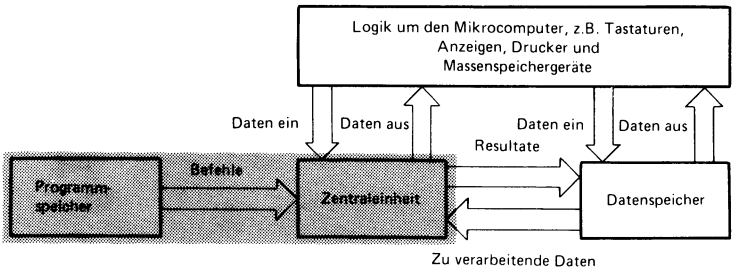
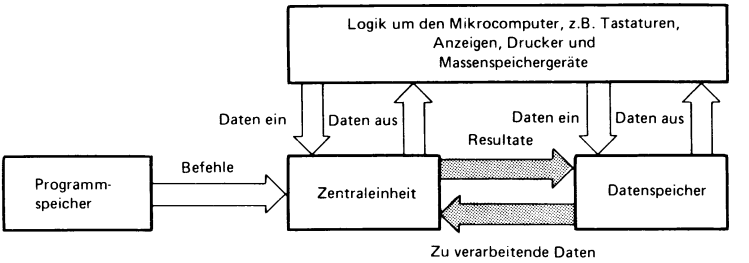


Bild 5.3 Mikrocomputer-Funktionslogik, die bei der Modifikation von Daten beteiligt ist

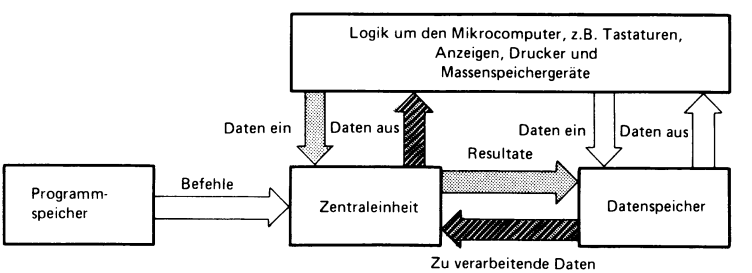
Es können jedoch Programmspeicher und Datenspeicher ein und derselbe Speicher sein. Darüber hinaus ist es möglich, daß ein Teil eines Programmes einen anderen Teil des Programmes als Daten behandelt, wobei sich das Programm selbst ändert. Wie man erwarten kann, werden sich selbst ändernde Programme sehr komplex werden. Für einen Anfänger ist es daher zweckmäßig, sich den Programmspeicher und den Datenspeicher als getrennte und unterschiedliche Einheiten zu denken.
Die Tatsache, daß wir bis jetzt noch nicht genauer verstehen, wie Programm- und Datenspeicher arbeiten, ist unwichtig. Abbildungen von Programm- und Datenspeicher-Chips sind in Kapitel 1 und 2 zu sehen. Diese Chips können Informationen in einer vom Computer lesbaren Form speichern. Im Moment ist dies alles was wir über Programm- und Datenspeicher wissen müssen.

INFORMATIONSWEGE

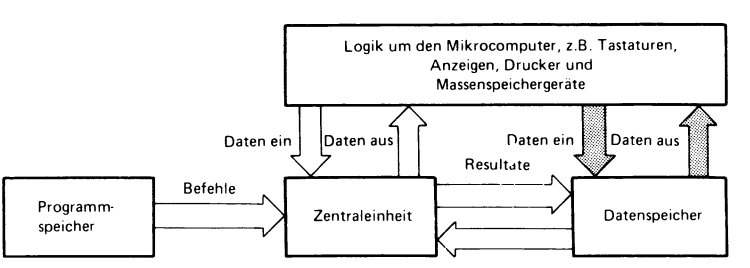
Wir wollen uns nun die verschiedenen Informationswege, wie sie in Bild 5.1 gezeigt werden, ansehen.
Wenn die Zentraleinheit Daten modifiziert, so holt sie gewöhnlich die zu modifizierenden Daten aus dem Datenspeicher und bringt die Ergebnisse meistens in den Datenspeicher zurück. **Deshalb gibt es Wege (Pfade) in beiden Richtungen zwischen dem Datenspeicher und der Zentralheinheit:**



Neue in den Mikrocomputer eingegebene Daten bewegen sich von den externen physikalischen Einheiten zum Datenspeicher über die Zentraleinheit. Auszugebende Resultate bewegen sich vom Speicher über die Zentraleinheit zu den externen physikalischen Einheiten. Dies kann folgendermaßen gezeigt werden.:

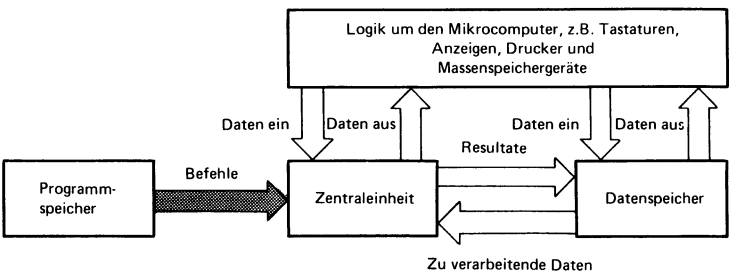


Ein sehr schneller Datentransfer zwischen Disketten und dem Datenspeicher tritt häufig direkt zwischen diesen beiden Geräten auf, indem sie an der CPU (Zentraleinheit) vorbeigehen:



DIREKTER SPEICHERZUGRIFF
DMA

Der auf Seite 5 - 13 gezeigte Datenweg wird als direkter Speicherzugriff (Direct Memory Access = DMA) bezeichnet.
Der direkte Speicherzugriff wird häufig nur mit seinen Anfangsbuchstaben verwendet: DMA. Während an einem Ende eines DMA-Datentransfers immer ein Speicher liegen muß, muß am anderen Ende nicht unbedingt eine Diskette vorhanden sein, obwohl dies sehr häufig der Fall ist. Jede externe Logik kann am anderen Ende des DMA-Datentransfers liegen.
Wenn immer die Zentraleinheit irgend etwas tut – Bewegen von Daten oder Modifizierung von Daten – wird ein Strom von Befehlen vom Programmspeicher zur Zentraleinheit gesendet und steuert die Operationen der Zentraleinheit. **Daher muß ein einseitig gerichteter (unidirektionaler) Weg für den Fluß von Informationen vom Programmspeicher zur Zentraleinheit vorhanden sein:**



DIE ZENTRALEINHEIT

Im Mittelpunkt aller Mikrocomputer-Logik steht die Zentraleinheit. Die Zentraleinheit ist die elektronische Logik, die tatsächlich alle Operationen an Daten ausführt. Mit anderen Worten, die verschiedenen anderen Teile des Mikrocomputer-Systems können Daten von einer Stelle zur anderen bewegen, jedoch nur innerhalb der Zentraleinheit können Daten tatsächlich verändert werden.

CPU

Die Zentraleinheit wird gewöhnlich mit ihren Anfangsbuchstaben „CPU" (Central Processing Unit) bezeichnet.

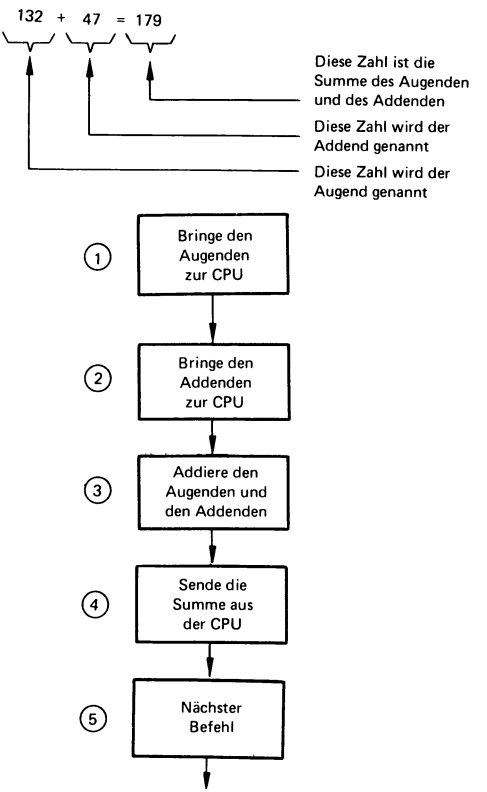
SERIELLE LOGIK

Um die Vielseitigkeit und Leistungsfähigkeit zu gewährleisten, die normalerweise mit Computern verbunden ist, muß die Logik der Zentraleinheit imstande sein, eine große

Anzahl verschiedener Operationen auszuführen. Und das ist es in der Tat was die Zentraleinheit kann. Die Zentraleinheit kann jedoch nur eine Operation zur gleichen Zeit ausführen.

AUGEND ADDEND

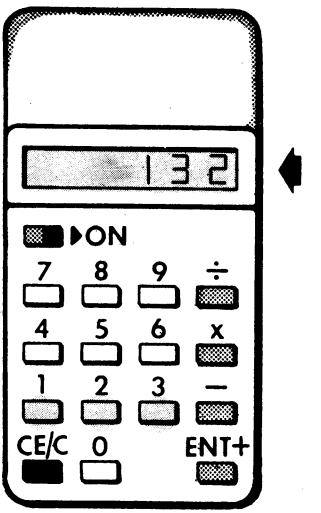
Betrachten wir die Addition von zwei Zahlen. Wenn zwei Zahlen addiert werden, so werden sie der Augend und der Addend genannt. Der Augend und der Addend werden über folgende serielle Folge von Vorgängen summiert:



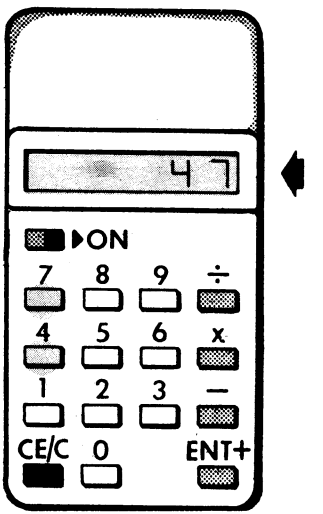
Jeder Vorgang ist durch eine Zahl ① , ② , ③ , ④ etc. identifiziert. Die CPU führt jeden Vorgang als einzelne Operation aus. **Um daher die oben gezeigte Addition zu bewerkstelligen, führt die CPU den Vorgang ① , dann den Vorgang ② , dann Vorgang ③ , dann Vorgang ④ aus.**

Während des ersten Schrittes wird der Augend zur CPU gebracht.
Während des zweiten Schrittes wird der Addend zur CPU gebracht.
Während des dritten Schrittes werden der Augend und der Addend durch die elektronische Logik innerhalb der CPU summiert.
Während des vierten Schrittes wird die Summe aus der CPU gesendet.
Diese vier Schritte sind im wesentlichen identisch mit den vier Schritten, mit denen wir zwei Zahlen unter Verwendung der meisten älteren Taschenrechner addieren.

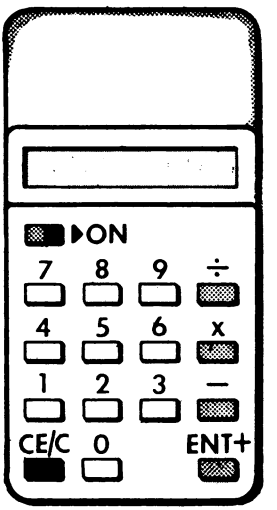
Während des Schrittes 1 tasten wir den Augenden ein:



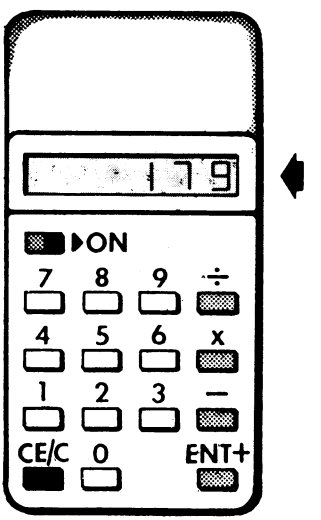
Während des Schrittes 2 tasten wir den Addenden ein:



Während des Schrittes 3 drücken wir die +-Taste:



Schritt 4 tritt automatisch auf: Die Summe wird von der Logik des Rechners ausgegeben, und ist auf der Anzeige abzulesen:



Wir wissen nun, warum wir bei diesen Rechnern manche Dinge etwas unhandlich ausführen müssen. Sie zwingen uns Computer-Logiksequenzen zu verwenden.
Dies verringert die Kosten und die Komplexität des Rechners.
Neuere Rechner verwenden eine komplexere Logik, die uns in einer bekannteren Sequenz arbeiten lassen:
Während Schritt 1 tasten wir den Augenden ein.
Während Schritt 2 drücken wir die +-Taste.
Während Schritt 3 tasten wir den Addenden ein.
Schritt 4 erfolgt automatisch: Die Summe wird ausgegeben.

SERIELLE BAUSTEINE

Wir können die vier Schritte des Taschenrechners (jeder Version), mit denen wir zwei Zahlen addieren, zur Erläuterung des Konzepts eines seriellen Bausteins verwenden, da ein Taschenrechner und eine Zentraleinheit beide serielle Bausteine sind. Jeder kann nur eine Operation zur gleichen Zeit ausführen. Dies ist im Falle des Taschenrechners sehr einfach zu verstehen. Wir können beispielsweise nicht gleichzeitig beide zu addierenden Zahlen eintasten. Die zwei Zahlen müssen seriell, das heißt eine nach der anderen eingetastet werden. Im Falle der Zentraleinheit können wir nicht gleichzeitig den Augenden und den Addenden in die Zentraleinheit einbringen. Jede Zahl muß über einen unabhängigen Schritt geholt werden, und die beiden Schritte müssen einer nach dem anderen auftreten.

SERIELLER LOGIKSCHRITT

BEFEHLS-SCHRITT

Als nächstes müssen wir untersuchen, worin ein einzelner "Schritt" eigentlich besteht. Im Falle des Taschenrechners ist dies keine sehr wichtige Überlegung. Wenn wir die Zahl 132 über Tasten eingeben, besteht dann die Eingabe der ganzen Zahl aus einem "Schritt"? Oder stellt jeder Tastenanschlag einen individuellen "Schritt" dar? Offen gesagt ist diese Frage für einen Taschenrechner belanglos. Aber wie ist es, wenn wir eine Folge von Befehlen niederzuschreiben haben, die irgendjemand befolgen muß? Wir könnten den folgenden Einzelschritt niederschreiben:
1) Gib 132 an der Tastatur ein.
Wir könnten diesen Einzelschritt in drei getrennte Schritte aufteilen:
1) Drücke die 1-Taste
2) Drücke die 3-Taste
3) Drücke die 2-Taste
Betrachten wir ein wesentlich banaleres Beispiel: Das Essen eines Stückes Kuchen.
Nehmen wir an, ein Stück eines Kuchens könne mit zehn Bissen verspeist werden. Ist daher das Essen eines Stückes Kuchen ein Vorgang mit zehn Schritten? Vielleicht, vielleicht auch nicht. Das Verarbeiten eines einzigen Bissens eines Kuchens kann selbst aus folgenden vier Schritten bestehen:
1) Abtrennen eines Stückes des Kuchens mit unserer Gabel.

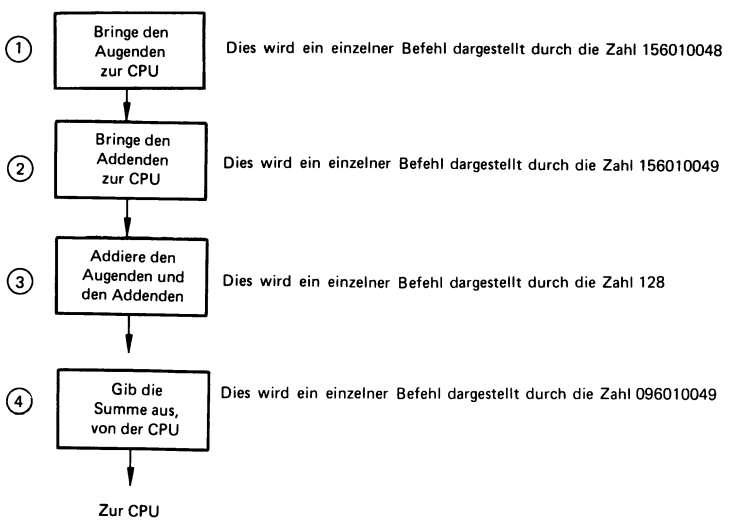
2) Aufspießen des abgeteilten Stücks des Kuchens mit dem Ende unserer Gabel.
3) Transportieren des abgeteilten Stückes des Kuchens zu unserem Mund.
4) Kauen und Schlucken des Kuchenstückes.

BEFEHLE

Es wär sehr leicht, diese vier Schritte beim Essen eines Kuchens noch weiter zu unterteilen, indem wir jede beliebige Anzahl zusätzlicher kleinerer Schritte bilden. Das gleiche gilt für einen einzelnen Schritt der Zentraleinheit. Manche Zentraleinheiten führen Operationen in relativ großen Stufen aus. Andere bilden eine Folge von Vorgängen in relativ kleinen Schritten. **Jedoch für jede Zentraleinheit ist jeder Schritt klar und unmißverständlich als ein "Befehl" definiert.** Ein individueller Befehl oder Schritt, der von jeder Zentraleinheit ausgeführt wird, ist in keiner Weise unbestimmt oder undeutlich.

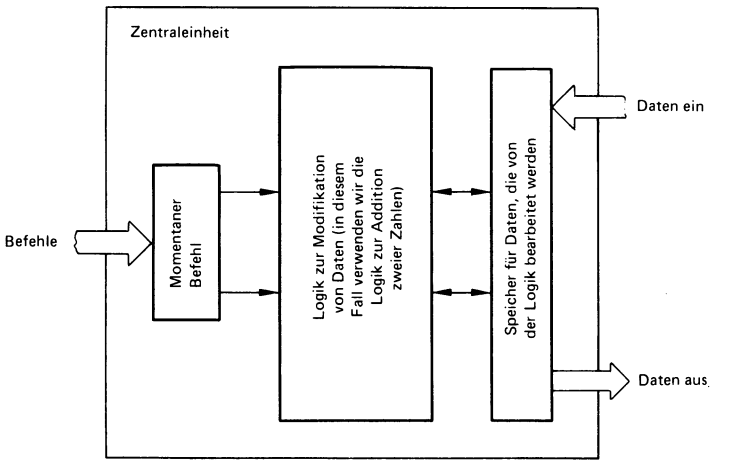
BEFEHLS-VORRAT

Jede Zentraleinheit reagiert auf eine feste Anzahl von Befehlen. Diese Befehle werden zusammengefaßt als Befehlsvorrat oder Befehlssatz bezeichnet. Eine Zentraleinheit besitzt typisch zwischen 40 und 200 verschiedene Befehle in ihrem Befehlssatz.
Jeder Befehl wird durch eine eindeutige Zahl dargestellt, die bei der Übertragung an die Zentraleinheit zur richtigen Zeit die Zentraleinheit zur Ausführung der dem Befehl zugeordneten Operationen veranlaßt. Unsere Additionsfolge könnte folgendermaßen dargestellt werden:



DATENSPEICHERUNG IN DER ZENTRALEINHEIT

Die vier auf Seite 5-19 dargestellten Befehle zeigen ein mit der CPU verbundenes logistisches Problem.
Die CPU hat so viel Speicherraum, daß sie die Daten, die sie gerade bearbeitet, aufbewahren kann, und das ist alles. Dies kann folgendermaßen gezeigt werden:

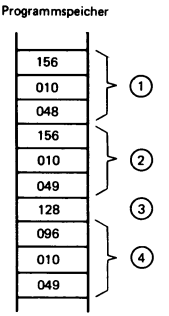


Wir können nicht erwarten, daß wir den Augenden, Addenden und die Summe im Speicherraum der Zentraleinheit belassen, da dieser Platz mit Sicherheit für die unmittelbar danach von der Zentraleinheit auszuführende Operation benötigt wird. Der Augend, der Addend und die Summe muß daher einen ständigen Speicherplatz irgendwo außerhalb der Zentraleinheit besitzen — zum Beispiel in einem externen Schreib-/Lesespeicher. **Aus diesem Grund existieren die Schritte ① ② und ④**

PROGRAMMSPEICHER

PROGRAMM

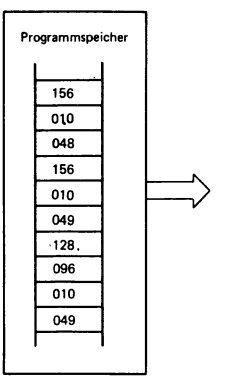
Um jede Operation wie die gezeigte Addition auszuführen, müssen wir eine Folge von Operationen schaffen, die zusammengenommen das Programm bilden. Das Programm ist eine Folge von Zahlen. Diese Zahlenfolge wird in einem Speicher mit schnellem Zugriff aufbewahrt, den wir Programmspeicher nennen. Unter Verwendung willkürlich zugeordneter Zahlencodes für die Additionsbefehle, **könnte das Additionsprogramm folgendermaßen grundsätzlich dargestellt werden:**



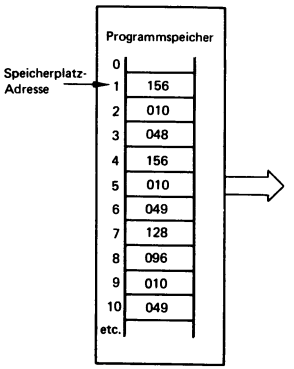
Die oben verwendete Methode zur Darstellung des Speicherinhalts werden wir häufig in diesem Buch sehen, sowie in anderen Büchern dieser Serie. Man könnte diese Darstellung mit einem Stapel übereinandergeordneter Fächer vergleichen, von denen jedes Fach eindeutig identifizierbar und adressierbar ist.
Wann immer eine Zahl von der CPU zum Speicher transferiert wird, wird ein „Fach" belegt. Wird eine Zahl vom Speicher zur CPU transferiert, so empfängt die CPU den Inhalt eines „Faches".

SPEICHERPLÄTZE UND ADRESSEN

Jedes "Fach" wird ein "Speicherplatz" genannt. Jeder Speicherplatz ist individuell über eine eindeutige Speicheradresse identifizierbar.
Wir gehen nun daran, die Speicheradresse zu schaffen, die einen individuell adressierbaren Platz innerhalb des Speichers identifiziert. Daher wird die Befehlsfolge des oben gezeigten Additionsprogramms durch Belegen einer unbestimmten Folge von Programmspeicherplätzen wie folgt dargestellt:



Ohne überhaupt an eine Speicheradressierung zu denken, könnten wir die Befehlsfolge des Additions-Programmes in den ersten zehn adressierbaren Plätzen des Programmspeichers wie folgt aufscheinen lassen:



Wir brauchen keine weitere Computerlogik zu verstehen um zu sehen, wie die ersten zehn adressierbaren Plätze des Programmspeichers mit Zahlen, wie oben gezeigt, gefüllt werden können. Man muß noch etwas mehr von der Computerlogik wissen um zu erklären, wie wir jeden von diesen oder irgend einen anderen Speicherplatz identifizieren. Dieses Thema wird in Kapitel 6 besprochen.

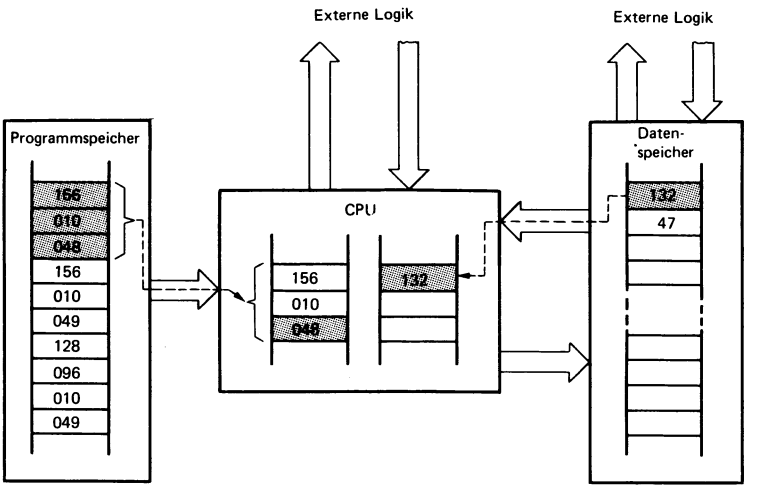
DATENSPEICHER

Die Informationen, die von einem Programm während seiner Ausführung verwendet werden können, werden als Daten bezeichnet. **In unserem einfachen Additionsbeispiel sind wir daran gegangen, drei verschiedene Daten zu handhaben: Den Augend und den Addend, die zu addieren waren, und die Summe. Diese drei verschiedenen Daten werden wir wahrscheinlich in einem Datenspeicher mit schnellem Zugriff aufbewahren.**

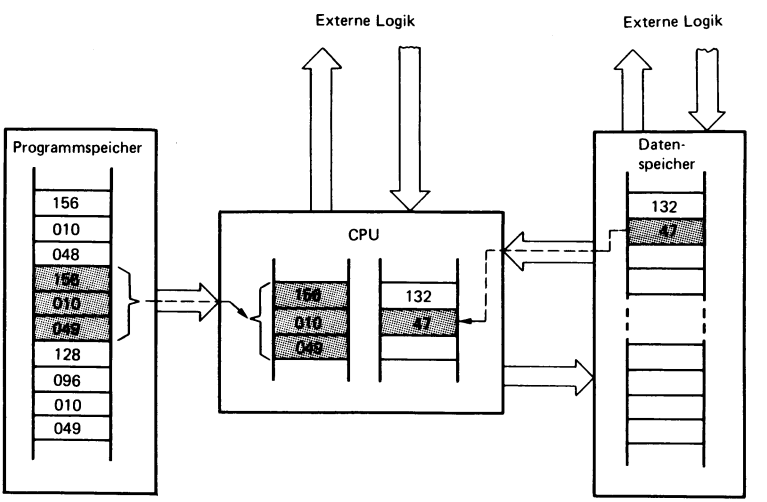
FOLGE DER EREIGNISSE BEI EINEM ADDITIONSPROGRAMM

Der Vorgang des Addierens zweier Zahlen kann nun grundsätzlich wie folgt dargestellt werden:

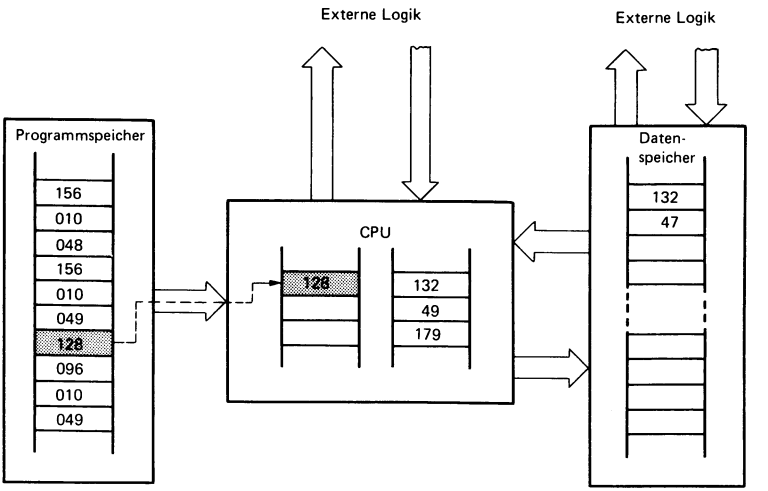
Schritt 1: Holen des Augenden



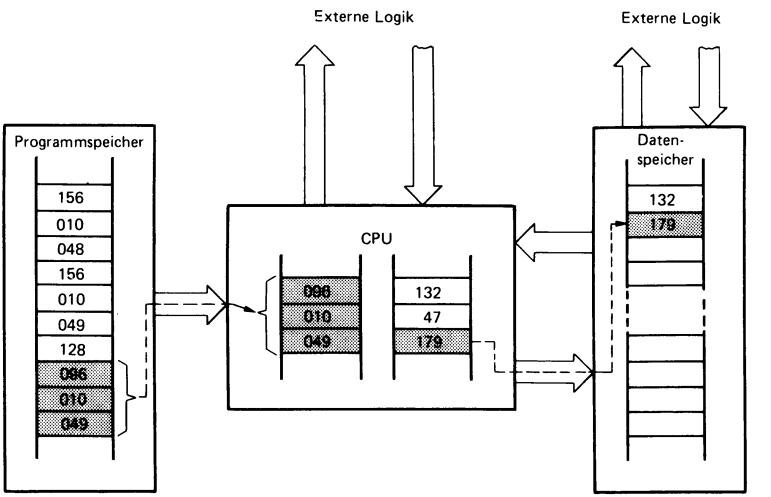
Schritt 2: Holen des Addenden



Schritt 3: Bilden der Summe



Schritt 4: Ausgabe der Summe



Für jeden der vier gezeigten Schritte wird der erste auftretende Vorgang der Transfer eines Befehls-Codes vom Programmspeicher zur CPU sein. In jedem Schritt ist der Befehlscode die schraffierte Zahl im Programmspeicher. Die CPU kann nicht wissen was sie tun muß, bis der Befehlscode bei ihr eingetroffen ist. Sobald der Befehlscode die CPU erreicht hat, laufen die von diesem Schritt erforderlichen Operationen tatsächlich ab. Die Operationen sind augenscheinlich. Wir sehen, daß in Schritt 4 die Summe willkürlich in den gleichen Daten-Speicherplatz zurückgeschrieben wird, von dem der Addend geholt wurde. Daher geht der Addend verloren.

KAPITEL 5

FRAGEN:

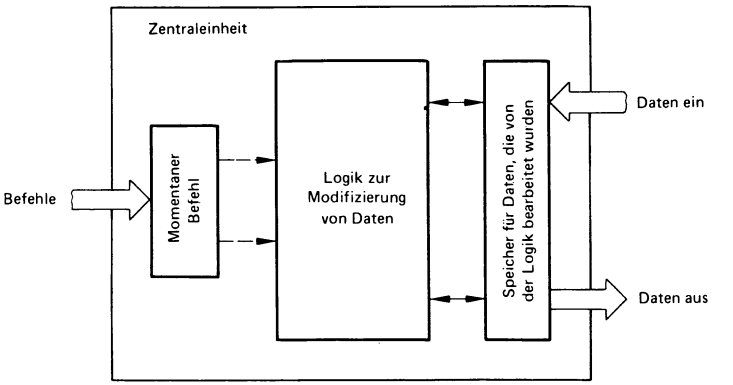
1. Unter höheren Programmiersprachen versteht man Sprachen wie _____, FORTRAN und COBOL.
2. Fundamentale Programmiersprachen bezeichnet man als _____-Sprachen.
3. Mit einem _____ wandelt man ein Programm in Assemblersprache in ein Programm um, das der Computer verstehen kann.
4. Ein _____ erzeugt aus einem Programm in einer höheren Programmiersprache ein dem Computer verständliches Programm.
5. Das ursprüngliche von einem Menschen geschriebene und von ihm lesbare Programm nennt man ein _____.
6. Wenn dieses Programm von einem Assemblierer oder Kompilierer in ein für den Computer lesbares Programm umgewandelt wurde, so spricht man von einem _____ _____-Programm oder _____-Programm.
7. Bei einem _____ liegt das Programm für die Umwandlung ständig im Speicher des Mikrocomputers und übersetzt das Quellprogramm laufend während seiner Eingabe.
8. Höhere Programmiersprachen sind meist _____ -orientiert, während Assemblersprachen immer _____-orientiert sind.
9. Der Vorteil der höheren Programmiersprachen liegt darin, daß diese _____ zu erlernen sind, jedoch meistens _____ Objekt-Programme ergeben, als maschinen-orientierte Programmiersprachen.
10. Alle an Daten ausgeführten Operationen werden in einem Mikrocomputer innerhalb der Logik der _____, dem eigentlichen _____ ausgeführt.
11. Das Programm für den Mikrocomputer wird meist in einem Festwertspeicher, dem sogenannten _____-Speicher aufbewahrt.
12. Daten werden dagegen im _____-Speicher aufbewahrt.
13. Für einen schnellen Datentransfer zwischen externen Geräten, wie beispielsweise Disketten und dem Datenspeicher, umgeht man häufig die Zentraleinheit und nennt das einen _____ oder _____.
14. Die Zentraleinheit, oder der Mikrocomputer kann immer nur _____ Operation zur gleichen Zeit ausführen. Man nennt dies eine _____ Logik.
15. Der serielle Ablauf von Vorgängen in einem Mikrocomputer erfolgt durch eine Folge von _____.
16. Der jeweilige Satz von Befehlen, über die ein Mikrocomputer verfügen kann, stellt seinen _____ oder _____ dar.
17. Jeder Platz in einem Speicher wird durch seine _____ eindeutig festgelegt. Dies gilt sowohl für Programm- wie Datenspeicher.

ANTWORTEN, KAPITEL 5

1. BASIC
2. Assembler
3. Assemblierer
4. Kompilierer
5. Quell-Programm
6. Objekt, Maschinen
7. Interpretierer
8. problem, maschinen
9. leichter, längere
10. Zentraleinheit, Mikroprozessor
11. Programm
12. Daten
13. direkter Speicherzugriff, DMA
14. eine, serielle
15. Befehlen
16. Befehlsvorrat, Befehlssatz
17. Adresse

Kapitel 6
NUN FÜGEN WIR ALLES ZUSAMMEN

Wir wollen uns nun die Logik der Zentraleinheit selbst ansehen. In Kapitel 5 wurde die Zentraleinheit oberflächlich wie folgt gezeigt:

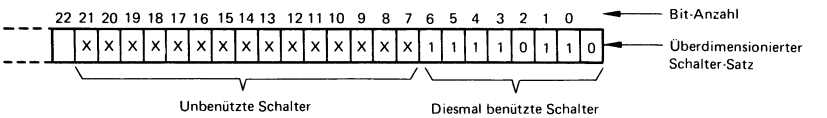


WORTGRÖSSE

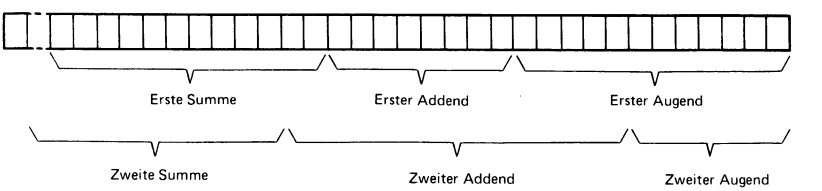
In Kapitel 4 sahen wir, wie Befehlscodes und Daten verschiedener Arten möglicherweise zu gleich aussehenden Bitmustern werden. Der wahrscheinlich wichtigste Schluß, den wir aus Kapitel 4 ziehen können, besteht darin, daß Schalter mit zwei Stellungen, oder Bits, allein nicht sehr nützlich sind. Jedoch Gruppen von Bits können unterschiedlich und sehr leistungsfähig interpretiert werden. **Daher ist die erste und wichtigste Entscheidung eines Mikroprozessor-Entwicklers die Auswahl der Anzahl von Bits, die der Mikroprozessor zur gleichen Zeit handhaben soll. Wir nennen diese Zahl die „Wortgröße" (oder Wortlänge) des Mikroprozessors.** Wir könnten einen Mikroprozessor entwickeln, der so viele verwendet, wie er gerade zu einer bestimmten Zeit benötigt. Alle realen Mikroprozessoren sind jedoch nicht auf diese Weise entwickelt. Sie haben irgendeine feste Wortlänge, die für jede Art der gehandhabten Informationen ausreicht. Betrachten wir arithmetische Vorgänge. Angenommen der Mikroprozessor habe 4213_{10} und 246_{10} zu addieren. Dies kann folgendermaßen gezeigt werden:

Dezimal	Binär	
	12 11 10 9 8 7 6 5 4 3 2 1 0 ←	Bit Nr.
4 2 1 3	1 0 0 0 0 0 1 1 1 1 0 1 0 1	Augend
+ 2 4 6	+ 1 1 1 1 0 1 1 0	Addend
= 4 4 5 9	= 1 0 0 0 1 0 1 1 0 1 0 1 1	Summe

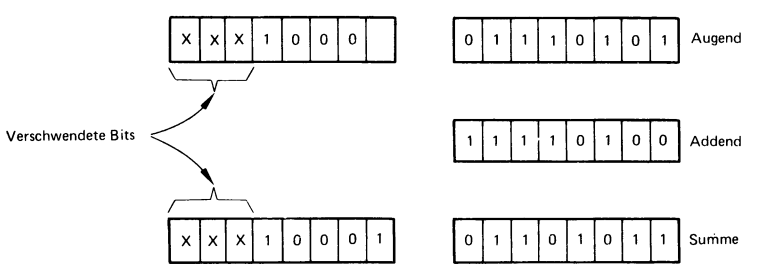
Die zwei Zahlen, die zu addieren sind, erfordern dreizehn Bits für den Augenden, acht Bits für den Addenden und dreizehn Bits für die Summe. Man könnte sich vorstellen, daß man einen Mikroprozessor so entwickelt, daß für das oben gezeigte Additionsbeispiel dreizehn Schalter dem Augenden und der Summe zugeordnet werden und acht Schalter dem Addenden. Aber was geschieht, wenn die nächste Addition sieben Schalter für den Augenden, fünfzehn Schalter für den Addenden und sechzehn Schalter für die Summe benötigt? Entweder müßte der Mikroprozessor eine übergroße Anzahl von Schaltern besitzen und zahlreiche Schalter die meiste Zeit verschwenden:



Oder der Mikroprozessor müßte die gleichen Schalter auf eine Vielzahl verschiedener Wege wiederverwenden. Und dies könnte unglaublich kompliziert werden:

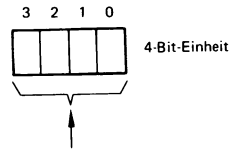


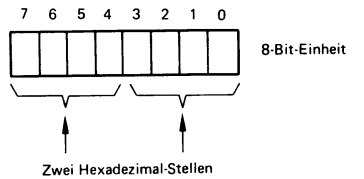
Die Art der oben gezeigten komplexen Summe ergibt keinen praktischen Vorteil gegenüber der Annahme einer festen Wortlänge für die Handhabung aller Informationen. Betrachten wir acht Bits (erinnern wir uns daran, daß acht Bits als Byte bezeichnet werden). Unser Additionsbeispiel würde bei der Ausführung in Byte-Einheiten folgendermaßen aussehen:

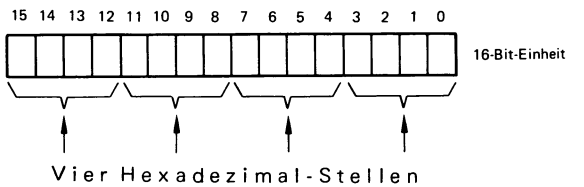


Es werden hier in der Tat einige Bits verschwendet, die vereinfachte Logik ist jedoch ein mehr als ausreichender Ausgleich.

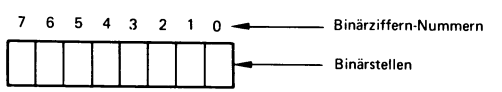
Die meisten der heute auf dem Markt bekannten Mikroprozessoren handhaben Informationen in Einheiten zu 8 Bits. Infolge dessen werden diese Mikroprozessoren als "8-Bit"-Bausteine bezeichnet. **Einige wenige überholte Mikroprozessoren verarbeiten Informationen in 4-Bit-Einheiten, während einige der neueren leistungsfähigeren Mikroprozessoren mit Einheiten zu 16 Bits arbeiten.** Man sieht, daß alle diese Wortlängen — 4, 8 und 16 alle so entworfen sind, daß sie ein einfaches Zählen im Binärsystem gestatten:





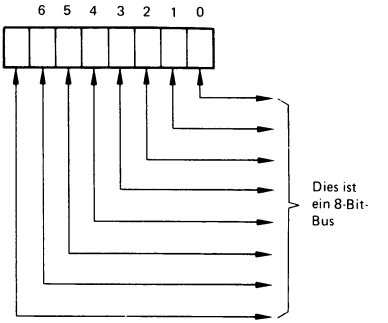


Sehen wir uns die Bedeutung einer festen Wortlänge an. Bei einem 8-Bit-Mikrocomputer müssen alle Informationen — Daten, Zeichen und Befehlscodes — in Einheiten zu acht binären Ziffern gespeichert werden. Wir werden die 8-Bit-Einheit folgendermaßen darstellen:

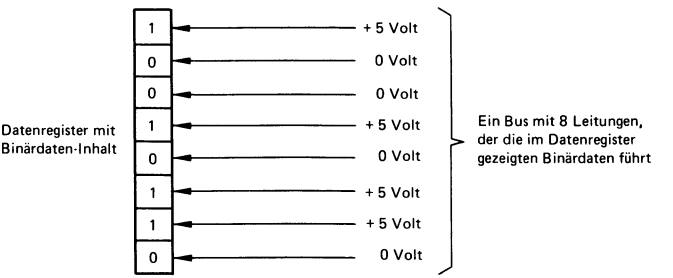


BUSSE

Informationen müssen auch von einem Teil eines Mikrocomputer-Systems zu einem anderen übertragen werden. Da alle Informationen als 8-Bit-Einheiten behandelt werden, müssen alle Informations-Übertragungen als 8 Bits zur gleichen Zeit auftreten. Daher erfolgen diese Transfers über acht parallele Leiter:

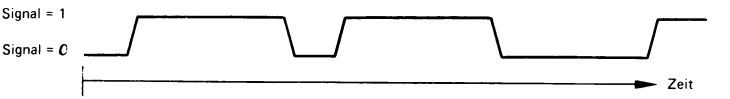


Wir nennen diese Leiter „Busse" (im Deutschen etwa Sammelschiene). Ein „Bus" ist nichts anderes als eine Sammlung von elektrischen Leitungen, über die binäre Daten transferiert werden. Ein 1-Bit- oder "Ein"-Schalter wird durch die Anwesenheit einer Spannung auf dem Leiter dargestellt. Ein 0-Bit- oder "Aus"-Schalter wird durch die Abwesenheit einer Spannung auf dem Leiter dargestellt. Dies kann folgendermaßen gezeigt werden:

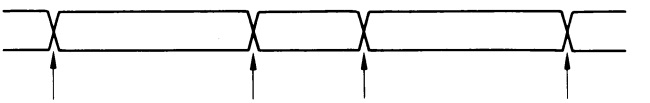


DARSTELLUNG VON SIGNALEN AUF DEN BUS-LEITUNGEN

Zur Identifizierung von Signalpegeln auf Bus-Leitungen greifen wir auf eine Art Kurzschrift zurück. Für eine einzelne Busleitung ziehen wir eine kontinuierliche Linie. Die Linie ist hoch, wenn Spannung vorhanden ist und sie ist niedrig, wenn keine Spannung vorliegt. Dies kann folgendermaßen gezeigt werden:

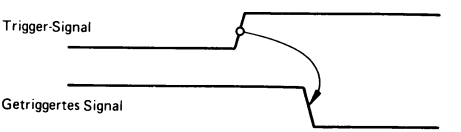


Wenn ein Bus mehr als eine Leitung besitzt, können einige Leitungen Spannung führen, während andere es nicht tun. Wir identifizieren den Moment, in dem eine oder mehrere Busleitungen ihren Zustand ändern, wie folgt:

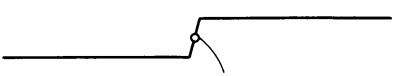


Eine oder mehrere Busleitungen können ihren Zustand an diesen Stellen ändern

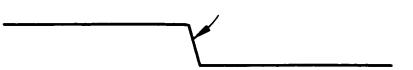
Manchmal bewirkt die Änderung des Zustandes eines Signales eine Zustandsänderung eines anderen Signales. Man bezeichnet dies als "Triggern" und stellt es folgendermaßen dar:



Es ist nicht von Bedeutung, wie sich ein Signalpegel ändert. Niedrig-zu-hoch könnte durch hoch-zu-niedrig ersetzt werden und umgekehrt. Wesentlich hierbei ist nur, daß die Änderung eines Signal-Pegels



ein anderes Signal veranlaßt, seinen Zustand zu ändern:



REGISTER

Die Schalter, die Informationen aufbewahren, werden „Register" genannt. Informationen werden zu und von Registern über Busse übertragen. Nun können wir unmit-

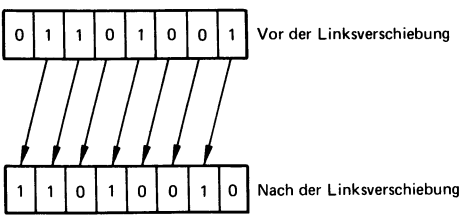
telbar sehen, daß ein 4-Bit-Mikroprozessor einfacher sein wird als ein 8-Bit-Mikroprozessor, da jedes Register oder jeder Bus nur vier Schalter oder vier Leiter erfordert, anstelle von acht. Ähnlich hat ein 16-Bit-Mikroprozessor eine kompliziertere Logik als ein 8-Bit-Mikroprozessor, da jedes Register oder Bus sechzehn Schalter oder Leiter erfordert, anstatt acht.

DIE ARITHMETISCHE UND LOGISCHE EINHEIT

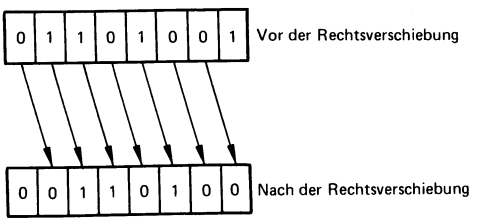
ALU

Die Wortgröße eines Mikroprozessors bezieht sich auch auf die Logik, mit der die Daten-Manipulationen durchgeführt werden. Die verschiedenen arithmetischen und logischen Operationen, die in Kapitel 4 beschrieben wurden, werden alle an Dateneinheiten mit fester Bitlänge ausgeführt, die gleich der Wortlänge des Mikroprozessors ist. Wir haben daher für einen 8-Bit-Mikroprozessor immer 8-Bit-Dateneingänge für jede arithmetische oder Boole'sche Operation und wir werden 8-Bit-Ergebnisse bilden. **Alle diese Logik ist auf einer Stelle zusammengefaßt und wird als arithmetische und logische Einheit (Arithmetic and Logic Unit = ALU, im Deutschen auch manchmal als Rechenwerk) bezeichnet.** Die arithmetische und logische Einheit hat ihren Namen von der Tatsache, daß sie arithmetische Operationen (Addition, Subtraktion) und logische Operationen ausführt. Grundlegende logische Operationen sind die Boole'schen Operationen UND, ODER, XOR und NICHT. Die arithmetische und logische Einheit wird gewöhnlich ALU genannt. Bild 6.1 zeigt funktionsmäßig die Logik einer arithmetischen und logischen Einheit.

Zusätzlich zu den beschriebenen arithmetischen und Boole'schen Operationen, die in Kapitel 4 beschrieben wurden, kann eine arithmetische und logische Einheit (ALU) wahrscheinlich auch Daten nach links verschieben:

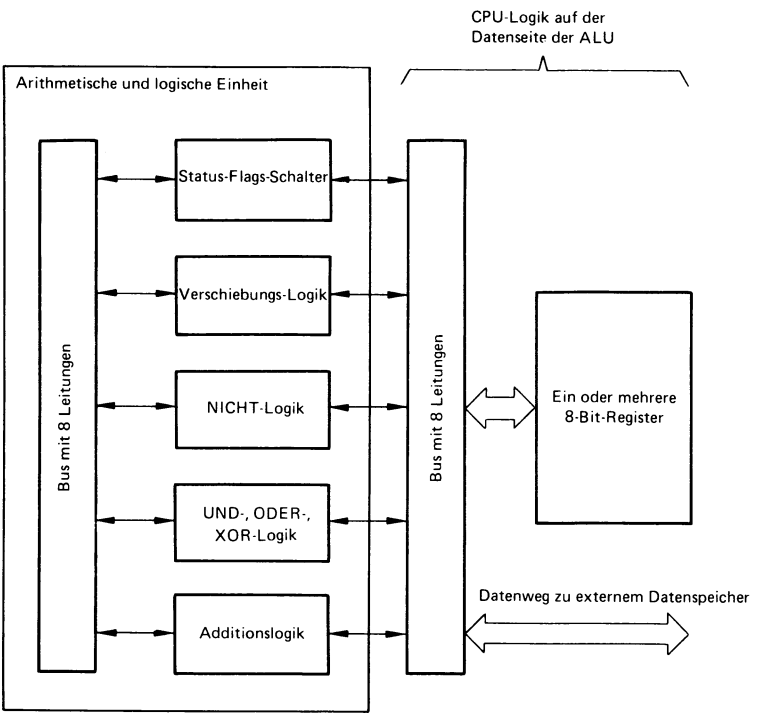


Und/oder wird auch Daten nach rechts verschieben können:



scheinlich mehr Wege zur Ausführung dieser Aufgabe, als Logikentwickler gibt. Außerdem ist für uns als Mikroprozessor-Anwender dieses Thema uninteressant. Aber erinnern wir uns daran, egal wie kompliziert eine gewünschte Computeroperation auch sein mag, sie wird letztlich in eine Serie von Schritten aufgeteilt, von denen jede das Senden von binären Dateneingaben zur ALU und den Empfang binärer Datenresultate von der ALU enthält.

Da die arithmetische und logische Einheit (ALU) Operanden von den Daten-Registern empfängt und die Ergebnisse zu Datenregistern zurückgibt, müssen wir entsprechende Daten-Register auf einer Seite der ALU haben:



STEUEREINHEIT

Auf der anderen Seite benötigt die ALU ein Register, das die Befehle enthält, die bestimmen, welcher Teil der ALU verwendet wird und wie sie verwendet wird. Die Bestimmung „welcher" und „wie" wird von einem neuen Logikblock ausgeführt, den wir die „Steuereinheit" (oder Steuerwerk) nennen wollen:

Die 0- und 1-Bits, die in den Abbildungen für die Verschiebungen gezeigt wurden, sind willkürlich gewählt und haben keine besondere Bedeutung.
Wir werden uns nun nicht mit dem tatsächlich verwendeten Verfahren zur Bildung einer Additionslogik befassen — oder irgendeiner anderen Logik innerhalb der ALU. Nur wer sich tatsächlich mit der Entwicklung von Mikroprozessoren befaßt, benötigt diese Details. Erwähnenswert ist, daß die arithmetische und logische Einheit (ALU) einen oder mehrere Busse besitzt, mit denen die Daten hereingebracht werden und einen oder mehrere Busse, über die Daten ausgesendet werden. (Die ALU wird wahrscheinlich einen oder mehrere interne Busse besitzen, wie in Bild 6.1 gezeigt ist, aber um diese brauchen wir uns nicht zu kümmern.) Busse sind 4-Bit-Busse für einen 4-Bit-Mikroprozessor, 8-Bit-Busse für einen 8-Bit-Mikroprozessor und 16-Bit-Busse für einen 16-Bit-Mikroprozessor.

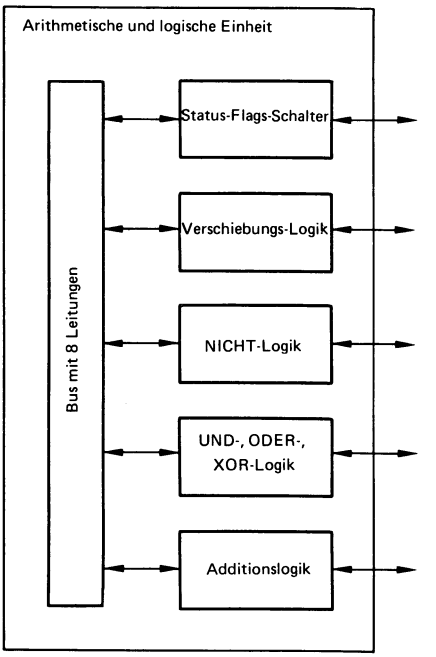
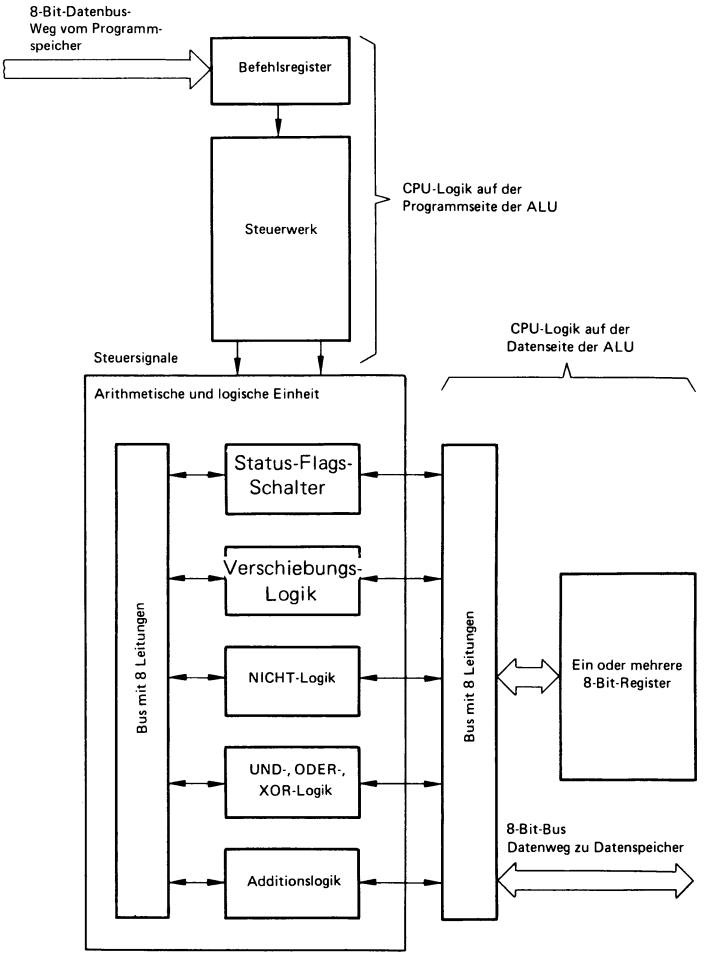


Bild 6.1 Die arithmetische und logische Einheit

Der wesentlichste Aspekt aus Bild 6.1 ist die Tatsache, daß eine arithmetische und logische Einheit (ALU) vorhanden sein muß und daß sie die tatsächlichen erforderlichen Datenmanipulationen ausführt. Damit diese Datenmanipulationen tatsächlich ablaufen, müssen die Daten von den Daten-Registern über entsprechende Busse zur arithmetischen und logischen Einheit (ALU) transferiert werden. Daher ist das einzige „Magische", das in jeder Computeroperation verbleibt, der tatsächliche Vorgang, mit dem Operationen innerhalb der arithmetischen und logischen Einheit (ALU) auftreten. Wir werden die interne ALU-Logik nicht beschreiben, da es wahr-



DIE ZUSÄTZLICHE CPU-LOGIK

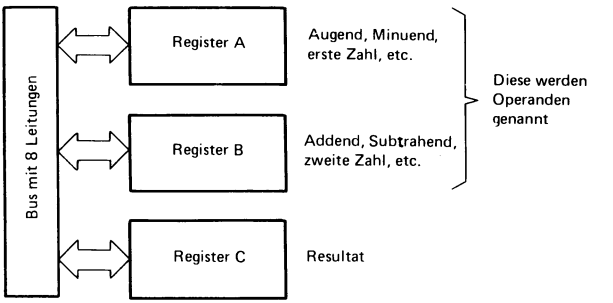
Sehen wir uns nun die Logik auf jeder Seite der ALU an.

DATENREGISTER

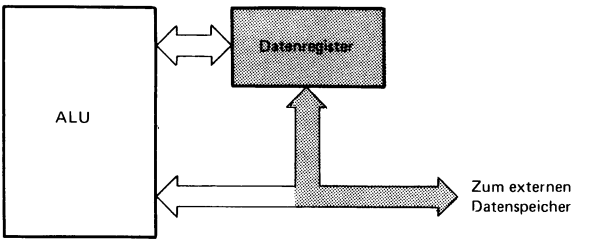
OPERANDEN

Auf der Datenseite der ALU benötigen wir ein oder mehrere Register zum Aufbewahren von Daten, die von der ALU bearbeitet werden. Es gibt keinen „natürlichen"

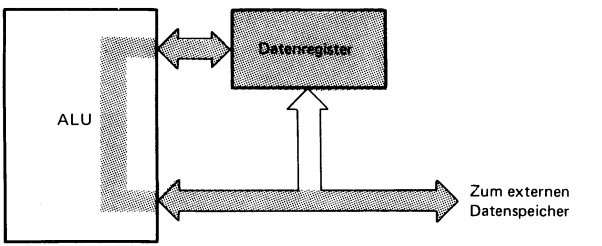
Grund für eine bestimmte Anzahl von Datenregistern, die jeder Entwickler an der Datenseite der ALU vorsieht. Wir könnten beispielsweise annehmen, daß drei Register ideal wären, da viele arithmetische und Boole'sche Operationen zwei Daten verwenden und ein Resultat bilden:



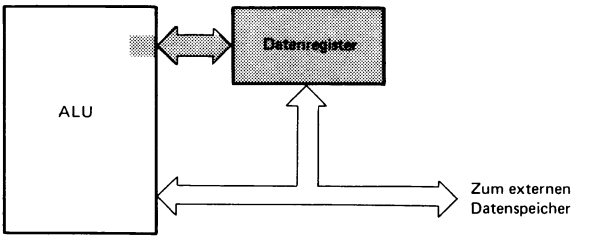
Wenn wir jedoch die arithmetische oder Boole'sche Operation in eine Anzahl von Schritten aufteilen, könnten wir mit nur einem Datenregister auskommen. Hier sind die notwendigen Schritte:
1) Bringe den ersten Operanden zum Datenregister:



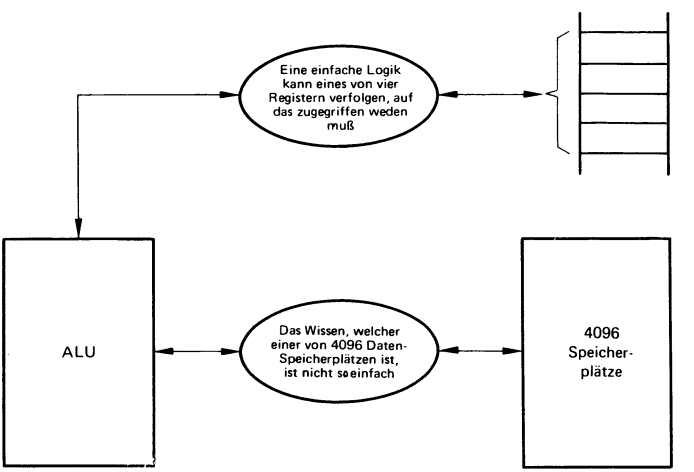
2) Führe irgendeine ALU-Operation aus, indem ein Operand vom Datenregister geholt wird, und der andere Operand direkt von einem externen Speicher:



3) Bringe das Ergebnis zurück zum Datenregister:

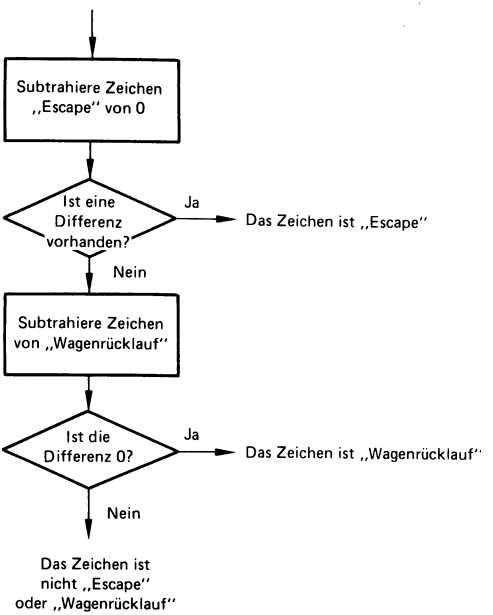


Aber es gibt auch gute Gründe für das Vorhandensein von mehr als drei Datenregistern auf der Datenseite der ALU. Daten können zwischen einem Register und der ALU wesentlich schneller transferiert werden, als Daten zwischen einem externen Datenspeicher und der ALU. Der Grund hierfür ist, daß nur sehr wenige Datenregister in der CPU vorhanden sind, so daß wir unmittelbar das Register, auf das zugegriffen werden soll, identifizieren können. Es gibt eine große Anzahl von externen Speicherplätzen für Daten, und dies bedeutet, daß jedesmal wenn die Zentraleinheit (CPU) auf einen externen Daten-Speicherplatz zugreift, ein ganzer Schritt zur Identifizierung des Speicherplatzes eingeschlossen werden muß. Anders ausgedrückt, die Logik der Zentraleinheit (CPU) muß eine gewisse Zeit aufwenden, um exakt festzulegen, auf welchen Daten-Speicherplatz zugegriffen werden soll. Dies kann folgendermaßen gezeigt werden:



VERWENDUNG VON DATENREGISTERN

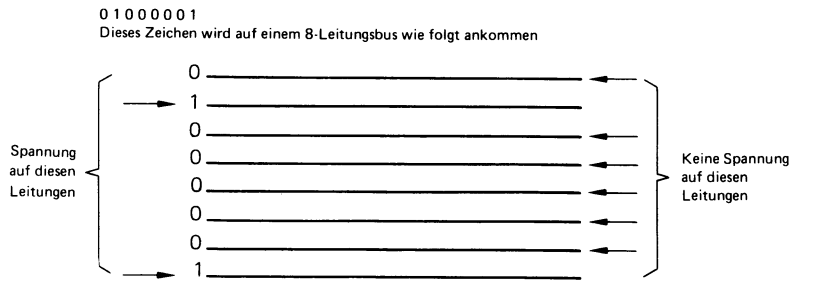
Wenn wir viele Datenregister haben, gestattet uns ein Mikroprozessor Daten innerhalb der CPU aufzubewahren, zu denen wir häufig zugreifen müssen. Um die optimale Anzahl von Datenregistern für eine CPU zu bestimmen, müssen wir mehr über die Funktion der Datenregister wissen. Wir weichen daher etwas von unserer Besprechung ab, um uns ein reales Beispiel anzusehen, wie Datenregister verwendet werden. In Joe Bitburgers Programm zum Bezahlen von Rechnungen testet das Programm jedesmal, wenn Joe ein Zeichen über die Tastatur eingibt, dieses Zeichen um festzustellen, ob es ein "Escape" oder ein "Wagenrücklauf" ist. Nun haben manche Mikrocomputer keinen "Test"-Befehl. **Beim Fehlen eines Test-Befehls können wir nun einen Test dadurch ausführen, indem wir den ankommenden Zeichencode vom "Escape"-Zeichen subtrahieren und dann vom "Wagenrücklauf"-Zeichen.** Die Logik hierfür kann folgendermaßen dargestellt werden:



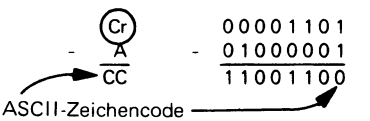
Wir wollen ein „Wagenrücklauf" (Carriage Return-)Tastaturzeichen mit dem Symbol (Cr) darstellen. Hat es einen Sinn, einen Buchstaben des Alphabets von einem "Wagenrücklauf" zu subtrahieren?

(Cr) - A = ?

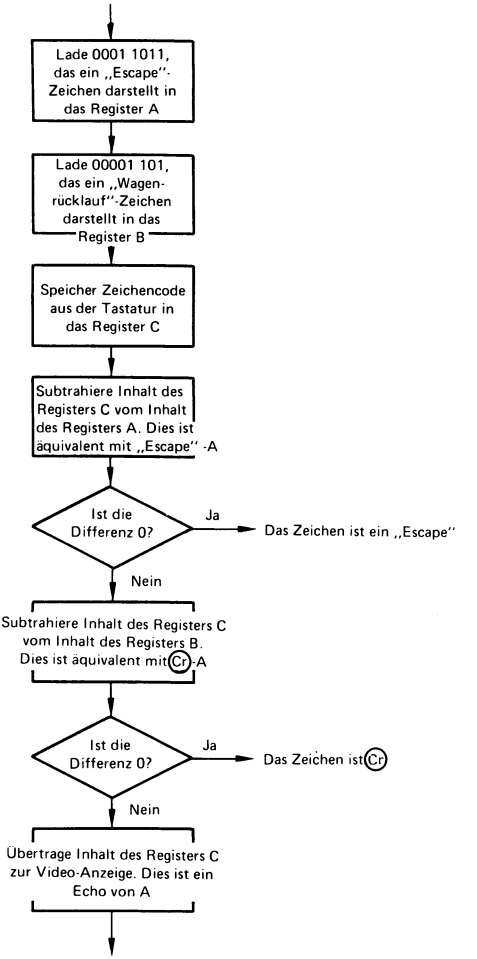
Für einen Menschen mag diese Subtraktion sinnlos sein. Aber erinnern wir uns daran, Mikrocomputer stellen Zeichen mittels binärer Ziffernmuster dar — und sie stellen auch Zahlen mit binären Ziffernmustern dar. Erinnern wir uns daran, wie oft wir in Kapitel 4 gesagt haben, daß wir nicht vergessen dürfen, wie binäre Ziffernmuster zu interpretieren sind? Nun sehen wir, daß dies Vorteile hat, wir andererseits aber auch an mehr denken müssen. Angenommen, Joe Bitburger drückt die A-Taste auf seiner Tastatur. Der Mikrocomputer sieht folgendes binäre Ziffernmuster von der Tastatur ankommen:



Soweit es den Mikrocomputer betrifft, ist dies ein einfaches binäres Ziffernmuster. Wenn wir es in den Speicher legen und später wieder verwenden, so könnten wir es zu einer Videoanzeige zur Darstellung des A im Echo-Betrieb senden. Die Videoanzeige wurde so entwickelt, daß sie alle ankommenden Bitmuster als ASCII-Zeichen interpretiert. Jedoch innerhalb der Zentraleinheit (CPU) ist ein Bitmuster einfach ein Bitmuster. Daher können wir das A-Bitmuster vom Wagenrücklauf-Bitmuster subtrahieren, indem wir beide als reine binäre Daten behandeln. Dies kann folgendermaßen gezeigt werden:



So lange wir nichts anderes in die Register einschreiben, die die A oder Cr-Zeichen beinhalten, wird der Inhalt des Registers aufbewahrt. Wenn wir das nächste Mal auf die gleichen Register zugreifen, können wir den Inhalt als ASCII-Zeichen interpretieren. Wir könnten beispielsweise den A-Zeichencode zur Videoanzeige senden, um ein Echo gleich nach der Subtraktion des A-Zeichencodes von einem Wagenrücklauf zu erzeugen. Wir können nun die ganze Befehlsfolge, die die ankommenden Zeichen empfängt und testet, unter Verwendung dieses vollständigeren Flußdiagrammes darstellen:



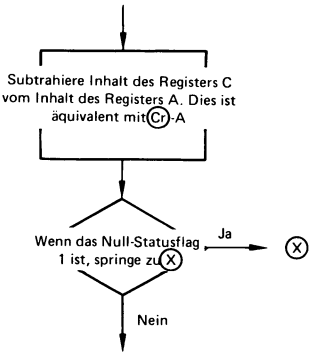
NULL-STATUSFLAG

Wenn wir jeder Subtraktion folgen, so werden wir ein Statusflag verwenden um zu bestimmen, ob das Zeichen ein Wagenrücklauf oder ein Escape-Code ist. Nahezu alle Mikroprozessoren besitzen ein Statusflag, das der Null-Status genannt wird. Dieses Flag wird zur Feststellung des Ergebnisses Null wie folgt verwendet:

Wenn das Resultat 0 ist, ist das Null-Statusflag 1

Wenn das Resultat nicht 0 ist, ist das Null-Statusflag 0

In einer etwas verdrehten Logik wird das Null-Statusflag immer auf 1 für ein Null-Ergebnis gesetzt und wird auf 0 für ein Ergebnis das nicht Null ist gesetzt. Wir können nun unsere Zeichen-Testlogik über die nachstehende Befehlsfolge bilden:



(X) ist der Befehl, der auszuführen ist, wenn ein Wagenrücklauf anstatt eines A über die Tastatur eingegeben wurde. Wir wählen diesen Befehl aus und identifizieren ihn durch seine Programm-Speicheradresse — oder anders ausgedrückt, die Adresse des Programm-Speicherplatzes, in dem der Befehlscode gespeichert ist.

All dies liegt etwas außerhalb unserer momentanen Betrachtung — der richtigen Anzahl von Registern, die man auf der Datenseite der ALU haben sollte. Aber da wir nun verstehen, wie Datenregister verwendet werden, können wir die "richtige" Anzahl von Datenregistern bestimmen, die jeder Mikroprozessor haben sollte? Unglücklicherweise gibt es keine "richtige" Anzahl, sondern der Entwickler des Mikroprozessors muß einige Abstriche machen. Wie viel von der begrenzten Logik kann man für Register reservieren, und wieviel für andere Dinge? Erinnern wir uns daran, daß wir acht Schalter für jedes Register benötigen sowie Verbindungen vom Register zu Bussen innerhalb des Mikroprozessors. Der gleiche Platz für Schalter und Bus könnte in einer Vielzahl anderer Wege verwendet werden.

CPU-SPEICHER

Gewöhnlich werden wir zwischen 4 und 8 Registern auf der Datenseite der ALU finden. Es gibt jedoch auch Fälle mit nur einem Register und solche mit 32. Es gibt auch Mikroprozessoren, die einen kleinen Teil des Datenspeichers innerhalb der CPU haben. Es können 64 oder mehr Speicherplätze innerhalb der Zentraleinheit sein und manchmal mehr oder weniger von diesen Speicherplätzen vorhanden sein als externe Datenspeicherplätze.

DAS BEFEHLSREGISTER UND DAS STEUERWERK

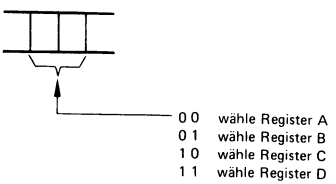
Auf der Programmspeicher-Seite der ALU muß ein einzelnes Register vorhanden sein in dem wir den Binärcode aufbewahren können, der den momentan auszuführenden Befehl darstellt. Wir bezeichnen dies als das Befehls-Register.

Angenommen der momentan auszuführende Befehl addiert den Inhalt der Datenregister A und B und legt die Summe zurück in das Datenregister A. (Wir nehmen für den Moment an, daß Datenregister A und B auf der Datenseite der CPU existieren.) Der

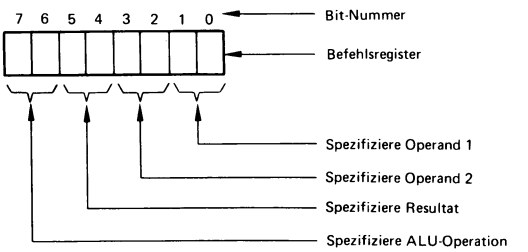
entsprechende binäre Befehlscode wird (als Daten) zur Zentraleinheit gebracht, wo er in das Befehlsregister gespeichert wird.
Der Inhalt des Befehlsregisters arbeitet als einer von 256 verschiedenen „Triggern" (Auslösern) für einen Logikblock, der Steuereinheit oder Steuerwerk genannt wird. Es gibt 256 verschiedene „Trigger", da es 256 Kombinationen der 0- und 1-Bits gibt, die wir aus acht Bits bilden können. Ein 16-Bit-Mikroprozessor würde ein 16-Bit-Befehlsregister haben — mit 65536 verschiedenen Kombinationen von 0 und 1.
Der Befehlsregister-„Trigger" startet eine Folge von Signalen, die vom Steuerwerk ausgegeben werden, um den Datentransfer und die ALU-Operationen freizugeben, die von dem auszuführenden Befehl verlangt werden.
Wir müssen nicht sehr viel über das Steuerwerk, das Befehlsregister und ihrer Zusammenarbeit wissen. Das Steuerwerk erzeugt die Steuersignale, die die Daten bewegen wo dies erforderlich ist und aktiviert die ALU-Logik zur richtigen Zeit. Bei den momentan auf dem Markt befindlichen Mikrocomputern können wir als Anwender der Mikrocomputer nichts am Steuerwerk ändern. Wir können beispielsweise nicht die Art ändern, mit der dieses auf einen Befehlscode reagiert. Das Steuerwerk arbeitet als Verbindungsglied zwischen einem Befehl und den Vorgängen, die bei der Ausführung eines Befehles auftreten. Das ist alles was wir über das Steuerwerk wissen müssen.
Wichtig ist jedoch die Größe des Befehlsregisters. Ein Befehlsregister mit 8 Bits kann nur 256 verschiedene Operationen und Variationen dieser Operationen spezifizieren. Dies liegt darin begründet, daß es nur 256 verschiedene Muster der 0- und 1-Ziffern gibt, die man aus acht binären Ziffern bilden kann. 256 mag als eine große Zahl erscheinen, ist es in Wirklichkeit jedoch nicht. **Betrachten wir Operationen der arithmetischen und logischen Einheit.** Innerhalb dieses Logik-Blocks haben wir nur sieben Operationen definiert:

Linksverschieben
Rechtsverschieben
Komplementieren
UND
ODER
Exklusiv-ODER
Addieren

Obwohl es nur sieben ALU-Operationen gibt, müssen die Befehle mehr definieren. Sie müssen die Quelle des Operanden definieren sowie den Bestimmungsort (Zielort) für das Resultat. Angenommen wir hätten vier Datenregister (Register A, B, C und D). Wir benötigen zwei der acht Befehls-Objektcode-Bits jedesmal, wenn wir eines der vier Register zu identifizieren haben:



Anders ausgedrückt, ein Befehlscode muß absolut spezifiziert sein. Wenn wir vier Register-Möglichkeiten besitzen, so werden wir vier getrennte und bestimmte binäre Zifferncodes zur Spezifizierung der ausgewählten Möglichkeit benötigen. **Daher kann ein 8-Bit-Befehlscode folgendermaßen dargestellt werden:**



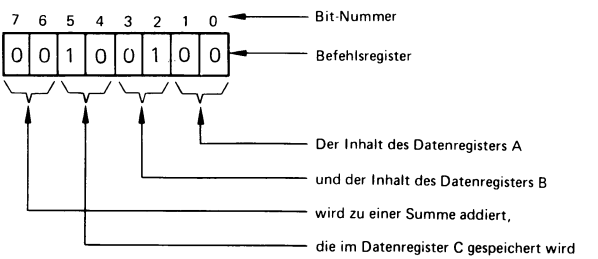
Wir könnten willkürlich annehmen, daß für jedes der drei Register-Spezifikationen das Steuerwerk die Befehlscode-Bits wie folgt interpretiert:

00 – Wähle Datenregister A
01 – Wähle Datenregister B
10 – Wähle Datenregister C
11 – Wähle Datenregister D

Die Bits 6 und 7 des Befehlsregisters könnten vom Steuerwerk wie folgt interpretiert werden:

Bits	
7	6
0	0 Addieren
0	1 UND
1	0 ODER
1	1 Exklusiv-ODER

Hier ist ein Beispiel, wie ein Befehlscode interpretiert werden könnte:



Wir haben nun genau dargestellt, wie das Steuerwerk den Inhalt des Befehlsregisters decodiert, die Darstellung, obwohl sie grundsätzlich richtig ist, ist unpraktisch.
Die auf Seite 6-17 gezeigten Bitmuster passen nun für vier der sieben ALU-Operationen — die vier, die zwei Eingangs-Operanden benötigen. Jedoch diese vier Operationen zusammen mit ihren Möglichkeiten, brauchen alle 256 Befehlscodes auf. Es bleiben keine Codes für irgendwelche anderen ALU-Operationen übrig, sowie keine für die Befehle, die die ALU nicht verwenden. **Offensichtlich sind 256 mögliche Befehls-Kombinationen nicht viel. In dem Buch „Einführung in die Mikrocomputer-Technik" werden wir untersuchen, wie Mikroprozessor-Entwickler die begrenzte Anzahl der Befehlscode-Möglichkeiten für alle Arten von erforderlichen Befehlen erweitern, indem eine begrenzte Möglichkeit innerhalb jeder Klasse von Befehlen gegeben wird. Dies ist ein leicht zu verstehender Vorgang, da es dasselbe ist, wie das Auskommen mit einem begrenzten Budget. Wenn wir nicht genügend Geld haben, um alles zu tun was wir wollen, so müssen wir uns in allen Bereichen einschränken, indem wir den besten Kompromiß zwischen unserem Lebensstil und Einkommen schaffen.**

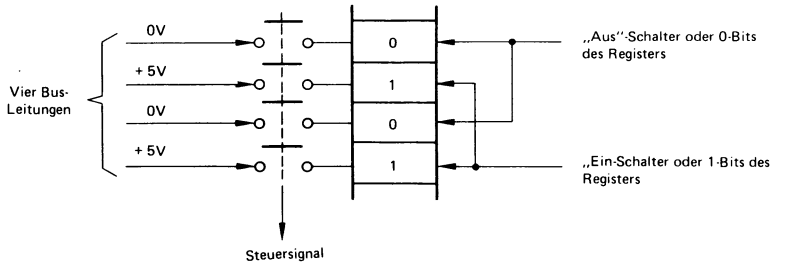
LOGISCHE KONZEPTE UND ZEITLICHE STEUERUNG

Obwohl die genaue Arbeitsweise einer Steuereinheit für uns ohne Bedeutung ist, gibt es einige logische Konzepte, die wichtig sind, da sie überall universell innerhalb von Mikrocomputer-Systemen auftreten. Wir werden uns nun diese Konzepte ansehen.
Jede Folge von logischen Operationen innerhalb der CPU (oder jedem anderen Teil eines Mikrocomputer-Systemes) besteht aus der Bewegung binärer Daten und der Änderung binärer Daten.
Innerhalb der CPU müssen binäre Daten zwischen den Datenregistern der ALU und externer Logik bewegt werden. Binäre Daten werden nur innerhalb der ALU geändert.
In jedem anderen Teil eines Mikrocomputer-Systems werden Daten zwischen Registern bewegt und die Daten werden durch eine Logik, die einfacher als die ALU jedoch sehr ähnlich ist, modifiziert.

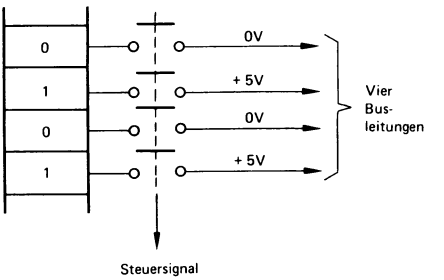
LOGIK ZUR BEWEGUNG BINÄRER DATEN

TOR-ODER GATTER-SIGNALE

Um binäre Daten zu bewegen, muß ein Steuersignal erzeugt werden, das als „Stecker" oder Verbindungsglied zwischen einem Register und einer Busleitung arbeitet. Das Steuersignal wird manchmal als ein Tor oder Gatter bezeichnet. Wir können Daten von Busleitungen in ein Register folgendermaßen bringen:



Die Schalterstellung (oder Register-Bit) kann auf die Busleitung folgendermaßen gelegt werden:

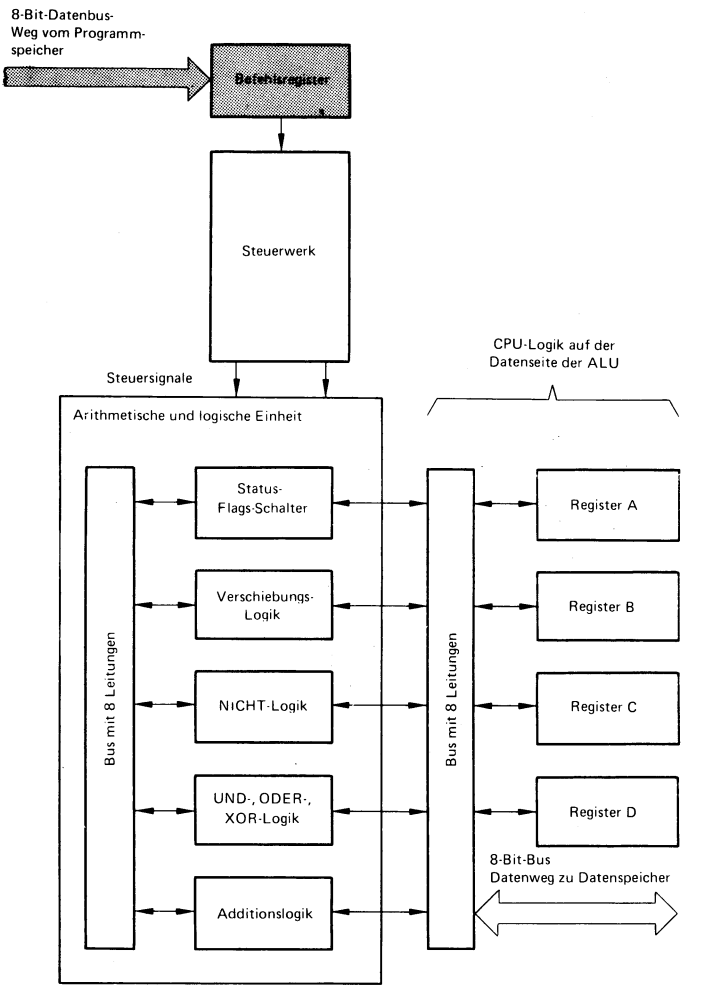


ZEITLICHE STEUERUNG DES BEFEHLS

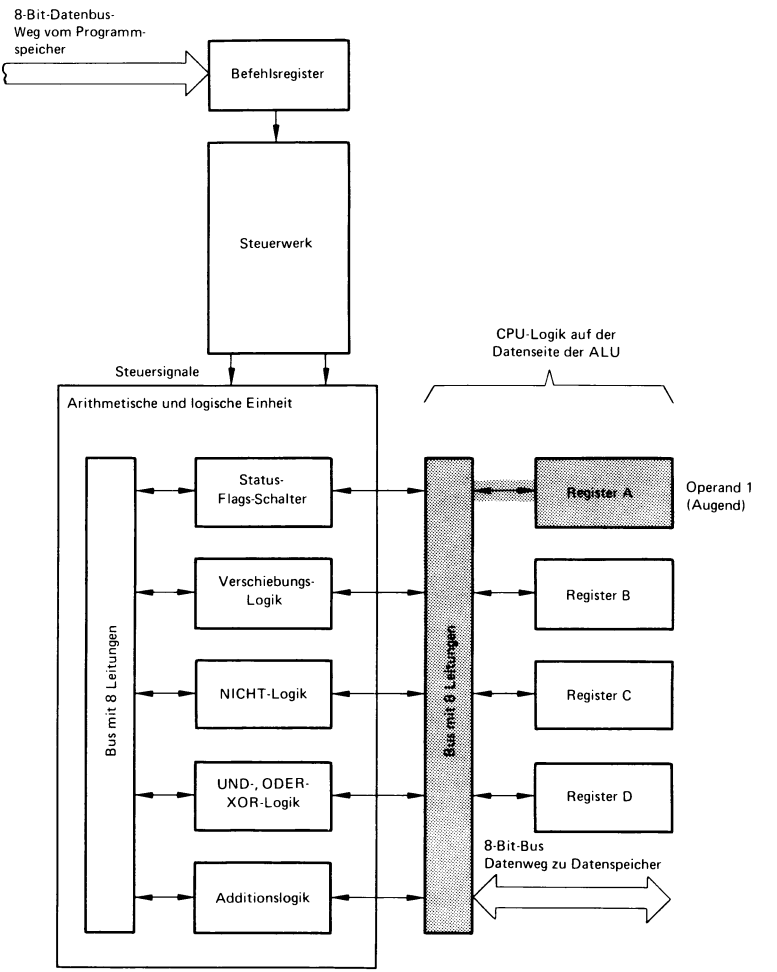
Die zeitliche Steuerung (Timing) ist während dieser Datentransferoperation außerordentlich wichtig, da es eine bestimmte Zeit erfordert, bis Spannungspegel auf Busleitungen aufscheinen oder verschwinden. Und solange nicht ein gleichbleibender Zustand auf der Busleitung existiert, kann die Logik die Busleitung nicht zu irgendeinem neuen Schalter verbinden.

Daher muß die Folge der Vorgänge entsprechend jeder Befehlsausführung in einer ganz bestimmten Art und Weise aufscheinen. Betrachten wir unser Additionsbeispiel. Dies würde die notwendige Folge von Vorgängen innerhalb der Zentraleinheit sein:

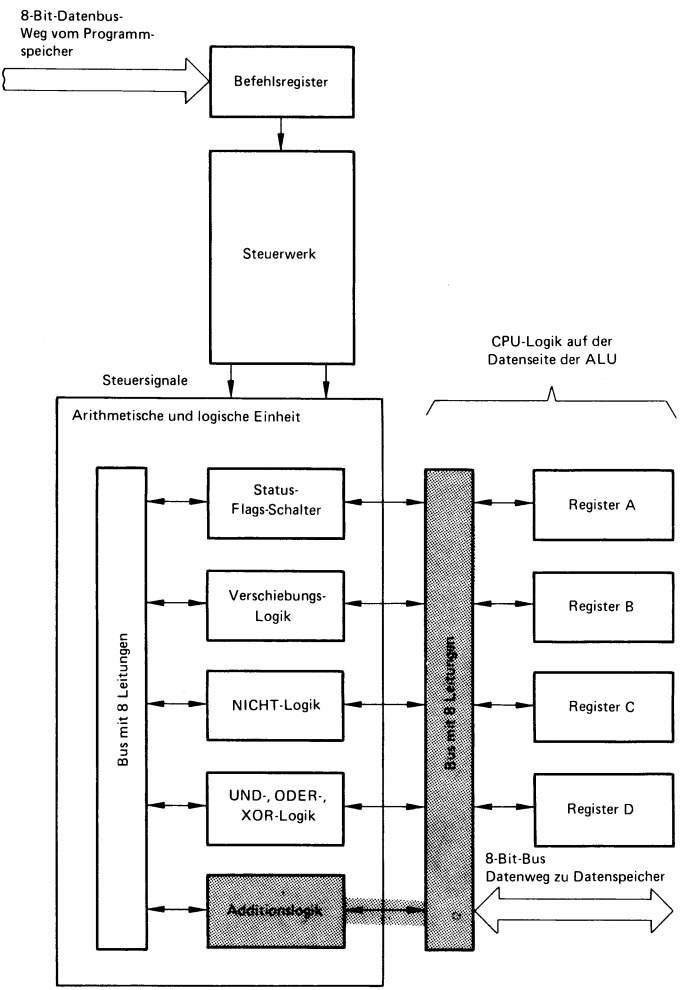
Schritt 1) Bringe den Befehlscode in das Befehlsregister:



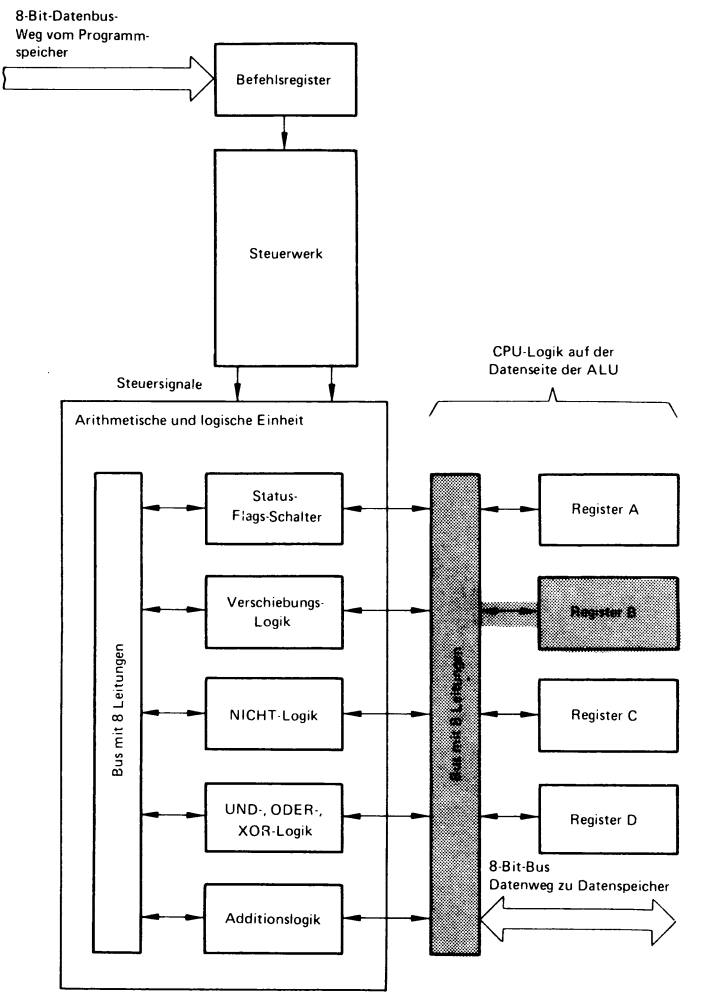
Schritt 2) Bringe den Inhalt des Registers A auf den Bus:



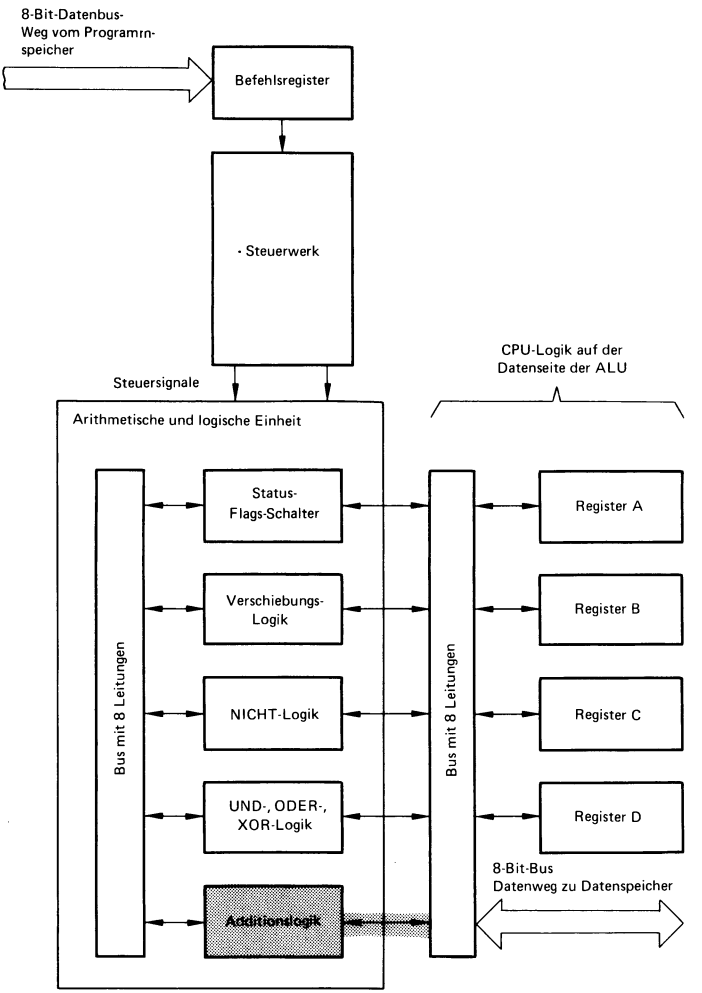
Schritt 3) Verbinde Bus mit der Additions-Logik:



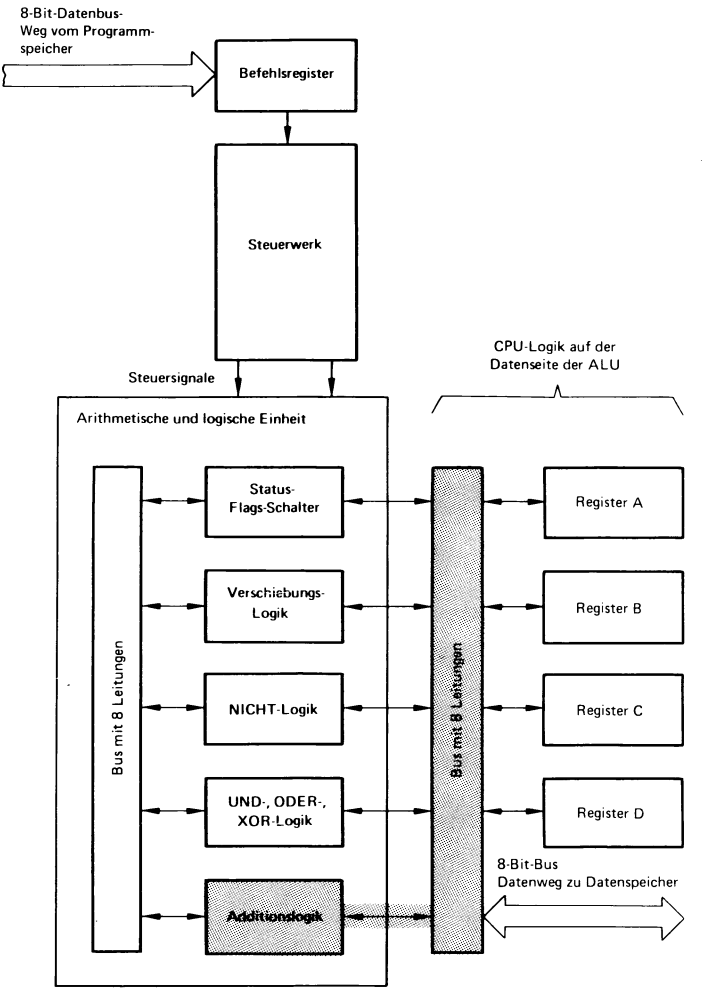
Schritt 4) Bringe Inhalt des Registers B auf den Bus:



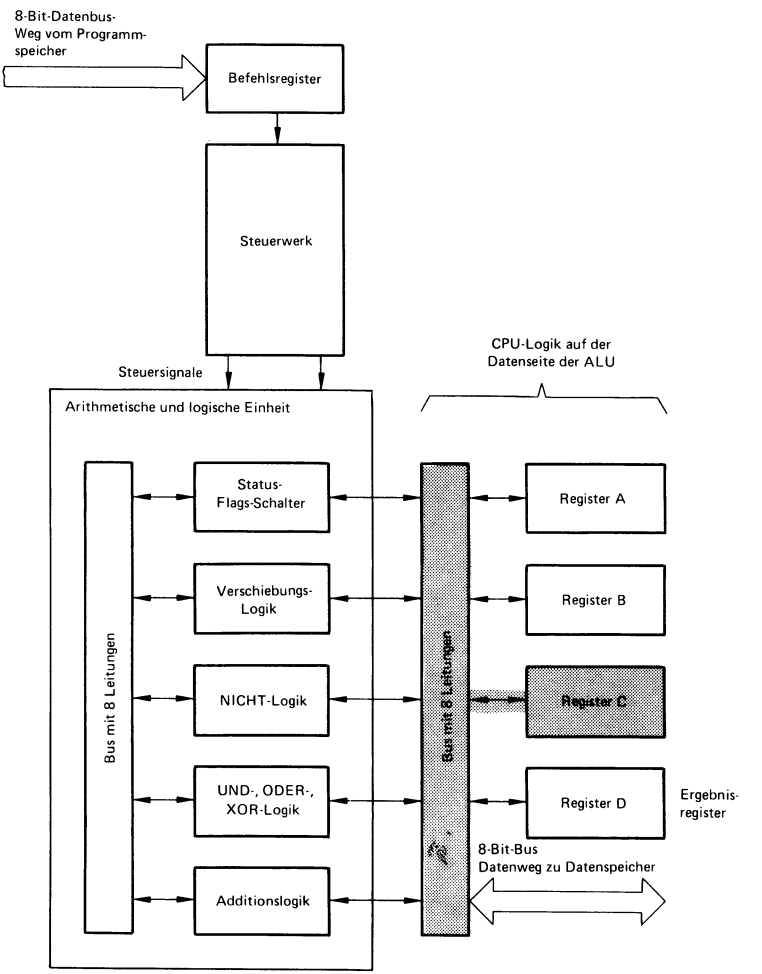
Schritt 5) Addiere:



Schritt 6) Bringe Summe auf den Bus:

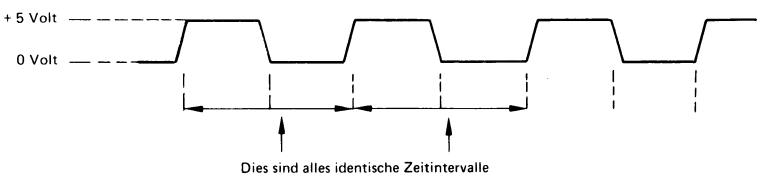


Schritt 7) Bringe Summe in das Register C:



DAS TAKTSIGNAL UND DIE ZEITLICHE STEUERUNG DER BEFEHLSAUSFÜHRUNG

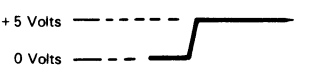
Eine Folge von Vorgängen, wie die eben beschriebenen sieben Schritte, wird durch ein Taktsignal festgelegt, das seinen Namen infolge der gleichmäßigen, periodischen Spannungsimpulse erhalten hat. Ein Taktsignal kann folgendermaßen dargestellt werden:



Das oben gezeigte Taktsignal wird zwischen 0 Volt und +5 Volt hin- und hergeschaltet. Diese Spannungen findet man sehr häufig in Mikrocomputer-Schaltkreisen, es gibt jedoch keinen besonderen Grund, weshalb nicht auch jede andere beliebige Spannung gewählt werden kann. Es ist nur erforderlich, daß zwei Spannungen zur Darstellung eines Taktsignales verwendet werden. Die beiden gleichen Spannungspegel können auch zur Darstellung von binären 0- und 1-Ziffernpegel verwendet werden. Ältere Mikroprozessoren verwenden mehr als zwei Spannungspegel. Zusätzliche Spannungspegel vereinfachen die Entwicklung der Chips. Sie spielen jedoch keine Rolle in der Darstellung von binären 0- und 1-Ziffern.

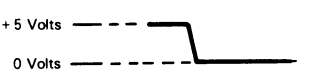
SIGNAL-ANSTIEGSFLANKE

Der Übergang von der Spannung 0 zu irgendeinem Spannungspegel wird als Anstiegsflanke des Taktsignales bezeichnet:



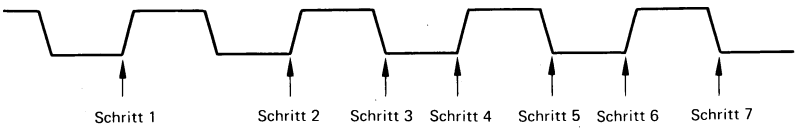
SIGNAL-RÜCKFLANKE

Der Übergang von einer Spannung zum Pegel 0 wird als Rückflanke oder abfallende Flanke des Taktsignales bezeichnet:



Alle Signale besitzen eine Anstiegs- und eine abfallende Flanke, nicht nur Taktsignale.

Die beiden Flanken des Taktsignales werden zur zeitlichen Steuerung von Vorgängen innerhalb der Zentraleinheit und im gesamten Mikrocomputer-System verwendet. Für unser Additions-Beispiel kann dies grundsätzlich folgendermaßen dargestellt werden:

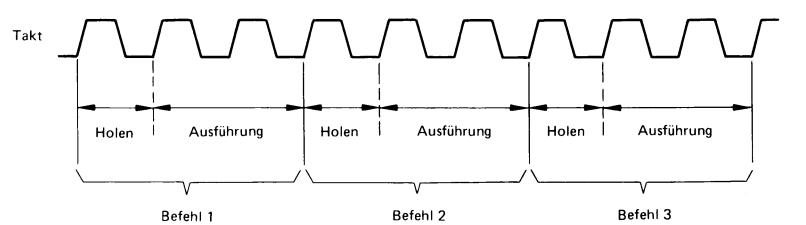


Nach dem Schritt 1 muß ein größeres Zeitintervall folgen, da man dem Steuerwerk genügend Zeit geben muß um den Inhalt des Befehlsregisters zu decodieren.
Verschiedene Mikrocomputer haben verschiedene Arten von Taktsignalen, die von sehr einfachen bis zu sehr komplexen Signalen reichen. Es kann auch mehr als ein Taktsignal verwendet werden, in welchem Fall jedoch die verschiedenen Taktsignale eine bestimmte Beziehung zueinander haben. **Aber in jedem Fall ist es das Taktsignal, das als „Dirigent" für die gesamte Logik arbeitet.**

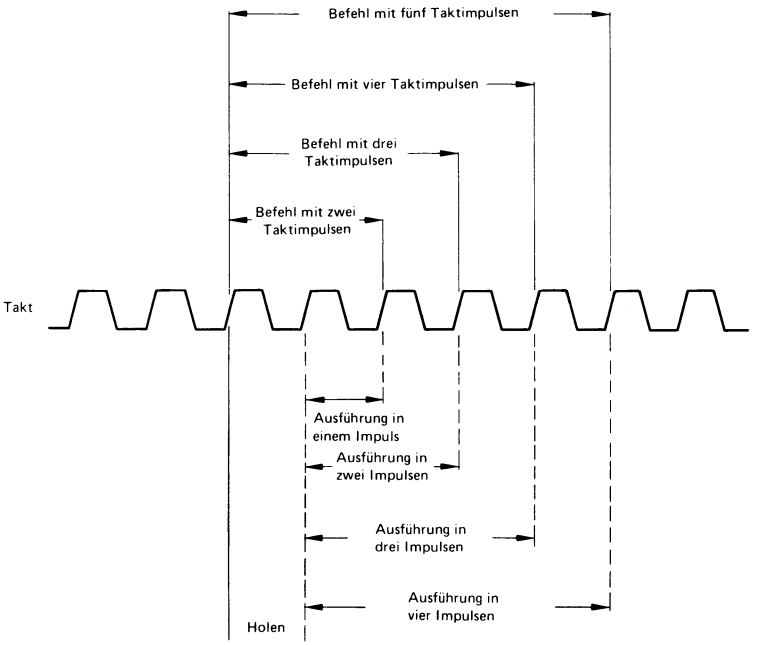
BEFEHLHOLEN

Sehen wir uns wieder die Folge von Ereignissen bei der Ausführung eines Befehls an.

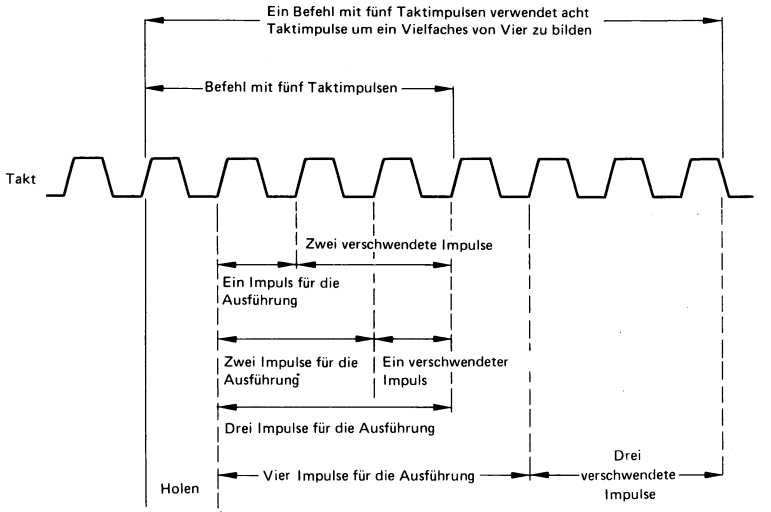
Jede Befehlsausführung muß mit einem Schritt beginnen, der den binären Befehlscode aus dem Programmspeicher zum Befehlsregister holt. Wir nennen diesen Schritt das „Befehlholen" (Instruction Fetch). Sobald der binäre Befehlscode sich im Befehlsregister befindet, übernimmt das Steuerwerk – nachdem es entsprechend durch das Befehlsregister getriggert wurde. Das Steuerwerk erhält so viel Zeit, wie erforderlich, um die logischen Vorgänge zu bilden, die vom Befehl verlangt werden. Nachdem alle logischen Abläufe vollendet sind, startet das Steuerwerk das Holen des nächsten Befehls. **Daher kann die Ausführung jedes Befehls in eine Befehlholphase und eine Befehlausführungsphase eingeteilt werden. Dies sieht folgendermaßen aus:**



Die Befehlholphase ist für jeden Befehl gleich lang. Die Befehlausführungsphase benötigt jedoch unterschiedliche Zeiten für verschiedene Befehle. Einige Mikroprozessoren haben variable Befehlausführungs-Zeiten. Um jedoch die Logik zu vereinfachen, verwenden andere Mikroprozessoren nur eine, oder sehr wenige, feste Befehlausführungs-Zeiten. **Nehmen wir beispielsweise einmal an, daß verschiedene Befehle Ausführungsphasen verwenden, die ein, zwei, drei oder vier Taktimpulse dauern.** Wenn die Befehlholphase einen Taktimpuls lang dauert, dann könnten wir den Mikroprozessor so entwerfen, daß er Befehle in zwei, drei, vier oder fünf Impulsen ausführt:



In diesem gegebenen Fall kann ein Mikroprozessor-Entwickler entscheiden, alle Befehle innerhalb fünf Taktimpulsen auszuführen, wobei er die unbenützten Taktimpulse für die kürzeren Befehle verschwendet. **Wenn es andererseits sehr wenige lange Befehle gibt, so kann der Mikroprozessorentwickler sich zu einer Standardisierung von Befehlen mit vier Taktimpulsen entscheiden, wobei er zwei vollständige Befehlausführungs-Zeiten für die einigen wenigen Befehle mit fünf Taktimpulsen verwendet.** Dies kann folgendermaßen dargestellt werden:



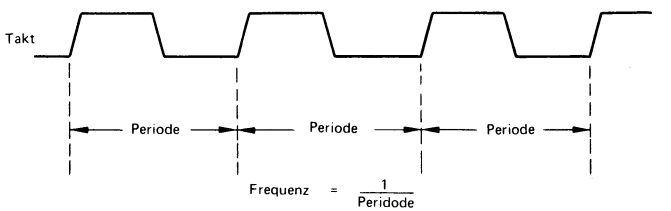
MASCHINEN-ZYKLUS

In der obigen Abbildung bilden vier Taktimpulse ein wichtiges Zeitintervall des Steuerwerks. Es ist die grundlegende Zeiteinheit für alle Befehlsausführungen. Diese Zeiteinheit wird häufig ein „Maschinenzyklus" genannt. Alle Mikroprozessoren verwenden keinen Maschinenzyklus mit einem festen Zeitintervall zur zeitlichen Steuerung der Befehlsausführungen. Aber jene, die dies tun, führen alle Befehle in irgendeiner bestimmten Anzahl von Maschinenzyklen aus – gewöhnlich in einem Maschinenzyklus, jedoch manchmal in zwei oder drei Maschinenzyklen.

Es werden jedoch alle Befehle innerhalb einer festen Anzahl von Taktimpulsen ausgeführt. **Daher ändert sich die Geschwindigkeit, mit der unser Mikrocomputer Programme ausführt, linear mit der Geschwindigkeit unseres Taktsignales.**

TAKTSIGNAL-FREQUENZ

Die Geschwindigkeit eines Taktsignales wird auch als Taktsignal-Frequenz bezeichnet. Die Taktsignal-Frequenz kann folgendermaßen dargestellt werden:



TAKTPERIODE
NANOSEKUNDE
MIKROSEKUNDE

Die Taktperiode ist die Zeit, die zwischen zwei wiederholten identischen Taktwellen erscheint. Heutzutage werden Taktperioden in Nanosekunden gemessen. Eine Nanosekunde ist gleich einem Tausendstel eines Millionstels einer Sekunde (1×10^{-9} Sekunden). **Die Periode eines Taktsignales liegt typisch zwischen hundert Nanosekunden und fünfhundert Nanosekunden.**
Einige wenige Mikroprozessoren verwenden ein Taktsignal von einer Mikrosekunde. Eine Mikrosekunde ist gleich einem Millionstel einer Sekunde.

MEGAHERTZ
MHz

Die Frequenz eines Taktsignales ist gleich dem Kehrwert der Taktperiode. Mit anderen Worten, sie ist gleich einer Sekunde, geteilt durch die Taktperiode. Wenn daher die Periode eine Mikrosekunde beträgt, so beträgt die Frequenz eine Million Impulse pro Sekunde. **Ein Taktsignal mit einer Million Impulsen pro Sekunde wird als ein Megahertz-(MHz-)Taktsignal bezeichnet.** Ein Taktsignal mit einer Periode von fünfhundert Nanosekunden hat daher zwei Millionen Impulse pro Sekunde:

$$\frac{1}{500 \times 10^{-9}} = \frac{1}{5 \times 10^{-7}} = \frac{10\ 000\ 000}{5} = 2\ 000\ 000$$

Daher wird dieses Taktsignal als ein Signal mit zwei Megahertz (2 MHz) bezeichnet. Ein Taktsignal mit einer Periode von hundert Nanosekunden wird zehn Millionen pro Sekunde bilden:

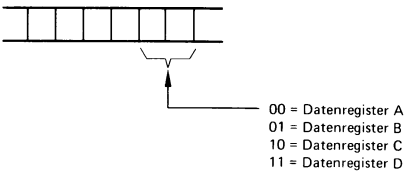
$$\frac{1}{100 \times 10^{-9}} = \frac{1}{1 \times 10^{-7}} = 10\ 000\ 000$$

Daher sagt man, daß dieses Taktsignal eine Fequenz von zehn Megahertz (10 MHz) besitzt.

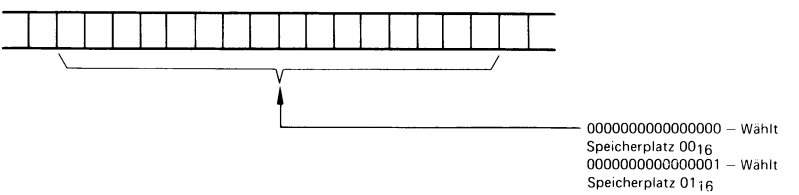
SPEICHERZUGRIFF

In einem Mikrocomputer-System kann nichts geschehen, bevor nicht ein Speicherzugriff erfolgt ist. Beispielsweise beginnt jeder Befehl mit einem Holen des Befehls, indem der Inhalt irgendeines identifizierbaren Programm-Speicherplatzes geholt wird und zwar als Daten und dann in das Befehlsregister geladen wird. Ähnlich müssen sämtliche Daten, die sich in einem Datenregister befinden, geholt werden, entweder von einer externen physikalischen Einheit oder vom Datenspeicher. Ergebnisse der ALU-Operationen in Datenregistern müssen zurück zum Datenspeicher gesendet

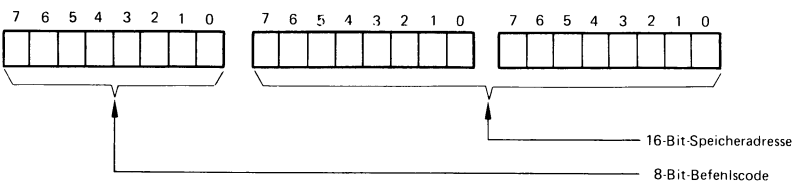
werden. **Da auch ein kleiner Speicher im allgemeinen mehr als tausend adressierbare Speicherplätze besitzt, ist das Verfahren zur Identifizierung eines einzelnen Speicherplatzes, zu dem wir zugreifen wollen, nicht unmittelbar ersichtlich.**
Wir haben bereits früher gezeigt, wie es möglich war, eines von vier Datenregistern unter Verwendung von zwei Bits des Befehlscodes zu adressieren:



Dieses einfache Verfahren zur Adressierung von Datenregistern, wenn es auf externe Speicher angewandt wird, kann bis zu 65536 Speicherplätzen zu adressieren haben. Es verwendet eine 16stellige Binärzahl zur Auswahl eines adressierbaren Speicherplatzes aus 65536 Möglichkeiten:

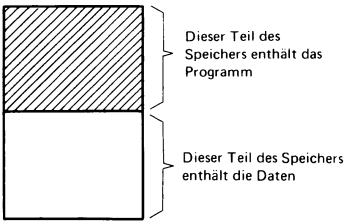


Nun gestatten viele Mikrocomputer die direkte Adressierung eines Speicherplatzes, indem sie eine 16-Bit-Speicheradresse liefern. Aber dies bedeutet, daß der Befehl durch drei Bytes von Binärdaten, und nicht einem, dargestellt wird:

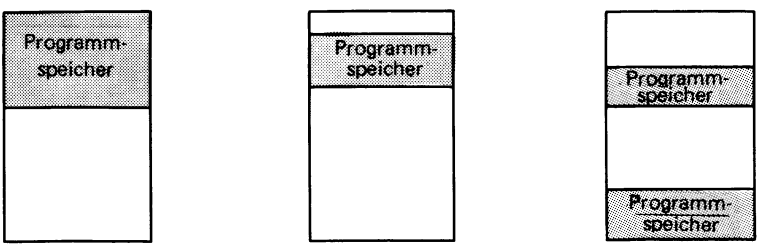


Es gibt jedoch einige Probleme, die mit dieser Lösung zur Bildung von Speicheradressen verbunden sind.
In erster Linie ist es **eine ziemliche Verschwendung von Speicherplatz.** Bei gegebener Frequenz, mit der ein Speicher-Ansprechbefehl auftritt und der Tatsache, daß die meisten Mikrocomputer nicht annähernd 65536 Speicher-Bytes besitzen, wären

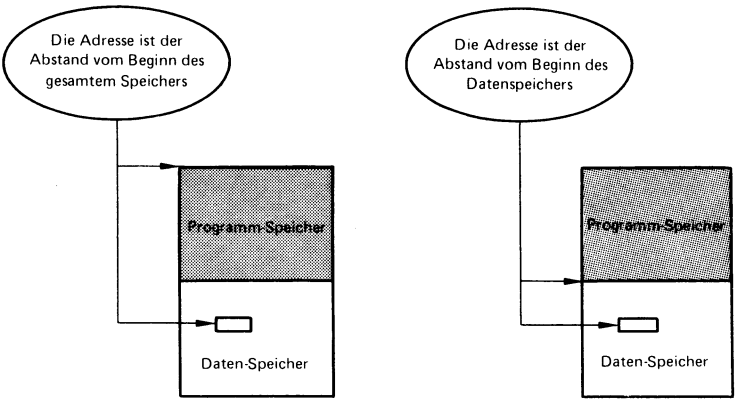
andere ökonomischere Speicher-Adressierverfahren wünschenswert. **Ferner sind in den meisten Mikrocomputer-Systemen, obwohl wir von einem „Programm"-Speicher und einem „Daten"-Speicher sprechen, diese gewöhnlich ein und dasselbe Ding:**



Wenn wir ein Programm für einen Mikrocomputer schreiben, so entscheiden wir, welcher Teil des Speichers zum Programmspeicher wird. Der Rest des Speichers ist der Datenspeicher. Wenn wir jedoch das nächste Mal ein Programm schreiben, oder wenn irgend sonst jemand ein Programm für den gleichen Mikrocomputer schreibt, so werden wir die Speicherzuordnung zwischen den Programm- und Datenbereichen völlig anders vorfinden:



Häufig werden wir einfache Programme schreiben (beispielsweise die Ausführung arithmetischer Operationen an bestimmten Zahlen). **Das Programm wird wesentlich nützlicher sein, wenn wir es in verschiedenen Anwendungen wieder verwenden können. Um dies auszuführen, benötigen wir irgendein Verfahren zur Speicher-Adressierung, das einen Datenspeicherplatz in Form seiner Versetzung (oder Abstand) vom Beginn des Datenbereiches des Speichers, anstatt vom Beginn des Speichers identifiziert:**



Sehen wir uns nun die Vorteile der Identifizierung eines Speicherplatzes durch seine Versetzung (Displacement) vom Beginn eines Datenbereiches an. Der wichtigste Vorteil ist, daß wir den Datenbereich bewegen können, ohne das Programm ändern zu müssen, das zu diesem Datenbereich zugreift. Um zu verstehen warum dies so ist, betrachten wir ein Alltagsproblem für die Identifizierung einer Zeit. Wir können die Zeit durch die Tageszeit identifizieren — was äquivalent der absoluten Speicher-Adressierung ist — oder wir können die Zeit als eine gewisse Anzahl von Stunden identifizieren, vor oder nach einem beweglichen Ereignis. Dies ist äquivalent der Adressierung eines Speichers als Versetzung vom Beginn des Datenbereiches. Nehmen wir beispielsweise an, wir hätten Befehle für die Vorbereitung eines Abendessens auszugeben. Unsere Befehle könnten folgendermaßen aussehen:

1) Schalte den Ofen um 18.00 ein.
2) Stelle den Eintopf um 18.15 in den Ofen.
3) Entferne den Eintopf um 20.15.
4) Serviere um 20.30.

Wenn wir die Zeit für das Abendessen ändern, müßten wir alle Befehle zur Vorbereitung des Abendessens ändern. Andererseits könnten wir die Befehle für die Essensvorbereitung wie folgt neu schreiben:

1) Schalte den Ofen 2 1/2 Stunden vor dem Abendessen ein.
2) 2 1/4 Stunden vor dem Abendessen gib den Eintopf in den Ofen.
3) Fünfzehn Minuten vor dem Abendessen nimm den Eintopf aus dem Ofen.
4) Serviere das Abendessen zur gewohnten Zeit.

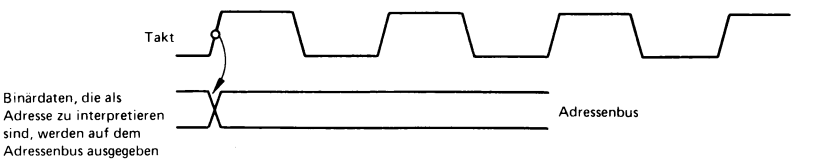
Da diese Befehle die Zeit als Versetzung von der Abendessenszeit adressieren, so bewirkt eine Änderung der Abendessenszeit keine Änderung der Vorbereitungs-Befehle für das Abendessen. Offensichtlich ist dies ein nützlicherer Weg für die Adressierung der Zeit. Er ist äquivalent der Adressierung eines Speichers über eine Versetzung vom Beginn eines Datenbereiches.

Ohne die Begriffe über die Bildung von Speicheradressen weiter zu erforschen, ist klar, daß man viele verschiedene Arten der Bildung von Speicheradressen haben kann.

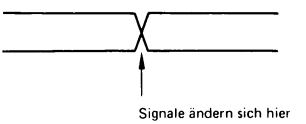
Speicheradressierung ist der Ausdruck, der allgemein für die Technik verwendet wird, in der ein Mikroprozessor uns die Identifizierung von Speicherplätzen gestattet. Sehen wir uns nun genau an, wie die Speicheradressierung arbeitet.

SPEICHERADRESSIER-LOGIK

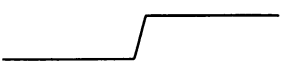
Der Speicher selbst besteht aus Logik-Bausteinen, die drei Arten von Informationen handhaben: Adressen, Daten und Steuersignale.
Jeder Speicherzugriff beginnt mit einer Adresse, die zum Speicher-Baustein gesendet wird:



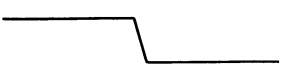
Wir erinnern uns von früheren Beschreibungen in diesem Kapitel, daß wir dies folgendermaßen darstellen:



um den Moment zu identifizieren, bei dem ein oder mehrere Signale auf einem Bus mit mehreren Leitungen ihren Pegel ändern können. Für ein einzelnes Signal können wir die tatsächliche Änderung zeigen - von niedrig auf hoch:



Oder von hoch auf niedrig:



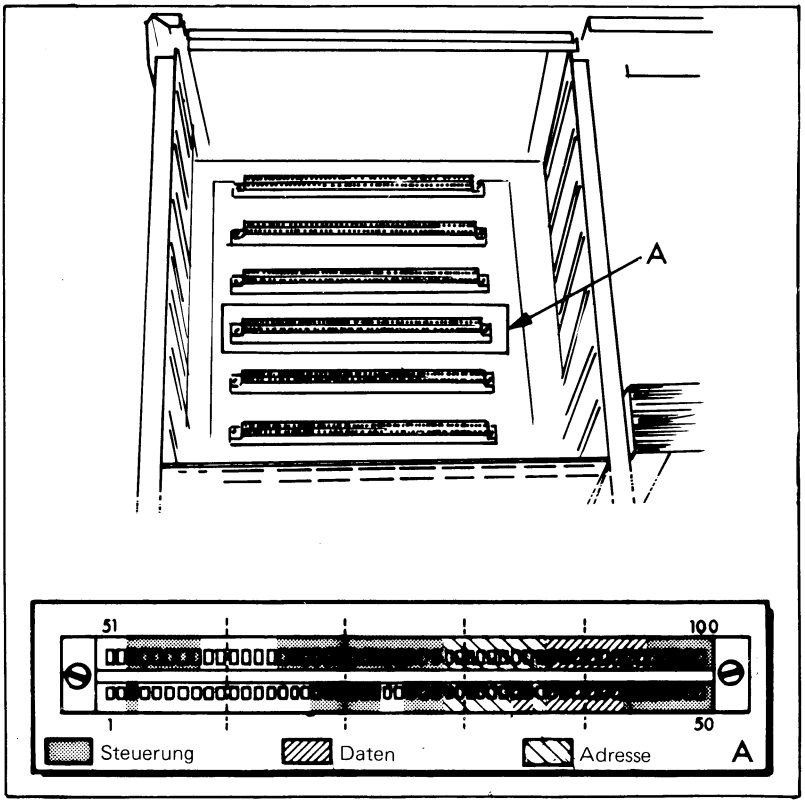


Bild 6.2 Mikrocomputer-Systembusse

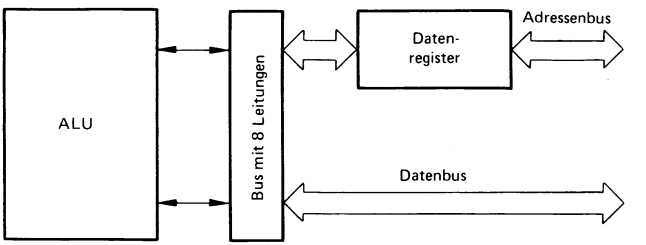
ADRESSEN-BUS

Eine Speicheradresse besteht aus binären Daten, die auf einen Bus ausgegeben werden, den wir (wie der Name sagt) den Adressen-Bus nennen. Der Adressen-Bus, wie jeder andere Bus, besteht aus einer Anzahl von parallelen Leitern, die die Zentraleinheit (CPU) mit den Speicherbausteinen des Mikrocomputer-Systems verbinden. Dies ist in Bild 6.2 dargestellt.

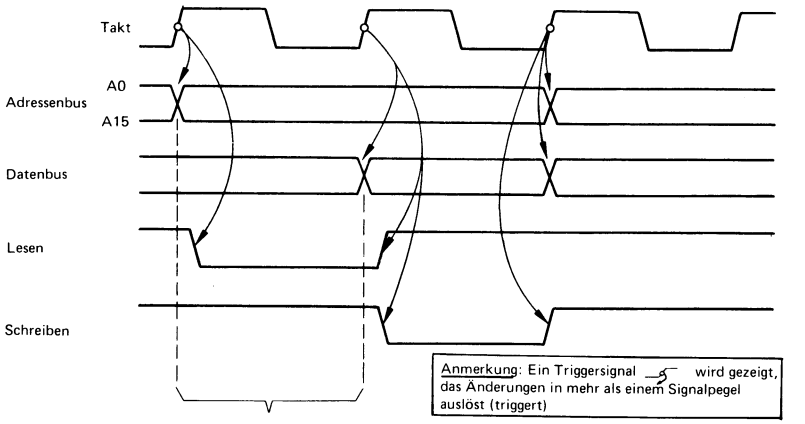
DATEN-BUS

Es gibt einen separaten Datenbus, der die Zentraleinheit mit den Speicherbausteinen verbindet. Der Datenbus ist ebenfalls in Bild 6.2 dargestellt. Sobald die Zentraleinheit (CPU) binäre Daten auf den Bus legt, gibt es keine Möglichkeiten mehr für Fehler in der Interpretation. Binäre Daten, die auf dem Adressen-Bus aufscheinen, müssen als eine Adresse interpretiert werden, während binäre Daten, die auf dem Datenbus

aufscheinen als Daten interpretiert werden müssen.
Die Abbildungen in diesem Kapitel über die zeitliche Steuerung zeigen die Adressen, die auf dem Adressenbus bei der Anstiegsflanke eines Taktimpulses aufscheinen. Die Logik innerhalb der Zentraleinheit (CPU) wird eine entsprechende ansteigende Flanke des Taktimpulses als Trigger zur Bildung von Steuersignalen verwenden, die die Adressenbus-Leitungen mit dem Datenregister innerhalb der CPU verbinden. Es ist diese Verbindung, die bewirkt, daß die Daten zu einer Adresse werden.
Die Logik innerhalb der Zentraleinheit wird wahrscheinlich imstande sein, den Datenbus und den Adressenbus mit den gleichen Datenregistern zu verbinden. Dies gestattet uns Berechnungsvorgänge an Adressen auszuführen, sowie an anderen Daten, bevor wir die Daten als eine Adresse auf dem Adressenbus ausgeben. Dies kann folgendermaßen dargestellt werden:



Der wichtigste grundsätzliche Punkt, den wir verstehen müssen, ist, daß die Logik innerhalb der CPU binäre Daten auf jede beliebige Weise interpretieren kann. Sobald jedoch die Daten auf einem Bus außerhalb der Zentraleinheit (CPU) erscheinen, bestimmt dieser Bus, innerhalb gewisser Grenzen, wie die Daten zu interpretieren sind. Beispielsweise müssen die Informationen auf dem Adressenbus eine Speicheradresse sein, aber diese Speicheradresse kann von irgendwo innerhalb der CPU kommen, und es wird nur „angenommen", daß die externe Logik sie als eine Adresse interpretiert. Wenn die externe Logik den Adressenbus in irgendeiner anderen Weise verwendet, wird die CPU-Logik nicht eingreifen, aber die externe Logik sollte besser wissen was sie tut.
Sehen wir uns nun an, wie die CPU und die Speicherbausteine über die Busleitungen miteinander verkehren.
Jeder Speicherplatz innerhalb eines Speicherbausteines wird eine zugeordnete eindeutige Adresse besitzen. Der Speicherbaustein decodiert die binären Daten auf dem Adressenbus und verwendet sie zur „Auswahl" eines einzelnen Speicherplatzes. Hierauf müssen Daten zum Speicherbaustein über den Datenbus gesendet werden, wobei sie im Falle einer Schreiboperation in den ausgewählten Speicherplatz eingeschrieben werden. Für eine Leseoperation müssen Daten vom Speicherbaustein über den Datenbus zur CPU gesendet werden. Steuersignale, die vom Steuerwerk der CPU gebildet werden, bestimmen, ob die CPU Daten aus dem Speicher „liest" oder Daten in den Speicher „schreibt". **Daher besteht die Verbindung zwischen Speicher und der CPU aus einem Adressenbus, einem Datenbus und Steuersignalen. Diese sind in Bild 6.2 dargestellt und arbeiten folgendermaßen:**

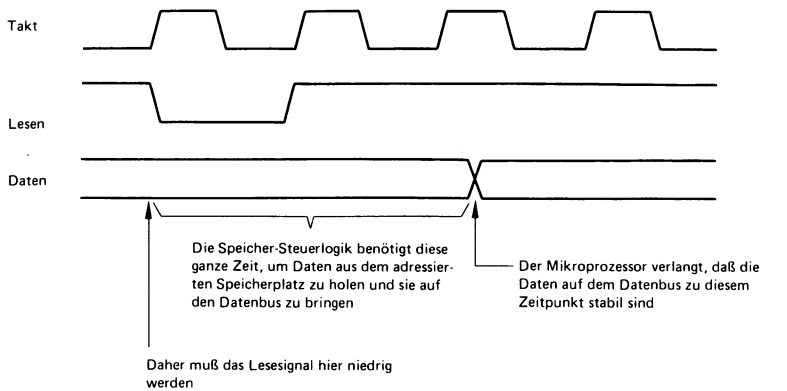


In der obigen Abbildung sind einige wichtige Konzepte enthalten.
Betrachten wir zuerst den Adressenbus. Dieser besteht, wie gezeigt, aus 16 parallelen Leitungen, was die gebräuchlichste Größe für Mikrocomputer-Adressenbusse ist. Ein Adressenbus aus 16 Leitungen kann eine 16-Bit-Adresse führen — und ist imstande bis zu 65536 verschiedene Speicherplätze zu adressieren.
Die Adresse selbst wird auf den Adressenbus bei der ansteigenden Flanke des Taktimpulses gelegt. Die Adresse wird zwei Taktimpulse lang auf dem Adressenbus festgehalten. Während dieser Zeitperiode hält die Zentraleinheit den Adressenbus mit den Datenregister-Bits verbunden, aus denen die Adresse gebildet wird. Wir bezeichnen diese Zeitperiode als die Periode, während der die Adresse auf dem Adressenbus „stabil" ist.
Nun sehen wir uns den Datenbus an. Wieder gibt es hier eine bestimmte Zeitperiode, während der die Daten auf dem Datenbus stabil sein müssen. Diese Periode wird wieder durch die Flanken des Taktsignales definiert. Der Datenbus beginnt für ein bestimmtes Zeitintervall stabil zu sein, nachdem der Adressenbus stabil ist. Dies ist notwendig, da der Logik im Speicherbaustein genügend Zeit gegeben werden muß, um die Adresse auf dem Adressenbus zu empfangen und entsprechend zu reagieren. Bis zu dem Zeitpunkt, in dem die Daten auf dem Datenbus stabil sind, muß genügend Zeit verstrichen sein, damit die adressierte Speicherstelle ausgewählt ist.

SPEICHER-LESE-STEUERUNG

Wenn eine Speicher-Leseoperation auftritt, dann wird das Lese-Steuersignal aktiviert. Hier ist willkürlich ein niedriger Impuls gezeigt. Es könnte jedoch ebensogut ein hoher Impuls sein. In jedem Fall ist es wichtig, daß das Steuersignal einen „passiven" Status besitzt, während dem nichts geschieht, und einen „aktiven" Zustand, während dem es anzeigt, daß etwas zu geschehen hat — in diesem Fall, daß eine Lese-Operation auftritt.

Eine Lese-Operation erfordert, daß der Inhalt des adressierten Speicherplatzes auf den Datenbus gebracht wird. Wieder ist eine bestimmte Zeit für die Logik im Speicherbaustein erforderlich, den niedrigen Lese-Impuls festzustellen und auf ihn zu reagieren, indem Daten vom adressierten Speicherplatz auf den Datenbus gegeben werden. Der niedrige Lese-Impuls muß daher früh genug auftreten, damit der Speicherlogik genügend Zeit zum Transfer von Daten vom gewählten Speicherplatz auf den Datenbus zur Verfügung steht:

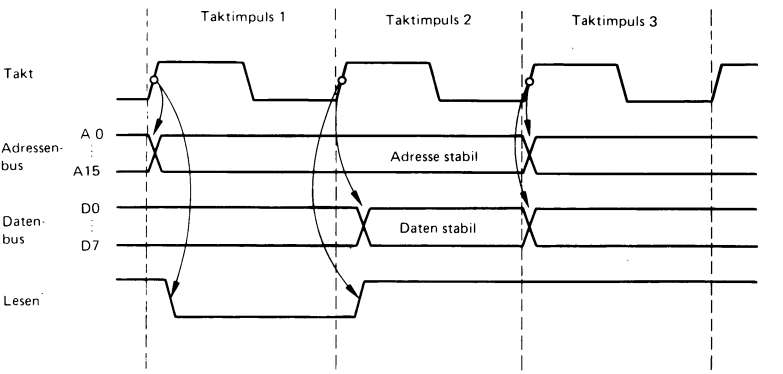


SCHREIB-STEUERSIGNAL

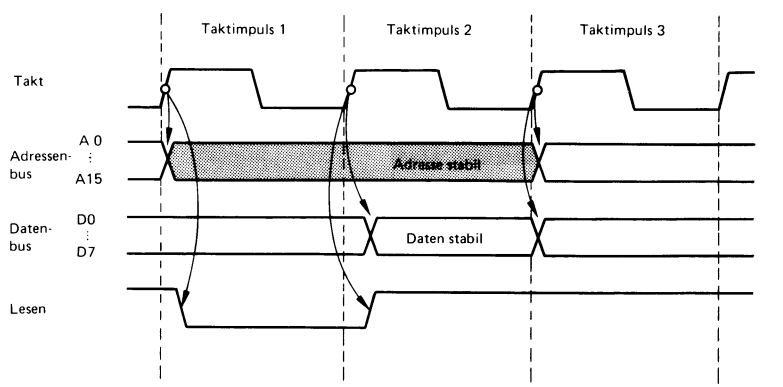
Das Schreib-Steuersignal, das wiederum als ein niedriges Impuls-Signal gezeigt wird, zeigt an, daß Daten auf den Datenbus in den adressierten Speicherplatz zu schreiben sind. Das Schreib-Steuersignal arbeitet als ein Austastsignal. Es kann nicht als „wahr" (ein niedriger Impuls ist „wahr") erscheinen, bis gültige Daten auf dem Datenbus stabil sind. Der Grund hierfür ist, daß der Speicherbaustein den niedrigen Schreib-Austastimpuls zur Verbindung des Datenbusses zum adressierten Speicherplatz verwendet. Würde der niedrige Schreib-Austastimpuls erscheinen, bevor die Daten auf dem Datenbus stabil sind, würden fehlerhafte Daten in den Speicherbaustein eingeschrieben.

SPEICHER-LESE-OPERATION

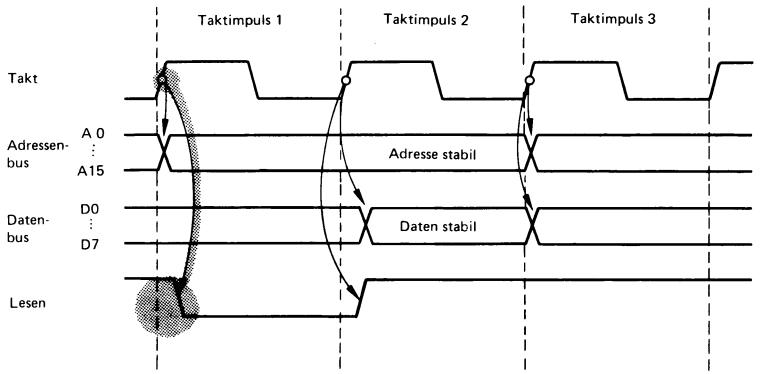
Wir wollen uns Speicherlese- und Schreiboperationen getrennt ansehen. Wir sehen hier als erstes die zeitliche Steuerung, die bei einer Lese-Operation auftritt:



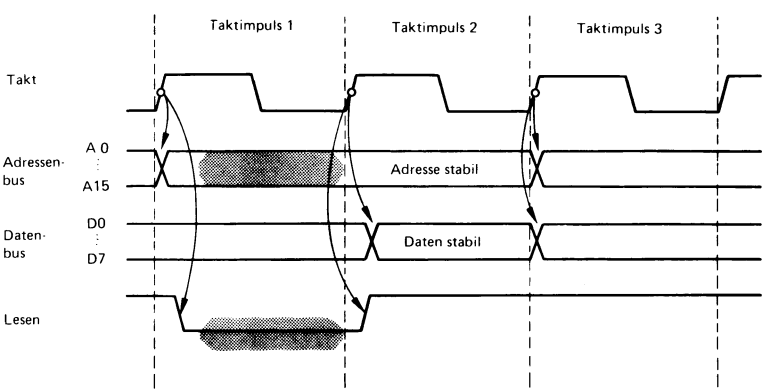
Die Speicher-Leseoperation beginnt an der ansteigenden Flanke des Taktimpulses 1. Die Zentraleinheit verwendet diesen Taktimpuls zur Verbindung der entsprechenden Datenregister mit dem Adressenbus und dies leitet das Auftreten einer stabilen Adresse auf dem Adressenbus ein:



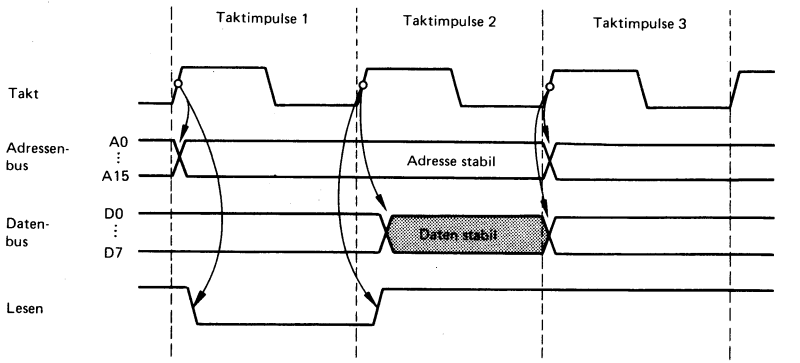
Ganz kurz danach wird das Lese-Steuersignal niedrig:



Die Speicherlogik decodiert den Inhalt des Adressenbusses, um einen einzelnen Speicherplatz als „ausgewählt" zu identifizieren. Diese Speicherplatz-Auswahllogik ist nichts weiter als eine Kombination von Boole'schen Operationen, die an den individuellen Signalen des Adressenbusses ausgeführt werden. Es ist jedoch die tatsächliche Methode, die von einem Speicherbaustein zur Decodierung des Inhalts des Adressenbusses verwendet wird, nicht wichtig. So lange wir uns darüber im klaren sind, daß eine derartige Logik existiert, die einen einzelnen Speicherplatz aus 65536 Möglichkeiten auswählt, ist dies alles, was von Bedeutung ist. Das zugehörige Lese-Steuersignal veranlaßt den Speicherplatz, die Bits des gewählten Speicherplatzes mit dem Datenbus zu verbinden. Der Vorgang der Identifizierung eines einzelnen Speicherplatzes und der anschließenden Verbindung mit dem Datenbus, muß vor der Anstiegsflanke des Taktimpulses 2 auftreten:



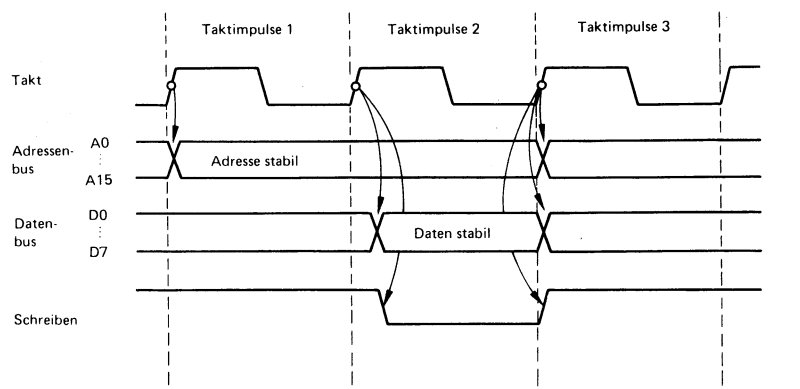
Zu dieser Zeit muß der Inhalt des ausgewählten Speicherplatzes auf dem Datenbus stabil sein. Die Daten müssen auf dem Datenbus während einer bestimmten Zeit stabil gehalten werden. Diese Zeit muß der Zentraleinheit zur Verfügung gestellt werden, um die Daten vom Datenbus zu nehmen und sie in das zugehörige Datenregister zu laden:



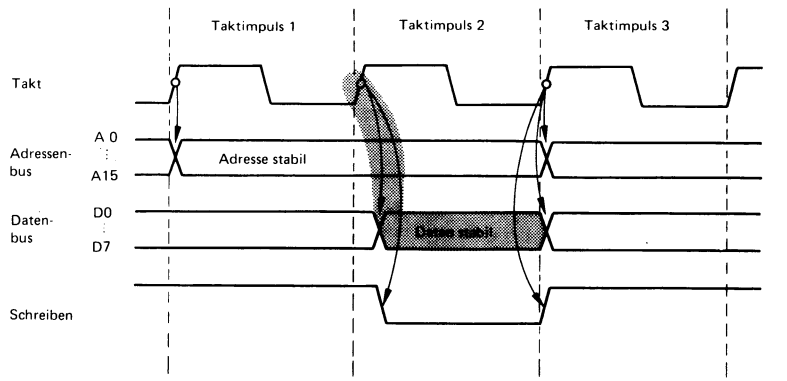
Die CPU verbindet das entsprechende Datenregister mit dem Datenbus und daher ist der Lesevorgang beendet.
In der obigen Abbildung ist ein Taktimpuls als die Zeitdauer gezeigt, während der die Daten auf dem Datenbus stabil gehalten werden.

SPEICHER-SCHREIB-OPERATION

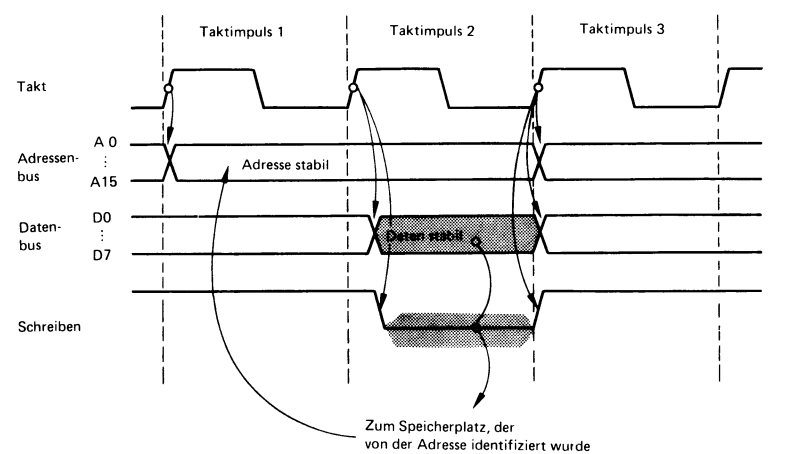
Nun betrachten wir eine Speicher-Schreiboperation. Ihre zeitliche Steuerung kann folgendermaßen dargestellt werden:



Die zeitliche Steuerung einer Speicher-Schreiboperation unterscheidet sich nicht wesentlich von der zeitlichen Steuerung einer Speicher-Leseoperation. Die Logik, die eine stabile Adresse auf den Adressenbus ausgibt, ist in der Tat identisch für die beiden Operationen. Unterschiede scheinen nur auf dem Datenbus und in den Steuersignalen auf. Wenn die Adresse auf dem Adressenbus aufscheint, gibt es kein zugehöriges niedriges Schreib-Steuersignal. Daher ist der ausgewählte Speicherplatz nicht mit dem Datenbus verbunden und sein Inhalt wird nicht auf den Datenbus ausgegeben. Im Anschluß hieran verbindet die Zentraleinheit ein geeignetes Datenregister mit dem Datenbus um stabile Daten auszugeben:



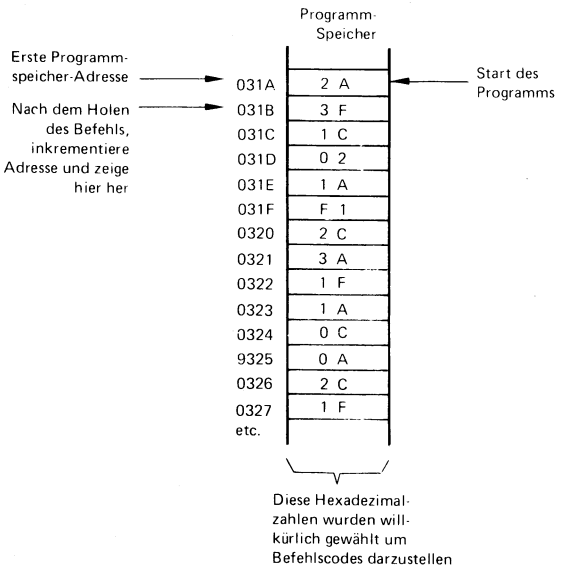
In diesem Moment wird das Schreib-Steuersignal niedrig gemacht. Das Schreib-Steuersignal wird vom Speicherbaustein empfangen und zur Verbindung des ausgewählten Speicherplatzes mit dem Datenbus verwendet, so daß der Inhalt des Datenbusses in den ausgewählten Speicherplatz eingeschrieben und dort ständig gespeichert wird:



ADRESSIERUNG DES PROGRAMMSPEICHERS UND DER BEFEHLSZÄHLER

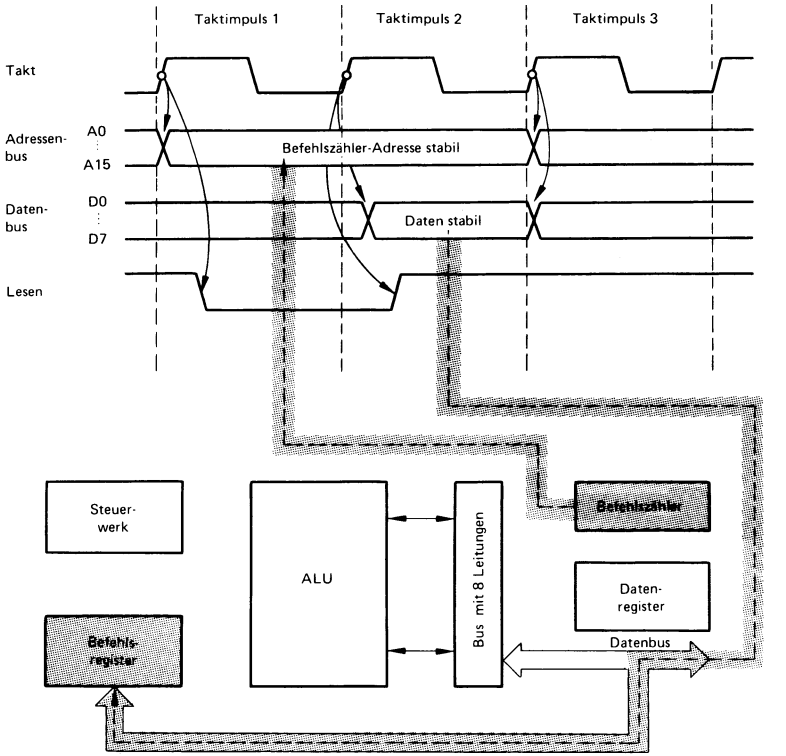
Sehen wir uns nun die CPU-Register an, die zur Bildung von Speicheradressen erforderlich sind. Wir wollen mit der Betrachtung der Programmspeicher-Adressierung beginnen.

Die für die Adressierung des Programmspeichers erforderliche Logik ist sehr einfach. Ein Programm ist nichts weiter als eine Folge von Befehlscodes, die normalerweise in Speicherplätzen mit ansteigenden Adressen aufbewahrt sind:



BEFEHLS-ZÄHLER

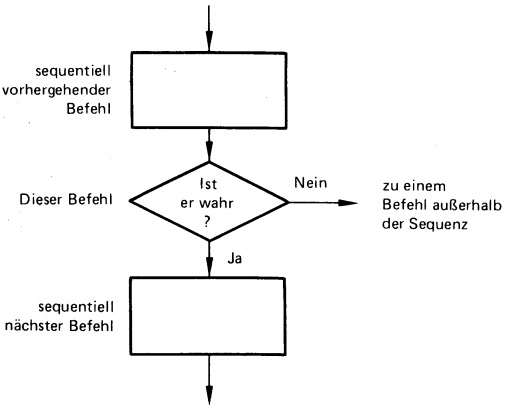
Wenn dies der Fall ist, so brauchen wir nur die Adresse des ersten Befehls innerhalb dieser Folge (oder Sequenz) zu identifizieren. Nach jedem Holen eines Befehls, wenn wir diese Adresse erhöhen (oder inkrementieren), wird diese genau zum nächsten Befehl in der Folge zeigen. Die Adressier-Logik für den Programmspeicher wird von einem Register gehandhabt, das als Befehlszähler (oder Programmschritt-Zähler = Program Counter) bezeichnet wird. Der Befehlszähler enthält binäre Daten, die als eine Speicheradresse interpretiert werden, und zwar nur weil die binären Daten auf den Adressenbus ausgegeben werden. Daher liefert der Befehlszähler die Speicheradresse für alle Befehl-Holoperationen. Für einen externen Speicher sieht ein Befehl-Holen genau wie irgendeine Speicher-Leseoperation aus. Dies kann wie folgt gezeigt werden:



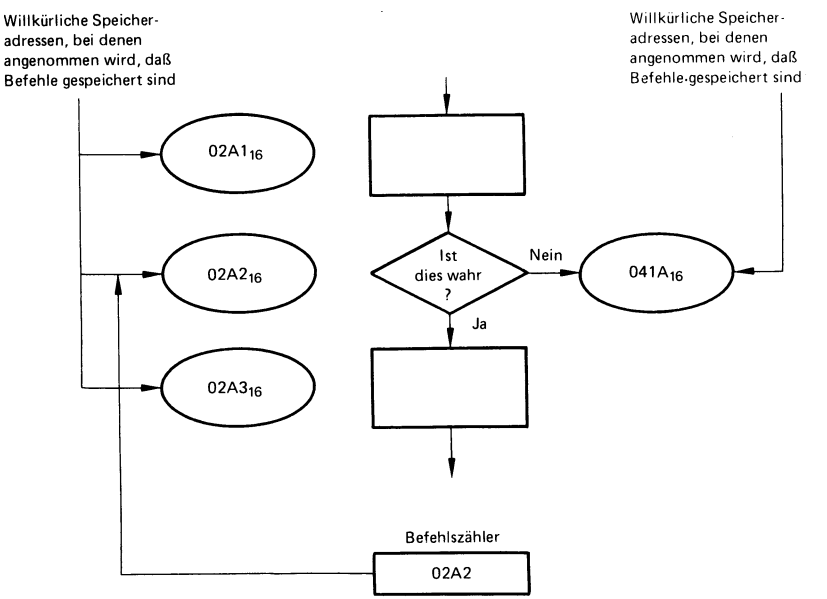
PROGRAMMLOGIK UND DER BEFEHLSZÄHLER

Der Befehlszähler wird innerhalb der Zentraleinheit sowohl als ein Register zur Erzeugung von Adressen als auch ein Datenregister behandelt. Es ist sehr wichtig, daß der Befehlszähler als Datenregister zugänglich ist, da dies die Basis für die Programmlogik darstellt.

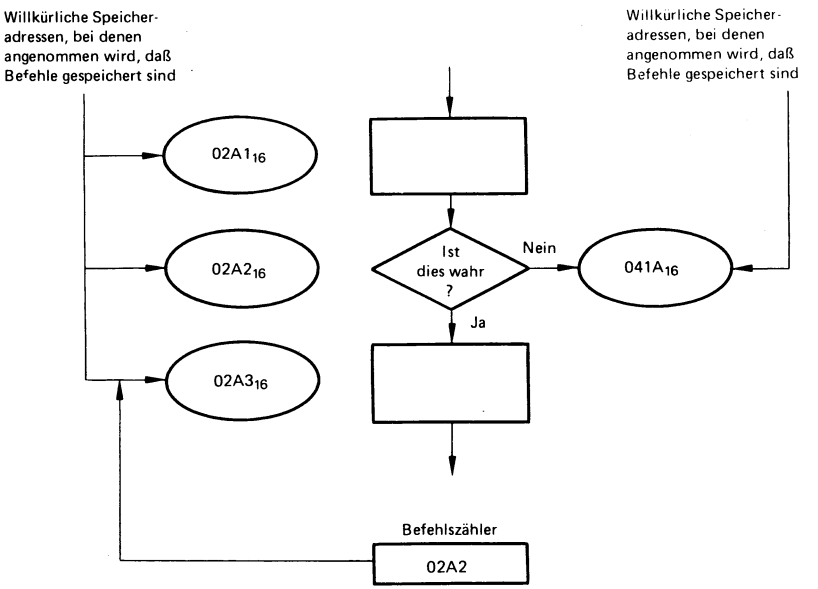
Wir haben bereits in Programm-Flußdiagrammen Schritte zur Entscheidungsfindung gesehen, die dem Programm die Fortsetzung durch die Ausführung des nächsten sequentiellen Befehls gestattet oder Befehle außerhalb dieser Folge:



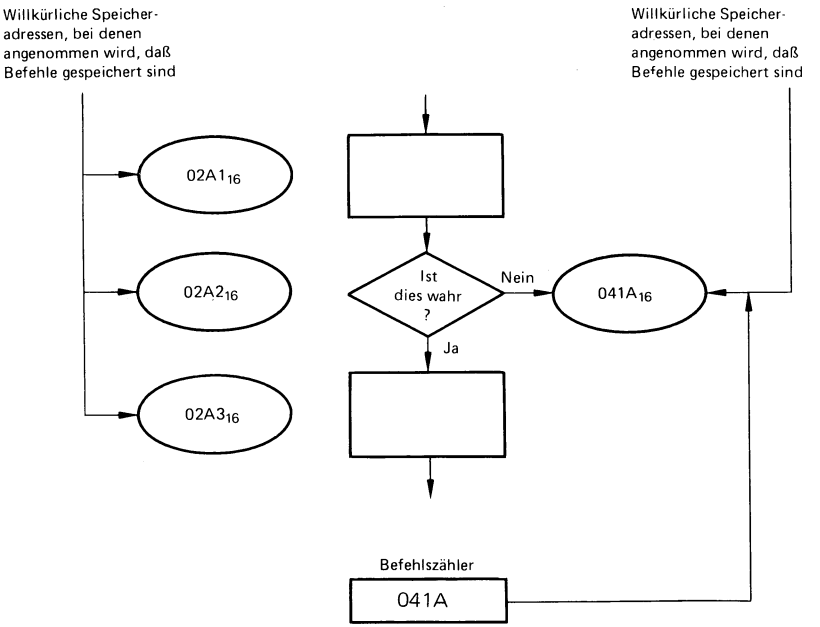
Wie identifizieren und holen wir nun einen Befehl außerhalb der Sequenz? Die Antwort ist, indem wir im voraus die Adresse dieses Befehls außerhalb der Sequenz wissen. Wenn wir diese bekannte Adresse, als Daten, in den Befehlszähler laden, so führen wir einen Sprungbefehl aus. Wenn der Sprungbefehl ausgeführt ist, so zeigt der Befehlszähler zum Sprungbefehl selbst:



Nachdem der Sprungbefehl geholt wurde, läßt man den Befehlszähler zum nächsten sequentiellen Befehl zeigen:



Wenn der Sprung auftreten soll, wird die Zentraleinheit den Sprung ausführen lassen, indem sie die Adresse des Befehls außerhalb der Sequenz, als Daten, in den Befehlszähler lädt. Diese Daten werden in den Befehlszähler während der Befehls-Ausführungsphase des Sprungbefehls geladen:



Wenn nun der Sprungbefehl seine Ausführung beendet hat, und die Ausführung des nächsten Befehls beginnt, wird ein Befehlholen auftreten. Aber anstatt, daß man den nächsten sequentiellen Befehl erhält, bekommt man den neuen Befehl außerhalb der Sequenz.

ADRESSIER-REGISTER FÜR DATENSPEICHER

Wie wir früher in diesem Kapitel gesehen haben, ist die Adressierung von Datenspeichern nicht so einfach wie die Adressierung des Programmspeichers, da hier keine „übliche" Sequenz vorliegt, in der Daten gespeichert werden. Daher werden eine Vielzahl von ausgeklügelten Verfahren zur Bildung von Adressen von Datenspeichern verwendet.
Die Berechnung der Daten-Speicheradressen wird zu einer einfachen Erweiterung der Logik auf der Datenseite der arithmetischen und logischen Einheit (ALU). Wenn wir Datenregister zur Speicherung binärer Daten, sowie eine arithmetische und logische Einheit besitzen, in der wir Berechnungen ausführen können, haben wir alle Voraussetzungen zur Berechnung binärer Daten, die in der Folge zu Daten-Speicheradressen

werden. Daher ist die Berechnung von Daten-Speicheradressen durch die bereits in der Zentraleinheit vorhandenen Logik einfach zu handhaben.
In diesem Buch werden wir nicht die verschiedenen Methoden besprechen, die zur Bildung von Daten-Speicheradressen verwendet werden. Diese Informationen benötigen wir nur dann, wenn wir beginnen, das Schreiben von Programmen in Assemblersprache zu lernen, und dies ist ein Thema für die „Einführung in die Mikrocomputer-Technik". Eine Daten-Speicheradresse ist das Ergebnis von Datenberechnungen, die einem Datenzugriff vorausgehen. Das ist alles was wir für den Moment wissen müssen.

ADRESSIERUNG EXTERNER LOGIK

Um irgendwelche externe Logik oder periphere Geräte (wie eine Tastatur oder eine Videoanzeige) zu adressieren, wird die Zentraleinheit (CPU) eine Folge von Schritten durchlaufen, die sehr ähnlich jenen sind, die wir für den Zugriff zu einem Speicher benötigen. Die externen Logik-Bausteine, auf die zugegriffen werden soll, haben eine spezifische Adresse, so wie jeder Speicherplatz eine spezifische Adresse besitzt. **Daher besteht der Zugriff auf externe Geräte aus diesen Schritten:**

1) Bilde eine Eingabe-/Ausgabe-Bausteinadresse. Diese Adresse ist eine Binärzahl, die als Daten gebildet und dann über einen Adressenbus ausgegeben werden.
2) Triggere geeignete Steuersignale, die dem E/A-Baustein sagen, was er zu tun hat.
3) Sende oder empfange Daten über einen Datenbus.

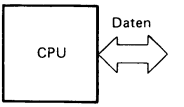
Die Folge der Vorgänge, die für den Zugriff zu einer externen Logik oder E/A-Bausteinen erforderlich ist, ist so ähnlich mit einem Speicherzugriff, daß viele Mikroprozessoren diese beiden als ein und dasselbe Ding behandeln. In diesem Fall reagieren E/A-Bausteine und externe Logik auf genau die gleichen Signale wie ein Speicherbaustein. Und das einzige, das diese beiden unterscheidet, sind die Speicheradressen, die für jedes reserviert sind.
Mikroprozessoren, die Speicher und E/A-Bausteine getrennt behandeln, haben in Wirklichkeit einen sekundären Satz von Logik, die im wesentlichen die Speicher-Adressierlogik verdoppelt — jedoch innerhalb eines kleinen Bereiches. Wenn beispielsweise der Adressenbus für Speicher gewöhnlich aus 16 Leitungen besteht, die imstande sind 65536 individuelle Speicherplätze zu adressieren, so könnte der E/A-Adressenbus nur 8 Leitungen breit sein, was bedeutet, daß 256 verschiedene E/A-Bausteine adressiert werden könnten.

BEFEHLSVORRAT UND PROGRAMMIERUNG

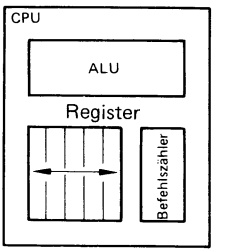
Die Befehle eines Mikroprozessors, wie wir uns erinnern, identifizieren die individuellen Operationen, die als einzelne Einheiten durch die Logik innerhalb des Mikroprozessors ausgeführt werden können. Es gibt fünf Arten von Vorgängen, die durch individuelle Befehle spezifiziert werden können. Sie werden wie folgt dargestellt:

Mikroprozessor-Befehle

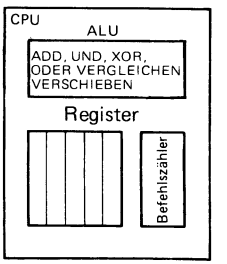
① Transfer von Daten zwischen dem Mikroprozessor und der Logik um den Mikroprozessor



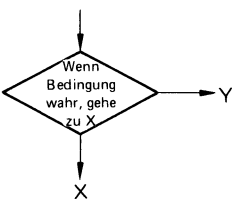
② Bewegung von Daten von einem Register zu einem anderen innerhalb des Mikroprozessors



③ Spezifizierung einer Operation der arithmetischen und logischen Einheit



④ Modifizieren des Inhalts des Befehlszählers und daher Freigabe der programmierten Logik



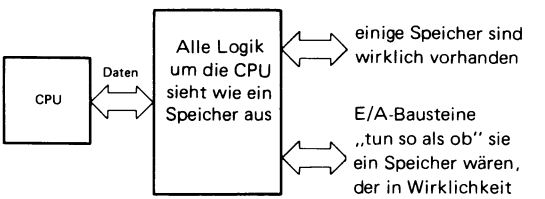
⑤ Manipulation der „Bedingung" des Statusflags in Type④ ist ein Beispiel des Status

Wir wollen diese Befehlstypen nacheinander einzeln betrachten.
Befehle, die Daten zwischen dem Mikroprozessor und der den Mikroprozessor umgebenden Logik bewegen, können auf den Speicher oder physikalische Bausteine zugreifen. Physikalische Bausteine außerhalb des Mikrocomputer-Systems werden im allge-

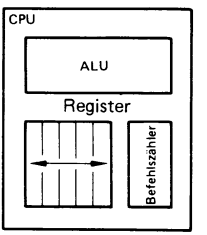
meinen als Eingabe-/Ausgabe-(oder E/A-)Bausteine oder Geräte bezeichnet. Befehle, die auf E/A-Bausteine zugreifen, werden auch E/A-Befehle genannt. **Manche Mikroprozessoren besitzen getrennte Befehle, um Daten zwischen dem Mikroprozessor und E/A-Bausteinen oder dem Speicher zu transferieren:**

Mikroprozessor-Befehle

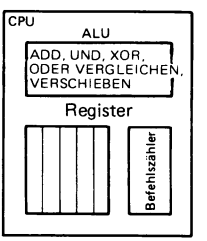
① Transfer von Daten zwischen dem Mikroprozessor und der Logik um den Mikroprozessor



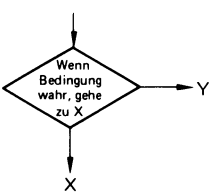
② Bewegung von Daten von einem Register zu einem anderen innerhalb des Mikroprozessors



③ Spezifizierung einer Operation der arithmetischen und logischen Einheit



④ Modifizieren des Inhalts des Befehlszählers und daher Freigabe der programmierten Logik

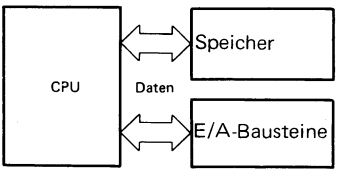


⑤ Manipulation der „Bedingung" des Statusflags in Type④ist ein Beispiel des Status

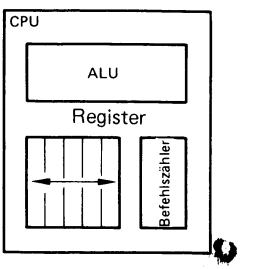
Andere Mikroprozessoren verwenden die gleichen Befehle zum Transfer von Daten zwischen dem Mikroprozessor und dem Speicher oder E/A-Bausteinen:

Mikroprozessor-Befehle

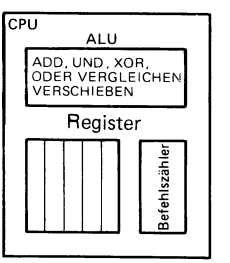
① Transfer von Daten zwischen dem Mikroprozessor und der Logik um den Mikroprozessor



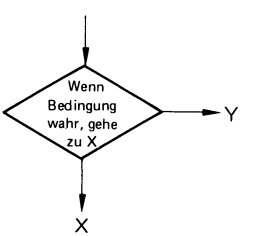
② Bewegung von Daten von einem Register zu einem anderen innerhalb des Mikroprozessors



③ Spezifizierung einer Operation der arithmetischen und logischen Einheit

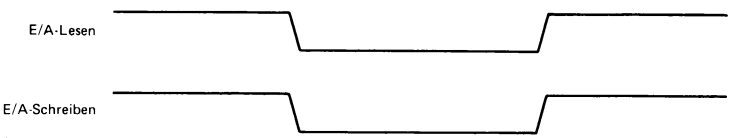


④ Modifizieren des Inhalts des Befehlszählers und daher Freigabe der programmierten Logik

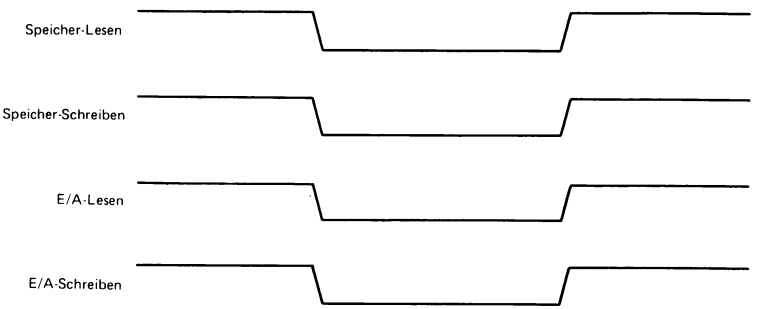


⑤ Manipulation der „Bedingung" des Statusflags in Type ④ ist ein Beispiel des Status

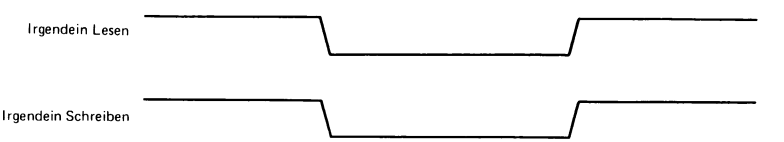
Wir werden uns von früheren Betrachtungen der Signale und Busse erinnern, daß hier tatsächlich ein sehr kleiner Unterschied zwischen den E/A- und Speicher-Ansprechbefehlen besteht. E/A-Ansprechbefehle erzeugen Steuersignale, die den Transfer von Daten zu oder von einem E/A-Baustein identifizieren:



Speicher-Ansprechbefehle erzeugen äquivalente Signale:

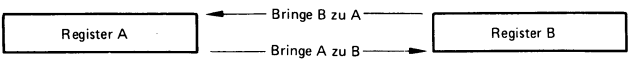


Ein Mikroprozessor, der die gleichen Befehle für den Zugriff zu Speichern oder einem E/A-Baustein verwendet, wird einfach einen Satz von Steuersignalen besitzen:

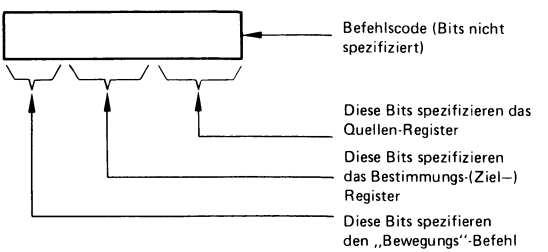


Wenn ein Mikroprozessor die gleichen Steuersignale für den Zugriff zu Speichern oder E/A-Bausteinen verwendet, so unterscheidet die mit der Speicheradressierung ver-

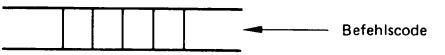
bundene Logik zwischen Speicher oder einem E/A-Baustein. Wer immer den Mikrocomputer entwickelt, entscheidet, welche Adressen in der Tat auf Speicher zugreifen und welche Adressen statt dessen einen E/A-Baustein auswählen.
Es ist zu beachten, daß wohl ein Mikroprozessor getrennte Speicher- und E/A-Befehle besitzt, ein Mikrocomputer-Entwickler die E/A-Befehle nicht verwenden muß. Der Mikrocomputer-Entwickler könnte trotzdem E/A-Bausteine adressieren, als ob sie Speicherplätze wären. Das Umgekehrte ist nicht richtig. Ein Mikroprozessor, der keine E/A-Befehle hat, zwingt den Mikrocomputer-Entwickler die E/A-Bausteine als Speicherplätze zu adressieren. In jedem Fall, haben wir als Anwender eines Mikrocomputers keine Wahl. Wir müssen Speicheransprech- und E/A-Ansprechbefehle genau verwenden, wie sie der Mikrocomputer-Entwickler uns angibt.
Die Anzahl und die Komplexität von Befehlen, die Daten zwischen den CPU-Registern bewegen, ist exakt eine Funktion der Anzahl der im Mikroprozessor vorhandenen Registern. Offensichtlich besitzt ein Mikroprozessor, der nur ein adressierbares Register hat, keine Befehle zur Bewegung von Daten zwischen Registern. Ein Mikroprozessor, der zwei adressierbare Register besitzt, könnte zwei derartige Befehle haben:



Wenn die Anzahl von Registern innerhalb der CPU ansteigt, so werden dem Mikroprozessor bald die Möglichkeiten ausgehen. Nehmen wir beispielsweise an, ein Mikroprozessor habe 16 adressierbare Register. Um Daten von jedem beliebigen der 16 Register (als Quelle) zu jedem beliebigen anderen der 16 Register (als Bestimmungsort) zu bringen, muß der Befehlscode des Mikroprozessors einige Bits besitzen, mit denen er jedes mögliche Quellenregister und jedes mögliche Bestimmungsregister identifizieren kann:

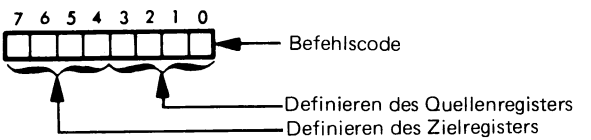


Aber man benötigt vier Befehlscode-Bits zur Identifizierung eines von 16 Registern:



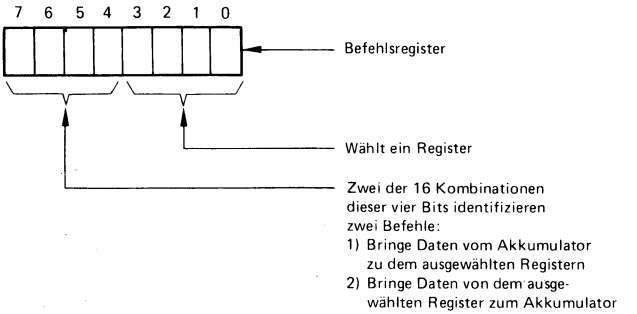
0000 Register 0
0001 Register 1
0010 Register 2
0011 Register 3
0100 Register 4
0101 Register 5
0110 Register 6
0111 Register 7
1000 Register 8
1001 Register 9
1010 Register 10
1011 Register 11
1100 Register 12
1101 Register 13
1110 Register 14
1111 Register 15

Wenn vier Befehlscode-Bits zur Spezifizierung des Quellenregisters erforderlich sind, und weitere vier Befehlscode-Bits zur Spezifizierung des Bestimmungs-(oder Ziel-)Registers erforderlich sind, dann werden für einen 8-Bit-Mikroprozessor alle acht Befehlscode-Bits einfach aufgebraucht, nur zur Spezifizierung von Befehlen, die Daten zwischen einem Quellenregister und einem Zielregister bewegen:

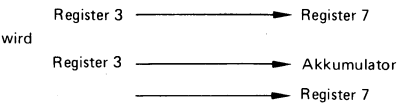


kein Code übrig!!

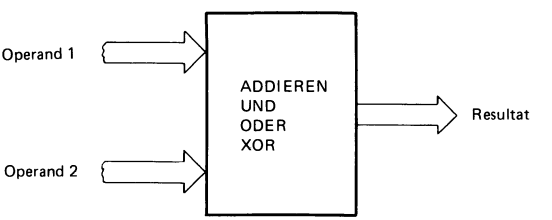
Eine von vielen Mikroprozessor-Entwicklern verwendete Lösung besteht darin, ein sogenanntes „primäres" Register, auch oft als Akkumulator bezeichnet, zu verwenden. Der Akkumulator muß die Quelle oder der Bestimmungsort für jede Datenbewegung innerhalb der CPU sein. Nun müssen Befehle, die Daten von einem Platz zu einem anderen innerhalb der CPU bewegen, immer Daten durch den Akkumulator bewegen. Für den Fall der 16 Register benötigen wir nun gerade vier Bits für die Definition des ausgewählten Registers:



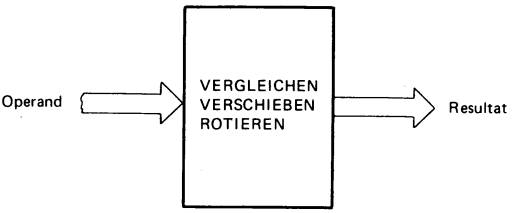
Wir sehen, daß wir noch Daten von jedem Register zu jedem anderen bewegen können, aber diese Operation erfordert nun zwei Befehle anstatt einem. Dies kann wie folgt gezeigt werden:



Sehen wir uns nun Befehle an, die Operationen der arithmetischen und logischen Einheit spezifizieren. Allgemein gibt es zwei Klassen von arithmetischen und logischen Operationen: Solche, die zwei Operanden benötigen, und solche, die nur einen Operanden erfordern. Die Addition, UND-, ODER- und Exklusiv-ODER-Operationen erfordern zwei Operanden.



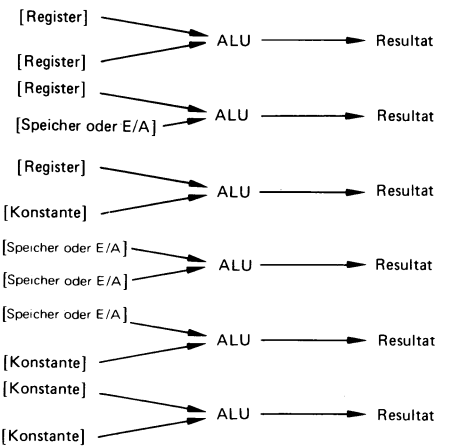
KOMPLEMENTIER- SCHIEBE- und ROTATIONS-Operationen benötigen nur einen Operanden:



Operanden können von einem von drei Plätzen kommen:

1) Von einem Register innerhalb des Mikroprozessors.
2) Von einem externen Speicher oder E/A-Platz.
3) Als eine Konstante, die vom Befehl selbst geliefert wird.

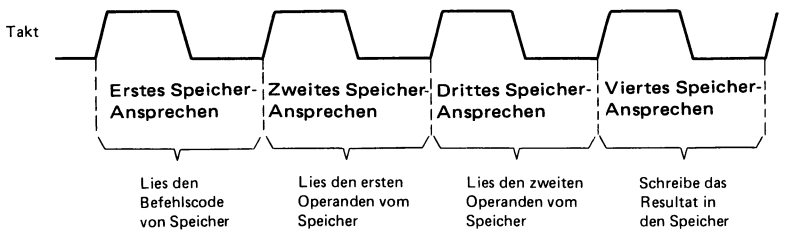
Betrachten wir zuerst Befehle für zwei Operanden. Wir haben sechs Möglichkeiten, nur für die Operanden. Wenn wir die Bezeichnung [ ] zur Kennzeichnung von „Inhalt von" verwenden, so können diese Möglichkeiten wie folgt dargestellt werden:



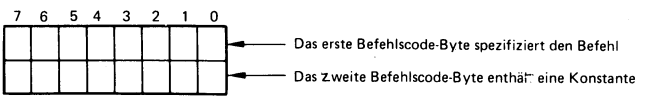
Wir wollen nun willkürlich die Additionsoperation wählen und die Möglichkeiten überprüfen.

Wir wollen etwa den Inhalt eines Registers zu dem eines anderen addieren. Die Summe könnte in eines der zwei Operanden-Register gespeichert werden oder es könnte

in ein drittes Register gebracht werden, in einen externen Speicher oder E/A Bausteinplatz untergebracht werden. Die Speicherung des Ergebnisses in eine „Konstante" würde ohne Bedeutung sein, da es diese Konstante zerstören würde.
Wir könnten den Inhalt eines Registers und eines externen Speichers oder E/A-Platzes addieren. Wieder könnte die Summe in einen der Operandenplätze gespeichert werden, oder in irgendeinen getrennten Platz.
Ein Befehl könnte auch spezifizieren, daß der Inhalt zweier externer Speicher- oder E/A-Plätze addiert wird und die Summe in einem Register gespeichert wird, in einen der Operandenplätze oder in einen anderen externen Speicher- oder E/A-Platz. In der Praxis gibt es wenige Mikroprozessoren, die Befehle haben, mit denen zwei Operanden von zwei externen Speicherplätzen geholt werden können. Der Grund hierfür ist, daß der Mikroprozessor einen Lesevorgang für einen externen Speicher für jeden Operanden ausführen muß, und das ergibt relativ komplexe Befehle. Beispielsweise würde es vier Speicher-Ansprechvorgänge brauchen, die innerhalb einer Befehlsausführung auftreten, um den Inhalt zweier externer Speicherplätze zu addieren. Dies könnte folgendermaßen gezeigt werden:



Es sind natürlich auch derartig komplexe Befehle, wie oben gezeigt möglich, und in der Tat haben viele Minicomputer derartige Befehle. Mikroprozessoren sind im allgemeinen einfacher als Minicomputer. Typische Mikroprozessoren sind auf einen Befehls-Holvorgang begrenzt und einen zusätzlichen Speicher-Ansprechvorgang während einer einzelnen Befehlsausführung. Daher kann ein Operand vom Speicher (oder einem E/A-Baustein) kommen oder das Resultat kann in einen Speicher- oder einen E/A-Baustein gespeichert werden, jedoch nicht beide.
Ein Operand kann auch eine Konstante sein. Die Konstante wird durch den Befehlscode geliefert. Ein Befehl könnte beispielsweise spezifizieren, daß der konstante Wert Drei zum Inhalt eines CPU-Registers oder eines externen Speicherplatzes addiert wird. Ein Befehl, der eine Konstante spezifiziert, enthält eigentlich die Konstante als Teil des Befehls. Dies kann wie folgt dargestellt werden:



In Wirklichkeit ist eine Konstante nichts anderes als der Inhalt eines Speicherplatzes im Programmspeicher. Aber da der Programmspeicher häufig ein Festwertspeicher ist, wird die Konstante in der Tat eine Konstante – da sie in einem Teil eines Speichers liegt, der nicht geändert werden kann.

ALU-Operationen, die einen einzelnen Operanden benötigen, können den Inhalt eines Registers, Speicherplatzes oder E/A-Bausteines als den Operanden spezifizieren. Mikroprozessor-Befehle gestatten nicht, daß eine Konstante als Eingabe für einen ALU-Befehl mit einem einzelnen Operanden spezifiziert wird, da dies keinen Sinn haben würde. Beispielsweise würde ein Befehl zur Komplementierung von Drei ein unnötiger Befehl sein. Wir wissen was das Komplement von Drei ist und können daher diesen Wert ebenfalls speichern. **Befehle, die den Inhalt des Befehlszählers ändern, fallen in eine von drei Kategorien. Diese sind:**

1) Befehle, die ohne Bedingungen den Inhalt des Befehlszählers ändern.
2) Befehle, die den Inhalt des Befehlszählers nur dann ändern, wenn spezielle Bedingungen erfüllt werden, die durch einen zugehörigen Status identifiziert werden.
3) Befehle, die den Inhalt des Befehlszählers retten (oder aufbewahren), bevor sie ihn modifizieren. Diese Befehle geben uns die Möglichkeit zu dem Punkt zurückzukehren, in dem der Inhalt des Befehlszählers geändert wurde. Wir können zu diesem Punkt zurückkehren, indem wir den aufbewahrten (geretteten) Wert in den Befehlszähler zurückspeichern.

Während Befehle, die Daten bewegen und die ALU-Logik manipulieren, leicht zu verstehen sind, sind Befehle, die den Inhalt des Programmzählers manipulieren, für einen Programmier-Neuling schwerer erfaßbar. Dies liegt darin, daß diese Befehle zusammen mit den Statusbefehlen eine Programmlogik darstellen – ein Thema, das für uns wenig Sinn hat, bevor wir die Programmierung überhaupt verstehen. Das Buch „Einführung in die Mikrocomputer-Technik" bringt eine ausführliche Diskussion dieser Befehlstypen.

Der Zweck der vorausgehenden Zusammenfassung von Befehlstypen lag darin, die Art von Operationen zu identifizieren, die den Befehlsvorrat eines Mikroprozessors bilden können. Natürlich sind wir in diesem Punkt noch weit davon entfernt, uns den Befehlssatz eines Mikrocomputers anzusehen und ihn zu gebrauchen. Wir sind jedoch nun vorbereitet, das Buch „Einführung in die Mikrocomputer-Technik" besser zu verstehen, welches im wesentlichen das gleiche Material wie Kapitel 4, 5 und 6 dieses Buches behandelt, jedoch wesentlich weiter ins Detail geht. Wenn wir dieses Anschlußwerk gelesen haben, so sind wir in der Lage mit der Anwendung von Mikroprozessoren zu beginnen.

KAPITEL 6

FRAGEN:

1. Die Anzahl der Bits, die von einem Mikroprozessor zur gleichen Zeit gehandhabt werden können, nennt man die _____ oder _____ eines Mikroprozessors.
2. Die meisten derzeit auf dem Markt befindlichen Mikroprozessoren verarbeiten Informationen in Einheiten zu _____ Bits.
3. Informationen innerhalb eines Mikrocomputer-Systems werden auf einer entsprechenden Anzahl von parallelen Leitern, dem sogenannten _____ übertragen.
4. Zur kurzzeitigen Aufbewahrung von Informationen in einem Mikrocomputer verwendet man eine mehr oder weniger große Anzahl von _____ .
5. Die verschiedenen arithmetischen und logischen Operationen an Dateneinheiten mit einer festen Wortlänge erfolgen in einer Einheit, die als _____ und _____ Einheit oder _____ bezeichnet wird.
6. Als _____ bezeichnet man die Größen oder die Werte, an denen die gewünschten arithmetischen oder logischen Operationen ausgeführt werden.
7. Das richtige Zusammenwirken der verschiedenen Einheiten im Mikroprozessor wird durch die _____ oder das _____ ausgeführt.
8. Da der Mikroprozessor Daten, Befehle und Zeichen nur in Form binärer Ziffernmuster verarbeiten kann, muß er die Ziffernmuster selbst richtig _____ .
9. Der momentan auszuführende Befehl wird in einem _____ für die arithmetische und logische Einheit gespeichert.
10. Die zeitliche Steuerung der Befehls-Ausführung wird durch das _____ bewirkt.
11. Jede Ausführung eines Befehls beginnt mit einer Phase, in der der Befehl im Binärcode aus dem Programm-Speicher _____ wird.
12. Das Zeitintervall, in der der Befehl geholt und ausgeführt wird, nennt man den _____ .
13. Die erforderliche Zeit zur Ausführung eines Befehls hängt von der Anzahl der für einen Befehl erforderlichen Taktimpulse und von der _____ ab.
14. Taktperioden moderner Mikrocomputer werden in Mikrosekunden und _____ gemessen.
15. Die Frequenz der Taktimpulse beträgt meist eine oder mehrere Millionen Impulse pro Sekunde, das heißt ein oder mehrere _____ .
16. Mit einer 16stelligen Binärzahl kann man bis zu _____ Speicherplätze adressieren.
17. Man kann einen Speicherplatz jedoch auch durch seine _____ vom Beginn des Datenbereiches an identifizieren.
18. Die Speicher-Adresse wird von der Zentraleinheit zum Speicher über den _____ gebracht.
19. Über den _____ werden die Daten zwischen Speicher und Zentraleinheit übertragen.

20. Damit der Speicher weiß, ob in ihn Daten eingeschrieben oder ausgelesen werden, erhält er über eine getrennte Steuerleitung geeignete _____ im richtigen Moment zugeführt.
21. Bei Schreib- oder Lese-Operationen müssen die Daten auf dem Daten-Bus so lange _____ gehalten werden, bis die entsprechenden Operationen abgeschlossen sind.
22. Die Adressierung des Programm-Speichers erfolgt über ein spezielles Register, dem sogenannten _____.
23. Bei normalem Ablauf eines Programmes wird der Befehlszähler nach der Ausführung eines Befehls automatisch um _____ erhöht, bzw. _____.
24. Bei Auftreten eines _____-Befehls wird die normale Reihenfolge der Befehle verlassen und zu einem anderen Befehl _____.
25. Auf externe Logik oder periphere Geräte wird ebenso wie auf einen Speicherplatz mittels einer bestimmten _____ zugegriffen.
26. In vielen Mikrocomputer-Systemen werden _____-Bausteine wie Speicher behandelt und ebenso adressiert.
27. In vielen Mikroprozessoren verwendet man ein sogenanntes "primäres" Register, das als _____ bezeichnet wird.
28. Befehle für KOMPLEMENTIEREN, SCHIEBEN und ROTATION benötigen nur einen _____.
29. Befehle für _____, _____, _____ und Exclusiv-ODER benötigen dagegen zwei Operanden.
30. Befehle für Operationen mit zwei Operanden sind im allgemeinen _____ als solche für Operationen mit einem Operanden.

ANTWORTEN, KAPITEL 6

1. Wortgröße, Wortlänge
2. 8
3. Bus
4. Registern
5. arithmetisch und logische, ALU
6. Operanden
7. Steuereinheit, Steuerwerk
8. interpretieren
9. Befehls-Register
10. Taktsignal
11. geholt
12. Maschinen-Zyklus
13. Taktfrequenz
14. Nanosekunden
15. Megahertz
16. 65536
17. Versetzung
18. Adressen-Bus
19. Daten-Bus
20. Schreib-/Lesesignale
21. stabil
22. Befehlszähler
23. 1, inkrementiert
24. Sprung, gesprungen
25. Adresse
26. Eingabe/Ausgabe
27. Akkumulator
28. Operanden
29. Addition, UND, ODER
30. komplexer

ANHANG A

Standard-Zeichencodes

Hexadezimale Darstellung	ASCII (7 Bit)	EBCDIC (8 Bit)	Hexadezimale Darstellung	ASCII (7 Bit)	EBCDIC (8 Bit)
0			30	0	
1			31	1	
2			32	2	
3			33	3	
4			34	4	
5			35	5	
6			36	6	
7			37	7	
8			38	8	
9			39	9	
A			3A	:	
B			3B	;	
C			3C	<	
D			3D	=	
E			3E	>	
F			3F	?	
10			40	@	blank
11			41	A	
12			42	B	
13			43	C	
14			44	D	
15			45	E	
16			46	F	
17			47	G	
18			48	H	
19			49	I	
1A			4A	J	]
1B			4B	K	.
1C			4C	L	<
1D			4D	M	(
1E			4E	N	+
1F			4F	O	!
20	blank		50	P	&
21	!		51	Q	
22	"		52	R	
23	#		53	S	
24	$		54	T	
25	%		55	U	
26	&		56	V	
27	'		57	W	
28	(		58	X	
29	)		59	Y	
2A	*		5A	Z	[
2B	+		5B	[	$
2C	,		5C	\	*
2D	–		5D	]	)
2E	.		5E		;
2F	/		5F		^

Hexadezimale Darstellung	ASCII (7 Bit)	EBCDIC (8 Bit)
60		
61	a	
62	b	
63	c	
64	d	
65	e	
66	f	
67	g	
68	h	
69	i	
6A	j	
6B	k	,
6C	l	%
6D	m	-
6E	n	)
6F	o	?
70	p	
71	q	
72	r	
73	s	
74	t	
75	u	
76	v	
77	w	
78	x	
79	y	
7A	z	
7B		#
7C		@
7D		'
7E		=
7F		"
80		
81		a
82		b
83		c
84		d
85		e
86		f
87		g
88		h
89		i
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		j
92		k
93		l

Hexadezimale Darstellung	ASCII (7 Bit)	EBCDIC (8 Bit)
94		m
95		n
96		o
97		p
98		q
99		r
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		s
A3		t
A4		u
A5		v
A6		w
A7		x
A8		y
A9		z
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		
C0		
C1		A
C2		B
C3		C
C4		D
C5		E
C6		F
C7		G

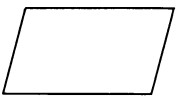
Standard-Zeichencodes (Fortsetzung)

Hexadezimale Darstellung	ASCII (7 Bit)	EBCDIC (8 Bit)	Hexadezimale Darstellung	ASCII (7 Bit)	EBCDIC (8 Bit)
C8		H	E4		U
C9		I	E5		V
CA			E6		W
CB			E7		X
CC			E8		Y
CD			E9		Z
CE			EA		
CF			EB		
D0			EC		
D1		J	ED		
D2		K	EE		
D3		L	EF		
D4		M	F0		0
D5		N	F1		1
D6		O	F2		2
D7		P	F3		3
D8		Q	F4		4
D9		R	F5		5
DA			F6		6
DB			F7		7
DC			F8		8
DD			F9		9
DE			FA		
DF			FB		
E0			FC		
E1			FD		
E2		S	FE		
E3		T	FF		

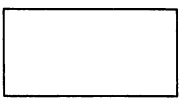
Standard-Zeichencodes (Fortsetzung)

ANHANG B

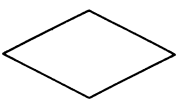
Standard-Symbole für Flußdiagramme



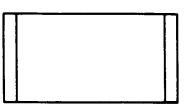
Eingabe, Ausgabe



Operationen für arithmetische Berechnungen und Datentransfer



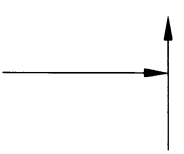
Verzweigungen für Entscheidungen



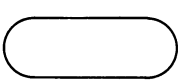
Unterprogramm

Übergangsstelle

Flußlinien, zur Verbindung der Symbole und zur Anzeige der Flußrichtung.



Zusammenführung



Grenzstelle. Zur Darstellung von Start oder Ende eines Programms oder Unterprogramms.

ANHANG C

μP-Lexi

access — *Zugriff*
Möglichkeit, eine bestimmte Speicherstelle anzusprechen (zu adressieren).

access time — *Zugriffszeit*
Die zwischen dem Anlegen der Adresse und der Abgabe von Daten eines Speichers erforderliche Zeit.

accumulator — *Akkumulator*
Register, in dem die Ergebnisse arithmetischer, logischer und E/A-Operationen gebildet werden. Der Inhalt des A. kann gelöscht, geprüft, komplementiert, inkrementiert, dekrementiert oder nach links oder rechts verschoben werden.

acknowledge — *Quittierung*
Bestätigung für ein empfangenes Signal.

ADC (Analog Digital Converter) — *Analog/Digital-Umsetzer, A/D-Wandler*
Schaltung zur Umwandlung eines in analoger Form vorliegenden Signales in den äquivalenten Digitalwert.

adder — *Addierer, Addierwerk*
Dient zur Bildung der Summe von zwei oder mehreren Eingangssignalen.

address — *Adresse, adressieren*
Ein Wort zur eindeutigen Identifizierung eines Speicherplatzes oder anderer Datenquellen und Senken.

algorithm — *Algorithmus*
Eine Rechenvorschrift zur Lösung einer Aufgabe in einer Reihe von Schritten. Jedes Computerprogramm stellt im Grunde einen Algorithmus dar.

ALGOL (ALGOrithmic Language) —
Problemorientierte, höhere Programmiersprache für technisch-wissenschaftliche Programme.

alphanumeric — *alphanumerisch*
Gemischte Darstellung aus Ziffern und Buchstaben.

ALU (Arithmetic and Logical Unit) — *Arithmetisch-logische Einheit, Rechenwerk*
Teil der Computer-Zentraleinheit. In ihr werden die arithmetischen und logischen Operationen ausgeführt. Diese Operationen werden durch Befehle in binär codierter Form ausgeführt.

ASCII (American Standard Code for Information Interchange) — *Amerikanischer Code für Informationsaustausch*
Alphanumerischer Standardcode meist „Aski" gesprochen.

assembler — *Assembler oder Assemblierer*
Ein Programm, das Programme aus der Assemblersprache (assembly language) in Maschinensprache übersetzt. Formale Fehler werden von diesem Programm erkannt.

assembly language — *Assemblersprache*
Programmiersprache, bei der anstelle des binären Maschinencodes einfach zu merkende Kürzel (Mnemoniks) verwendet werden. Außer den absoluten Adressen können symbolische Adressen verwendet werden. Anweisungen an den Assembler können mit Pseudo-Befehlen ausgeführt werden.

asynchronous — *asynchron*
Taktunabhängig, ohne Takt, nicht synchronisiert.

background program — *Hauptprogramm*
Hauptprogramm in einem Computersystem, das imstande ist, Unterbrechungen zu verarbeiten.

BASIC (Beginners All purpose Symbolic Instruction Code)
Einfach zu lernende höhere Programmiersprache.

baud rate — *Baud-Rate, Baud-Zahl*
Einheit der Übertragungsgeschwindigkeit. Ist gleich der Anzahl der Signalvorgänge pro Sekunde.

BCD (Binary Coded Decimal) — *Binär codierte Dezimalzahl*
Ein Code, bei dem jede Ziffer einer Dezimalzahl in ein binäres 4-Bit-Wort umgewandelt wird. Auch 1-2-4-8-BCD-Code genannt.

benchmark program — *Bewertungsprogramm*
Ein Programm zur Bewertung der Leistungsfähigkeit eines Computers.

bidirectional — *bidirektional*
In beide Richtungen arbeitend.

bit (binary digit) — *Bit*
Binäre Informationseinheit. Binärzeichen oder Dualziffer.

Boolean algebra — *Boole'sche Algebra*
System von Rechenregeln für binäre Variable.

branch — *Verzweigung*
Die Fortsetzung eines Programms an einer von zwei verschiedenen Adressen.

branch instruction — *Verzweigungsbefehl*
Befehl zur Weiterführung eines Programms an einer von zwei möglichen Adressen, abhängig von einer Bedingung.

breakpoint — *Haltepunkt*
Um einem Anwender Prüf- oder Eingabemöglichkeiten zu geben, kann ein Programm vor der Ausführung eines an einem bestimmten Haltepunktes gespeicherten Befehls gestoppt werden.

buffer — *Puffer*
Kleiner Speicher, in dem Daten kurzzeitig zwischengespeichert werden.

bug — *Fehler (wörtl. Wanze)*
Fehler in einem Software-Programm.

bus — *Bus, Sammelschiene*
Mehrfach-Datenleitung, an der verschiedene Einheiten gleichzeitig angeschlossen sind (Adressenbus, Datenbus, Steuerbus).

BUSEN (BUS ENable) — *Freigabesignal für den Bus*

byte — *Byte*
Bezeichnung für eine Gruppe von 8 Bits, die gemeinsam verarbeitet werden.

card puncher — *Lochkartenstanzer*

card reader — *Lochkartenleser*

carriage return — *Wagenrücklauf*

carry — *Übertrag*
Ein Signal oder ein Ausdruck, der auftreten kann, wenn die Summe zweier Ziffern mit demselben Stellenwert gleich oder größer als die Basis des verwendeten Zahlensystems ist.

carry look ahead — *Parallelübertrag*
Zur Erzielung einer raschen Verarbeitung eines Übertrages entsprechende logische Verknüpfungen.

cascade — *Kaskade, kaskadieren*
Beliebige Erweiterung der zu verarbeitenden Wortlänge durch Serienschaltung mehrerer Prozessor-Bausteine (slices).

character — *Zeichen, Symbol*
Kleinste Einheit, aus der sich Daten zusammensetzen. (Binäre Zeichen, Buchstaben, Ziffern und Sonderzeichen.)

chip — *Chip*
Monolithisches Halbleiterkristall-Plättchen oder Baustein, dessen Schaltung auf einem Halbleiterkristall aufgebracht ist.

chip-enable — *Baustein-Freigabe, Chip-Freigabe*
Freigabe eines Bausteines über ein Baustein-Freigabesignal.

chip-select — *Baustein-Auswahl, Chip-Auswahl*
Auswahl eines Bausteines über ein Baustein-Auswahlsignal.

chip slices — *(wörtl.) Chip-Scheiben*
Prozessorelemente für 2 oder 4 Bits, die zum Aufbau eines Mikroprozessors mit beliebiger Wortlänge zusammengeschaltet werden können.

clear — *löschen*
Zurücksetzen auf Null.

clock — *Takt*
Takt oder Taktimpuls. Bestimmt die Ausführungsdauer der Befehle.

compare — *vergleichen*

compatibel — *kompatibel, verträglich*
Software-kompatibel sind zwei Rechner, deren Programme ohne Änderung unter-

einander austauschbar sind.
Hardware-kompatibel sind Einheiten, die ohne Anpaßschaltung (z.B. bezüglich Signalpegel) zusammengeschlossen werden können.

compiler — *Compiler, Kompilierer*
Übersetzungsprogramm, das Programme aus einer höheren Programmiersprache in Maschinensprache umsetzt.

conditional — *bedingt*
Ausführung einer Operation entsprechend des Vorhandenseins bestimmter Bedingungen.

conditional jump — *bedingter Sprung*
Ein Sprung, der nur bei Vorliegen bestimmter Bedingungen ausgeführt wird.

control unit — *Steuereinheit, Steuerwerk*
1. Steuereinheit für die Reihenfolge der Befehle eines Maschinenprogramms.
2. Steuereinheit für periphere Geräte.

CPU (Central Processing Unit) — *Zentraleinheit*
Zentraleinheit oder Teil der Zentraleinheit. Wird manchmal auch als Mikroprozessor bezeichnet und besteht aus der arithmetisch-logischen Einheit (ALU), Steuerwerk (CU) und Registern.

CRC (Cyclic Redundancy Check) — *Zyklische Blockprüfung*
Verfahren zur Feststellung von Fehlern bei der Übertragung und Speicherung von Datenblöcken.

CRT (Cathode Ray Tube) — *Kathodenstrahlröhre*
Kathodenstrahlröhre für Datensichtgeräte.

cross assembler —
Ein Assembliererprogramm, das auf dem Computer 1 läuft und ein in Assemblersprache des Computers 2 geschriebenes Programm in die Maschinensprache von 2 übersetzt.

cycle — *Zyklus*
Ein Zeitintervall, in dem bestimmte Vorgänge ablaufen und abgeschlossen werden.

DAC (Digital to Analog Converter) — *Digital/Analog-Umsetzer, D/A-Wandler*

data bus — *Datenbus*
Gemeinsame Datenleitung, an die mehrere Einheiten gleichzeitig angeschlossen werden können.

debug, to — *Fehlersuchen (wörtl. Entwanzen)*
Suchen und Beseitigen von Fehlern in der Hardware und insbesondere in der Software.

decimal adjust — *Dezimalkorrektur, Dezimalanordnung*
Bei der Dezimalkorrektur wird der Inhalt eines Registers diesem entnommen und zur Schaffung des dezimalen Äquivalents der Binärzahl neu angeordnet.

decrement — *dekrementieren*
Schrittweises Vermindern um Eins.

development system — *Entwicklungssystem*
Mikrocomputer-Systeme zum raschen Testen von Software und Hardware, sowie dem Zusammenarbeiten beider. Wird auch zur PROM-Programmierung verwendet.

device — *Baustein, Einheit, Gerät*

diagnotistic routines — *Diagnose-Programme*
Programme zur Aufspürung und Lokalisierung von Fehlern in Hardware und Software.

digit — *Ziffer, Stelle*
Kleinste Einheit in einem digitalen System. Kann im Dezimalsystem die Werte der zehn Dezimalziffern annehmen, in einem Binärsystem die Ziffern Null und Eins. Im Engl. bedeutet digit auch eine Stelle in einer Zahl.

DIP (Dual-In-line-Package) —
Schaltungsgehäuse mit in zwei Reihen angeordneten Anschlüssen.

direct addressing — *direkte Adressierung*
Bei der direkten Adressierung ist die Adresse des Operanden Bestandteil des Befehls.

disk memory — *Plattenspeicher*
Speicher mit großer Kapazität (10^7- 10^9 Bits) aus rotierenden Platten mit magnetisierbarer Schicht. Der Zugriff zu den Speicherstellen ist wahlfrei.

display — *Anzeige*
Nichtschreibende, optische Anzeige von Daten. Reicht von der einfachen Anzeigelampe bis zum Sichtgerät mit Kathodenstrahlröhre (CRT).

DMA (Direct Memory Access) — *direkter Speicherzugriff*
Verfahren zur direkten Übertragung von Daten zwischen Speicher und E/A-Bausteinen, ohne daß die Daten über die CPU laufen müssen.

driver — *Treiber*
Schaltung zur Erzeugung der für das Ansteuern entsprechender Bausteine erforderlichen Leistung.

dynamic RAM (dynamic Random-Access Memory) — *dynamischer Schreib-/Lese-Speicher*
Speicher, in dem die Aufbewahrung der Informationen in elektrischen Ladungen erfolgt. Die Ladungen müssen ständig aufgefrischt werden (ca. alle 1 - 2 ms).

EAROM (Electrically Alterable ROM) — *elektrisch änderbarer Festwertspeicher*
Festwertspeicher, dessen Inhalt sich elektrisch verändern läßt.

editor — *Editor*
Ein Hilfsprogramm zur Unterstützung des Eingebens, Ausgebens, Korrektur und Speicherung von Programmen.

effective address — *effektive Adresse*
Bezeichnung für die absolute Adresse bei indirekter und indizierter Adressierung.

emulation — *Emulation*
Softwaremäßige Nachbildung (Simulation) eines Computers, wobei nur das äußere Verhalten eines Systems nachgebildet wird.

enable signal — *Freigabe-Signal*

encode — *codieren*
Das Zuordnen von Zeichen aus einem Zeichenvorrat zu Zeichen aus einem anderen Zeichenvorrat.

EPROM (Erasable-Programmable ROM) — *lösch- und programmierbarer Festwertspeicher*
Festwertspeicher, dessen gesamter Inhalt sich durch Bestrahlung mit UV-Licht löschen und sich anschließend wieder neu programmieren läßt.

erase — *löschen*
Löschen des Inhalts eines Speichers, z.B. bei einem EPROM mittels UV-Licht.

even parity — *gerade Parität*
Anzahl der auf "1" befindlichen Bits (pro Wort oder Byte) einschließlich des Paritätsbits wird gerade gemacht.

exchange, to — *austauschen*
Austauschen des Inhalts von Speicherplätzen mit Registern, Akkumulator etc.

execution time — *Befehls-Ausführungszeit*
Die Zeit, die zur vollständigen Ausführung eines Befehls erforderlich ist.

fetch, to — *holen, herausholen, abrufen*
Abrufen eines oder mehrerer Bytes (Befehle oder Daten) aus dem Speicher. Geschieht im Abrufzyklus (fetch machine cycle).

file — *Datei*
Zusammenstellung von Daten für einen bestimmten Zweck. Mehrere Sätze bilden eine Datei, mehrere Dateien eine Datenbank.

firmware — *Firmware*
In Festwertspeichern aufbewahrte Software.

flag — *Flagge, Kennzeichen*
Das Flag-Bit wird von der Software ge-

setzt, um einen bestimmten Zustand anzuzeigen und festzuhalten. Kann bei Bedarf später abgefragt werden, z.B. für bedingte Verzweigungen oder Sprünge.

floate, to — *floaten*
Das Versetzen eines Anschlusses oder einer Leitung in den hochohmigen Zustand. Beim DMA z.B. bringt sich die CPU in diesen Zustand und trennt sich damit vom angeschlossenen Bus.

floppy disk — *Magnetplatten-Speicher, Disketten*
Preiswerte magnetische Speicherplatten mit wahlfreiem Zugriff und großer Speicherkapazität.

flow-chart — *Flußdiagramm, Ablaufdiagramm*
Dient zur Darstellung des Verarbeitungsablaufes der Daten in Diagrammform. Für komplexe Programme ein unentbehrliches Hilfsmittel.

FORTRAN (FORmula TRANslation) — Problemorientierte Programmiersprache für technische und wissenschaftliche Programme.

FPLA (Field Programmable Logic Array) — Vom Anwender programmierbare logische Anordnung.

framing — *Rahmung, Einrahmung*
Bei asynchroner Datenübertragung gehen jedem Zeichen ein oder mehrere Startbits voraus und werden von einem oder mehreren Stopbits abgeschlossen.
Das Zeichen ist somit von Start- und Stopbits "eingerahmt".

hand shaking — *Quittungsbetrieb*
Geräte mit unterschiedlichen Reaktionsgeschwindigkeiten verständigen sich über den Beginn und Ende eines Austausches von Daten.

hardware — *Hardware*
Gemeinsamer Begriff für Bauteile und Geräte.

high — *hoch*
Zustand mit hohem Pegel (logisch H). Bei positiver Speisespannung entspricht "H" dem Zustand "Ein" (DIN 41785).

high-level language — *höhere Programmiersprache*
Sammelbegriff für alle problemorientierten Programmiersprachen (z.B. FORTRAN, BASIC, ALGOL, PL/1). Im Gegensatz hierzu stehen die maschinenorientierten Programmiersprachen (Assemblersprachen).

immediate addressing — *unmittelbare Adressierung*
Bei der unmittelbaren Adressierung enthält der Befehl keine Adresse sondern den Operanden direkt. Da ein Befehl nur Konstante beinhalten kann, ist dies nicht für Variable möglich.

increment — *inkrementieren*
Schrittweises Erhöhen um 1, z.B. automatisches Inkrementieren des Befehls-Zählers zur Bildung der neuen Adresse.

index register — *Index-Register*
Register, die zur Änderung von Adressen verwendet werden, die in Befehlen enthalten sind (indirekte Adressierung).

indexed addressing — *indizierte Adressierung*
Bei der indizierten Adressierung wird der im Befehl angegebenen Adresse der Inhalt eines Indexregisters hinzu addiert, um die Adresse des Operanden zu erhalten. Durch Änderung des Inhalts des Indexregisters kann die effektive Adresse abgeändert werden, ohne daß der Befehl modifiziert werden muß. Besonders nützlich bei Schleifen für die Abarbeitung von Tabellen.

indirect addressing — *indirekte Adressierung*
Bei der indirekten Adressierung ist der im Befehl angegebene Operand die Adresse des Operanden.

inhibit — *Sperr . . .*

instruction — *Befehl*
Anweisung für den Computer zur Durchführung bestimmter Operationen.

instruction set — *Befehlssatz, Befehlsvorrat*
Der Satz von Befehlen in Maschinensprache, der von einem Rechner ausgeführt werden kann. Wichtige Befehlsarten sind:
1. Arithmetische und logische Befehle.
2. Befehle zur Steuerung des Programmablaufs.
3. Befehle zum Transfer von Daten.

Interface — *Interface, Schnittstelle, Anpassungsschaltung*
Schnitt- oder Trennstelle zwischen zwei Geräten, pegel- und ablaufmäßig genormt. Elektronische Schaltung, die zwei Geräte oder Bausteine einander anpaßt.

Interrupt — *Unterbrechung, Programmunterbrechung*
Das momentan ablaufende Programm wird unterbrochen, alle wichtigen Registerinhalte werden gerettet (saved) und ein Unterbrechungs-Serviceprogramm ausgeführt. Danach werden die geretteten Werte wieder in die Register geladen und das Hauptprogramm wird fortgesetzt. Dient zur Behandlung von asynchronen (insbesonders E/A) Vorgängen.

interrupt request — *Unterbrechungs-Anforderung*
Signal von einer externen Einheit, die eine Unterbrechung anfordert.

interrupt service routine — *Unterbrechungs-Serviceprogramm*
Unterprogramm, zu dem bei Eintreffen einer Unterbrechungs-Anforderung gesprungen wird.

I/O (Input/Output) — *E/A (Eingabe/Ausgabe)*

I/O-port — *Ein-/Ausgabe-Kanal (oder Tor)*

jump — *Sprung*
Fortsetzung des Programms mit einem anderen als dem nächstfolgenden Befehl.

keyboard — *Tastatur*

label — *Marke, Markierung, Kennzeichen*
In Programmiersprachen eine symbolische Adresse.

latch — *Signalspeicher*
Signalspeicher oder Auffangspeicher, meist ein Flip-Flop.

least significant bit (LSB) — *Bit mit dem niedrigsten Stellenwert*

LIFO (Last-In/First-Out) —
Speicherprinzip, bei dem die zuletzt gespeicherten Daten wieder zuerst ausgegeben werden. Entnahmeprinzip beim Stapelspeicher (stack).

linking loader — *Binde-Lader*
Ein Ladeprogramm zum Zusammenfügen mehrerer unabhängig voneinander erstellten Programmteile zu einem Gesamtprogramm.

listing — *Auflistung, gedruckte Liste*

literal — *Literal*
Wenn eine Konstante, die als Rechengröße benötigt wird und sich während des Programmablaufs nicht ändert, direkt als Operand angegeben wird, nennt man sie Literal.

loader — *Lader*
Programm, das ein Maschinenprogramm an eine bestimmte Stelle im Speicher lädt. Hierbei erfolgt u.U. eine Umrechnung von Adressen.

location — *Speicherplatz, Speicherzelle*
Speicherstelle eines Wortes, die durch eine Adresse eindeutig gekennzeichnet ist.

loop — *(Programm-)Schleife*
Eine Serie von Befehlen, die zur Lösung einer Aufgabe mehrmals nacheinander durchlaufen werden kann.

LSI (Large Scale Integration) — *Hoher Integrationsgrad*
Komplexe Schaltungen mit mehr als 1000 Gattern auf einem einzelnen Chip.

machine language — *Maschinensprache*
Die Maschinensprache besteht aus Befehls-Bitmustern, die der Computer direkt decodieren und ausführen kann. Programme in anderen Sprachen müssen in Maschinensprachen übersetzt werden.

machine program — *Maschinenprogramm*
Programm in Maschinensprache.

macro-instruction — *Makrobefehl*
Eine Folge von Mikrobefehlen, die in einem Makrobefehl zusammengefaßt werden.

mask — *Maske*
Ein Bitmuster zur gezielten Ausblendung von Binärstellen.

mass storage — *Massenspeicher*
Speicher mit sehr großer Speicherkapazität, z.B.: Magnettrommeln, Magnetplatten — im Gegensatz zum Arbeits- oder Programmspeicher.

memory — *Speicher, Speicherbaustein*
Einrichtung zur Aufbewahrung von Informationen (Daten, Befehle, Adressen etc.) über einen beliebig langen Zeitraum.

microcomputer — *Mikrocomputer*
Ein Computer mit einem Mikroprozessor als Zentraleinheit, sowie Programmspeicher (meist ROM oder PROM), Datenspeicher (RAM) und E/A-Bausteinen.

micro instruction — *Mikrobefehl*
Computer führen Befehle in Maschinensprache meist als Sequenz (Folge) kleinerer Programmschritte in Form von Mikrobefehlen aus.

microprocessor — *Mikroprozessor*
Zentraleinheit (CPU = Central Processing Unit) auf einem oder mehreren Chips.

microprogram — *Mikroprogramm*
Folge von Mikrobefehlen.

mnemonic — *Mnemonik*
Mnemonischer oder mnemotechnischer Code. Ein Code, dessen Abkürzungen ein Hinweis auf dessen Eigenschaften zur Gedächtnisstütze enthält. Es handelt sich insbesonders um leicht zu merkende, alphanumerische Kürzel für Befehle, die wesentlich leichter als ein Zahlencode zu behalten sind.

modem (MOdulator/DEModulator) — *Modem*
Kombinierter Modulator und Demodulator für die Übertragung von Daten.

monitor — *Monitor*
Ein Programm zur Steuerung oder Koordination anderer Programmteile oder selbständiger Programme.

most significant bit (MSB) — *Bit mit dem höchsten Stellenwert*

MSI (Medium Scale Integration) — *Mittlerer Integrationsgrad*
Schaltungen mit weniger als 100 Gattern auf einem einzelnen Chip.

multibyte instruction — *Mehrwort-Befehl*
Ein Befehl, dessen erforderliche Bit-Zahl größer als die Wortlänge des Computers ist. Dieser Befehl muß daher in zwei oder mehr Speicherplätze abgelegt werden.

multilevel interrupt — *Mehr-Pegel-Unterbrechung*
Eine Unterbrechung mit hoher Priorität kann eine andere mit niedrigerer Priorität unterbrechen.

multiplex — *Multiplex*
Im allgemeinen werden im Zeitmultiplex-Verfahren die auf einer größeren Anzahl von Kanälen vorliegenden Daten auf eine geringere Anzahl (oder einen einzelnen) Kanal umgesetzt.

nesting — *Verschachtelung*
Wenn von einem Unterprogramm ein weiteres Unterprogramm aufgerufen wird (z.B. bei Unterbrechungs-Programmen).

number representation — *Zahlendarstellung*
Im Rechner werden Zahlen im Dualsystem oder im BCD-Code verarbeitet. Zur Eingabe oder Ausgabe werden drei oder vier Bits im Oktal- und Hexadezimalsystem zusammengefaßt.

object code — *Maschinencode*
Anweisungen, die von der Maschine unmittelbar interpretiert werden können.

object program — *Maschinenprogramm*
Ein Programm in Maschinensprache, das vom Assembler oder Kompilierer erzeugt wurde und direkt geladen werden kann.

odd parity — *ungerade Parität*
Anzahl der auf "1" befindlichen Bits (pro Wort oder Byte) einschließlich des Paritätsbits wird ungerade gemacht.

on-line — *(wörtl.) "in der Leitung"*
Mit der Anlage verbundener, synchronisierter Betrieb.

operand — *Operand*
Information oder Rechengröße, an der eine entsprechende Operation durchgeführt wird.

overflow — *Überlauf*
Überschreiten der Stellenanzahl durch das Ergebnis.

PACE (Processing And Control Element) — *Mikroprozessor von National Semiconductor*

paged addressing — *Seiten-Adressierung*
Durch Aufteilung des Speichers in Abschnitte (Seiten) kann man den Aufwand der Adressierung innerhalb der gleichen Seite verringern, indem kürzere Adressen verwendet werden. Der Nachteil ist, daß eine Adressierung außerhalb der Seite nur mit erhöhtem Aufwand möglich ist (Verbesserung siehe "relative addressing").

parity — *Parität*
Die Kontrolle der Parität ist ein Hilfsmittel zur Datensicherung. Durch Hinzufügen eines Paritätsbits wird die Anzahl der "1"-Bits gerade (even) oder ungerade (odd) gemacht und bei Empfang kontrolliert.

peripheral (device) — *Peripheriegerät, peripheres Gerät*
Alle Speicher- und Eingabe/Ausgabe-Einheiten, die nicht zur Zentraleinheit gehören und vom Computer angesprochen werden können.

PLA (Programmable Logic Array) — *programmierbare logische Anordnung*
Integrierte Schaltung, mit der sich beliebige Verknüpfungen binärer Eingangsgrößen durchführen lassen (z.B. Code-Wandlung).

PL/M (Programming Language for Microcomputer) — *Programmiersprache für Mikrocomputer*
Basierend auf PL/1 (Programming Language 1 = höhere Programmiersprache).

PMOS — *P-Kanal-MOS*
MOS-Technologie. Besitzt gegenüber NMOS größere Kristallflächen- und Leistungsbedarf sowie höhere Schaltgeschwindigkeiten.

pointer — *Zeiger, Adressenverweis*
Ein Speicherplatz, der eine Adresse enthält. Bei häufiger Verwendung wird hierfür ein Register benützt. (z.B. instruction counter, stack pointer).

pooling — *Pooling*
Bildung von Pools. Das Abspeichern von aus mehreren Bytes bestehenden Befehl-Codes (multibyte instruction object codes) in dafür bestimmte Speicher, von dort können sie mit speziellen Ein-Byte-Referenzen abgerufen werden. Diente ursprünglich zur Einsparung von Speicherraum, kann jedoch zur direkten Steuerung der Mikrocomputer-Logik durch externe Logik verwendet werden.

pop — *pop* Entnehmen von Informationen aus dem Stapelspeicher.

port — *Kanal (Tor) Port*
Eingabe- oder Ausgabe-Baustein.

priority — *Priorität, Rangfolge*
Teile eines Systems mit höherer Priorität werden bei Konkurrenzfällen mit Vorrang behandelt.

program — *Programm*
Folge (oder Sequenz) von Befehlen zur Lösung einer Aufgabe.

program counter — *Programmschritt-Zähler, Befehlszähler*
Register, das die Adresse des nächsten auszuführenden Befehls enthält.

PROM (Programmable ROM) — *programmierbarer Festwertspeicher*
Vom Benutzer programmierbarer Speicher, dessen Inhalt jedoch nicht mehr gelöscht werden kann.

pseudo instruction — *Pseudobefehl*
Eine Anweisung an den Assembler, die nicht übersetzt wird.

push — *push*
Einspeichern von Informationen in den Stapelspeicher.

RAM (Random Access Memory) — *Speicher mit wahlfreiem Zugriff*
Schreib-/Lese-Speicher mit wahlfreiem Zugriff.

random access — *wahlfreier Zugriff*
Bei Speichern mit wahlfreiem Zugriff ist die Zugriffszeit unabhängig von der gewählten Adresse (im Gegensatz zu den sequentiellen Speichern).

real-time — *Realzeit, Echtzeit*
Ein Computer, der im Echtzeit-Betrieb arbeitet, nimmt direkt Einfluß auf den Prozeß in den er einbezogen ist.

redundancy — *Redundanz*
Die zur Darstellung einer Information zur Verfügung stehenden, aber nicht benötigten Zeichen werden als Redundanz (wörtl. Weitschweifigkeit) dieser Information bezeichnet. Je höher die Redundanz, desto sicherer ist der Code gegen Fehler, er wird jedoch umso umfangreicher.

register — *Register*
Eine Anordnung die imstande ist, kleine Einheiten digitaler Informationen vorübergehend zu speichern und mit kurzer Zugriffszeit abzugeben. Auf die Bits wird meist parallel zugegriffen.

register-register instruction — *Register-Register-Befehl*
Ein Befehl, bei dem sich der Operand in einem Register befindet und das Ergebnis wieder in einem Register abgelegt wird.

relative addressing — *relative Adressierung*
Verbesserte Form der Seiten-Adressierung. Es wird eine Abweichung von der vorherigen Adresse von ± 1/2 Seite angegeben. Dadurch wird der erhöhte Aufwand bei den Seitenübergängen vermieden, der jedoch bei Adressierungen außerhalb des relativen Seitenbereiches wieder voll auftritt.

relocatable — *verschiebbar*
Nicht an eine absolute Adresse gebunden.

relocating loader — *Lader für verschiebbare Programme*
Von diesem Lader kann ein verschiebbares Objektprogramm an einen beliebigen Platz im Programmspeicher für die Ausführung gebracht werden.

REPROM (REprogrammable PROM) — *Wiederprogrammierbares ROM*
Ein Festwertspeicher, der gelöscht und wieder programmiert werden kann.

resident — *(wörtl.) ortsansässig*
a) Auf dem eigenen Computer laufend (Gegenteil: cross)
b) Ständig im Hauptspeicher befindlich.

restart — *neuerlicher Start*
Neuer Start nach einer Unterbrechung.

return address — *Rückkehradresse, Rücksprungadresse*
Die Adresse, zu der nach Beendigung des Unterprogramms zurückgekehrt wird.

ROM (Read Only Memory) — *Festwertspeicher, Nur-Lese-Speicher*
Festwertspeicher mit konstantem unveränderlichem Inhalt, z.B. Programm, Konstante.

scratch pad — *Zwischenregister (wörtl. "Notizblock")*
Ein Register, das für kurzfristige Aufbewahrung von Zwischenergebnissen verwendet wird.

sequential access — *sequentieller Zugriff*
Der Zugriff zu gespeicherten Daten ist nur nach dem Lesen aller vorher gespeicherten Informationen möglich, z.B. bei einem Magnetband.

shift — *Verschiebung*
Eine Operation, bei der eine Anzahl von Bits eine oder mehrere Stellen nach links oder rechts verschoben wird.

sign — *Vorzeichen*

simulation — *Simulation*
Anstelle eines tatsächlichen Systems wird ein Modell dieses Systems verwendet, das in seinem äußeren Verhalten der internen Arbeitsweise dem wirklichen System gleicht.

simulator — *Simulator*
Ein Programm zur Nachbildung von Befehlen. Wird meist zum Überprüfen eines fertigen Programms verwendet.

slice — *(wörtl.) Scheibe*
Prozessorelement (für 2 oder 4 Bits). Mehrere hiervon können zum Aufbau eines Mikroprozessors mit beliebiger Wortlänge zusammengeschaltet werden (bit slice).

software — *Software*
Zur praktischen Verwendung der Computer-Hardware benötigt man Programme, die als Software bezeichnet werden. System-Software sind alle Grundprogramme, Assembler, Kompilierer, Lader etc. Anwender-Software sind alle Programme zur Lösung spezieller Anwenderprobleme.

source program — *Quellenprogramm*
Ein Programm in einer Programmiersprache. Dieses muß anschließend assembliert oder kompiliert werden.

stack — *Stapelspeicher, Kellerspeicher*
Ein Speicherbereich außerhalb des Steuerwerks, aus dem Informationen nur am gleichen Speicherplatz entnommen werden können, an dem sie eingegeben wurden. (Das Prinzip wird daher auch LIFO = last in first out genannt.)

status — *Zustand*

stack pointer — *Stapelzeiger*
Verweist auf die Adresse des zugänglichen Speicherplatzes des Stapelspeichers, an dem zugefügt oder weggenommen wird.

storage — *Speicher*
Synonym mit memory.

subroutine — *Unterprogramm*
Eine Folge von Befehlen zur Lösung einer Teilaufgabe. Wird vom Hauptprogramm oder anderen Unterprogrammen aufgerufen. Der Abschluß mit einem RETURN-Befehl sichert die Rückkehr zum rufenden Programm.

symbolic address — *symbolische Adresse*
Frei gewählte Bezeichnung eines Speicherplatzes, der vom Assemblierer (oder Kompilierer) in eine absolute Adresse umgesetzt wird.

synchron — *synchron*
Mit festem Grundtakt arbeitend.

tape — *Band*
a) Lochstreifen (paper tape)
b) Magnetband (magnetic tape)

terminal — *Terminal, Datenstation*
Datenstationen für den Benützer, bestehend aus einer Ausgabeeinheit (z.B. Bildschirm, Drucker) und einer Eingabetastatur.

throughput — *Durchsatz*
Die Anzahl der von einem Computer je Zeiteinheit erledigten Aufgaben.

Time-sharing system — *Teilnehmer-System*
Eine Betriebsart, bei der mehrere Benutzer einen Rechner (scheinbar) gleichzeitig für verschiedene Probleme benutzen können.

TTY (Tele TYpewriter) — *Fernschreiber*
Für Mikrocomputer wichtiges E/A-Gerät.

twos-complement — *Zweierkomplement*
Wird aus dem Einerkomplement einer Binärzahl durch Addition von 1 gebildet, z.B. Binärzahl 0100, Einerkomplement 1011, Zweierkomplement 1100. Stellt das binäre Äquivalent eines Zehnerkomplements dar.

UART (Universal Asynchronous Receiver/ Transmitter) — *Universeller asynchroner Empfänger/Sender*
Ein Baustein, in dem der Sender die Parallel-Serien-Umsetzung der asynchron zu übertragenden Daten und der Empfänger die entsprechende Serien-Parallel-Umsetzung durchführt.

unconditional jump — *unbedingter Sprung*
Ein Sprung, dessen Ausführung an keine Bedingung gebunden ist.

user program — *Anwenderprogramm*
Spezialprogramm zur Lösung einer Aufgabe des Anwenders. (Gegensatz: Universal- oder Betriebsprogramm).

utility program — *Dienstprogramm*
Allgemeines Hilfsprogramm, das den Umgang mit einem Datenverarbeitungs-System erleichtert und vereinfacht.

vektor interrupt — *gerichtete Unterbrechung*
Unterbrechung, bei der jeder anfordernde Baustein ein eigenes Serviceprogramm erhält.

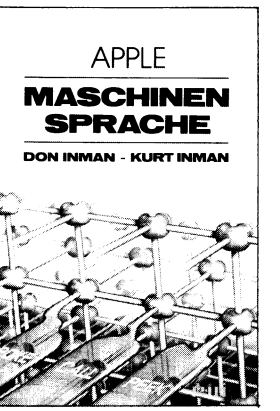
volatile — *flüchtig*
Speicher, deren Inhalt bei Ausfall oder Abschalten der Stromversorgung verlorengeht, nennt man flüchtig.

word size — *Wortlänge*
Entsprechend dem Anwendungsgebiet werden mehrere Bits zu Worten unterschiedlicher Länge (4, 8, 12, 16 und 32 Bits) zusammengefaßt. Ein Wort mit 8 Bits wird als Byte bezeichnet.

CBM COMPUTER HANDBUCH

Dieses Handbuch hilft Ihnen, Ihren CBM-Computer erst richtig zu verstehen! Beim Studium werden keine Vorkenntnisse erwartet. Es ist randvoll mit kurzen Programmbeispielen. Auch die Peripherie wird eingehend behandelt.

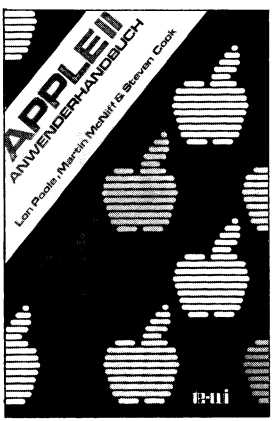
Von Adam Osborne und Caroll S. Donahue
544 Seiten, DM 59,–



APPLE MASCHINENSPRACHE

Ziel dieses Buches ist es, dem Benutzer eines Apple Computers die Verbindung zu schaffen, um vom Programmieren in BASIC auch zum Programmieren in MASCHINENSPRACHE zu gelangen. Grundkenntnisse in BASIC sind hierbei Voraussetzung. In kleinen, leichten Schritten und an anschaulichen Demonstrationsprogrammen mit Farbe, Grafik und Akustik wird der Übergang vom Programmieren von BASIC in MASCHINENSPRACHE geebnet.

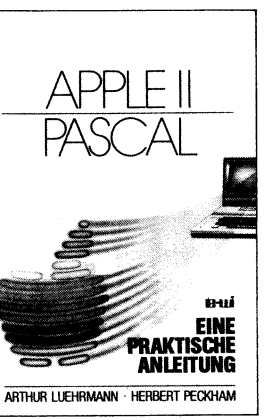
Von Don Inman und Kurt Inman
300 Seiten, DM 49,–



APPLE II PASCAL – EINE PRAKTISCHE EINFÜHRUNG

Dieses Buch ist unentbehrlich für alle, die die Programmiersprache PASCAL lernen wollen. Vorkenntnisse werden nicht erwartet. An Hand von Beispielen und Übungen lernen Sie, selbst Programme in PASCAL zu entwickeln und auszutesten.

Von Arthur Luehrmann und Herbert Peckham
544 Seiten, DM 59,–



APPLE II ANWENDERHANDBUCH

Der richtige Leitfaden für Ihren APPLE II! Mit Hilfe dieses Buches werden Sie Ihren APPLE-Computer erfolgreich einsetzen. Es beschreibt den Computer als solchen und gibt ausführlich Auskunft über die gesamte Peripherie. Die deutsche Übersetzung entstand in enger Zusammenarbeit mit der Herstellerfirma.

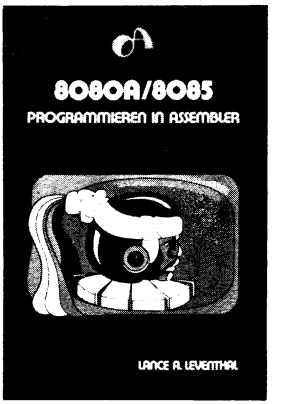
Von Lon Poole und Martin McNiff
400 Seiten, DM 56,–

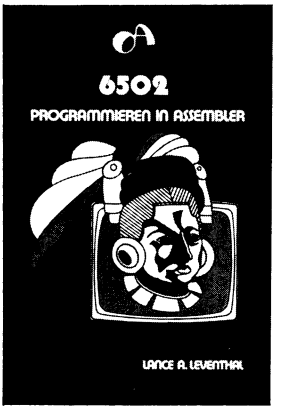
te-wi
te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Die Preise sind die Ladenpreise

PROGRAMMIEREN IN ASSEMBLER

In dieser Buch-Serie wird die Programmierung der Mikroprozessoren 6800, 8080A/8085 und 6502 in Assemblersprache beschrieben. Die Titel enthalten eine große Auswahl von praktischen Programmen in Standardformat einschließlich Flußdiagrammen, Quellprogrammen, Objektcodes und erläuternden Texten. Jeder Befehl der entsprechenden CPU wird detailliert erklärt.



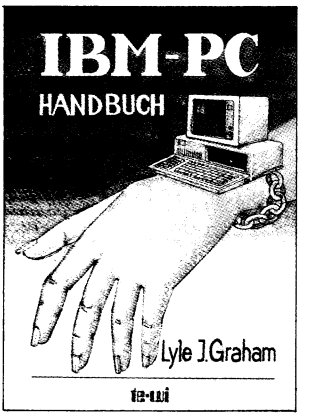


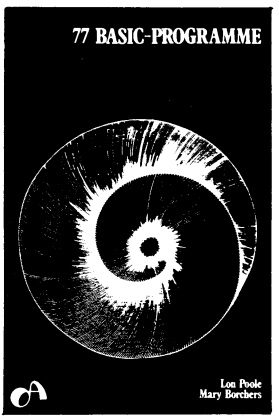
Von Lance A. Leventhal
600 bzw. 512 Seiten, DM 59,–
("6502") bzw. DM 49,–

IBM-PC HANDBUCH

Das US-Textbuch zum IBM-PC – jetzt in Deutsch! Souverän in der Darstellung von Hard- und Software des IBM-Modells eines anspruchsvollen Personalcomputers. Pragmatisch für die erste Begegnung mit einem IBM-PC, von der Installation bis zum geschäftlichen Einsatz. Professionell mit Kapiteln auch über Datenübertragung, CP/M86, Grafik, Drucksteuerung, IBMDOS, usw.

Von Lyle J. Graham
400 Seiten, DM 59,–





77 BASIC-PROGRAMME

Eine Sammlung von 77 praktischen Kurzprogrammen in BASIC, die mathematische, finanztechnische, statistische und verschiedene allgemeine Aufgaben behandeln. Die ausführlich erläuterten Befehle lassen sich leicht direkt anwenden oder sie dienen als Übungen.

Von Lon Poole und Mary Borchers
208 Seiten, DM 39,–

Die Preise sind die Ladenpreise

te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

C64 COMPUTERHANDBUCH

Ein Handbuch für jeden Erfahrungsstand: von der ersten Begegnung bis zum professionellen Einsatz des COMODORE 64 bzw. 1541. Das Werk ist sehr bildreich gestaltet und bietet somit eine schnelle Übersicht — als echtes Nachschlagwerk werden Sie es stets in der Nähe Ihres Computers finden.

ca. 400 Seiten, DM 56,–

MEIN ATARI® COMPUTER

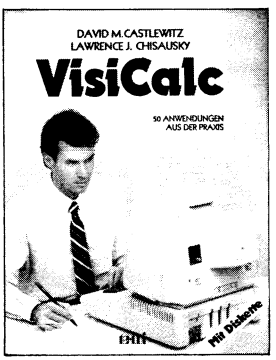
Dieses Buch macht die Möglichkeiten, die in Ihrem ATARI-Computer stecken, auf leichtverständliche Art transparent. In einfachen Schritten wird der Anwender mit der Bedienung der Geräte und der Software vertraut gemacht. Zahlreiche Tips zur Aufdeckung und Beseitigung von möglichen Fehlerquellen bei Hard- und Software helfen bei scheinbar unlösbaren Problemen.

Von Lon Poole, Martin McNiff & Steven Cook
500 Seiten, DM 59,–

EINFÜHRUNG IN DIE MIKROCOMPUTERTECHNIK

Dieses schon legendäre Standardwerk über die Mikrocomputertechnik von Erfolgsautor Osborne ist neu aufgelegt und völlig neu überarbeitet worden. Jetzt spiegelt sich darin der allerletzte Stand dieser faszinierenden Technologie wieder. In bewährter Manier ist es überaus reichhaltig bebildert. Bereits an mehr als 500 Universitäten weltweit ist es als reguläre Studiengrundlage eingeführt.

Von Adam Osborne
488 Seiten, DM 66,–



VISICALC® – 50 ANWENDUNGEN AUS DER PRAXIS

VisiCalc ist zum Inbegriff für die moderne Art der Berechnung und die anschaulich-übersichtliche Darstellung von Daten auf Heimcomputern geworden. Dieses Buch enthält eine Sammlung der 50 häufigsten VisiCalc-Anwendungen aus Wirtschaft und Privatbereich. Alle Berechnungen und die Darstellungsform sind auf der dem Buch beigefügten 5 1/4"-Diskette gespeichert.

Von David M. Castlewitz und Lawrence J. Chisausky
184 Seiten, DM 79,–

te-wi
te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Die Preise sind die Ladenpreise

D736.-170

Weitere deutsche Bücher aus dem te-wi Verlag

Einführung in die Mikrocomputer-Technik (Adam Osborne)
Die umfassendste, vollständigste und neutralste Darstellung. Das μP-Standardwerk von Dr. Adam Osborne. Jetzt völlig neu überarbeitet!

VisiCalc – 50 Anwendungen aus der Praxis
(David M. Castlewitz/Lawrence J. Chisausky)
VisiCalc ist zum Inbegriff für die moderne Art der Berechnung und anschaulich-übersichtliche Darstellung von Daten auf Heimcomputern geworden. Dieses Buch enthält eine Sammlung der 50 häufigsten VisiCalc-Anwendungen in Wirtschaft und Privatbereich. Alle Berechnungen und die Darstellungsform sind auf der beigefügten 5¼"-Diskette gespeichert.

Programmieren in Assembler (Lance A. Leventhal)
Mit den bekannten Bausteinen 6502, 6800 und 8080 A/8085 gibt es eine ganze Reihe von Büchern in deutscher Sprache über die Programmierung in Assembler.

CBM Computer Handbuch (A. Osborne/Caroll S. Donahue)
Für alle CBM-Besitzer oder die, die es werden wollen, bietet diese einzigartige Fundgrube eine schrittweise Einführung bis hin zur professionellen Ausnutzung aller Möglichkeiten des beliebten Computers.

APPLE II – Anwender Handbuch (Lon Poole)
Dieses Buch erspart Ihnen zeitraubendes und nutzloses Suchen nach der wirklich verwendbaren Dokumentation für Ihren Computer. Hiermit lernen Sie Ihren Apple II erst richtig kennen.

APPLE II PASCAL – Eine praktische Anleitung (Arthur Luermann/Herbert Peckham)
Unentbehrlich für alle, die die Programmiersprache PASCAL lernen wollen und Zugang zu einem Apple-Computer haben.

APPLE MASCHINENSPRACHE (Don Inman und Kurt Inman)
Dieses Buch schafft dem Benutzer eines APPLE-Computers eine Brücke vom Programmieren in BASIC zum Programmieren in MASCHINENSPRACHE. Grundkenntnisse in BASIC werden vorausgesetzt.

Mein ATARI®-COMPUTER (L. Poole, M. McNiff & S. Cook)
Dieses Buch macht die Möglichkeiten, die in Ihrem ATARI Computer stecken, auf leichtverständliche Art transparent. Dieser Leitfaden gehört in die Nähe Ihres Heimcomputers.

C 64 COMPUTERHANDBUCH
Ein Handbuch für jeden Erfahrungsstand: von der ersten Begegnung bis zum professionellen Einsatz des COMMODORE 64. Das Werk ist sehr bildreich gestaltet und ein klassisches Nachschlagewerk.

ISBN 3-921803-02-0