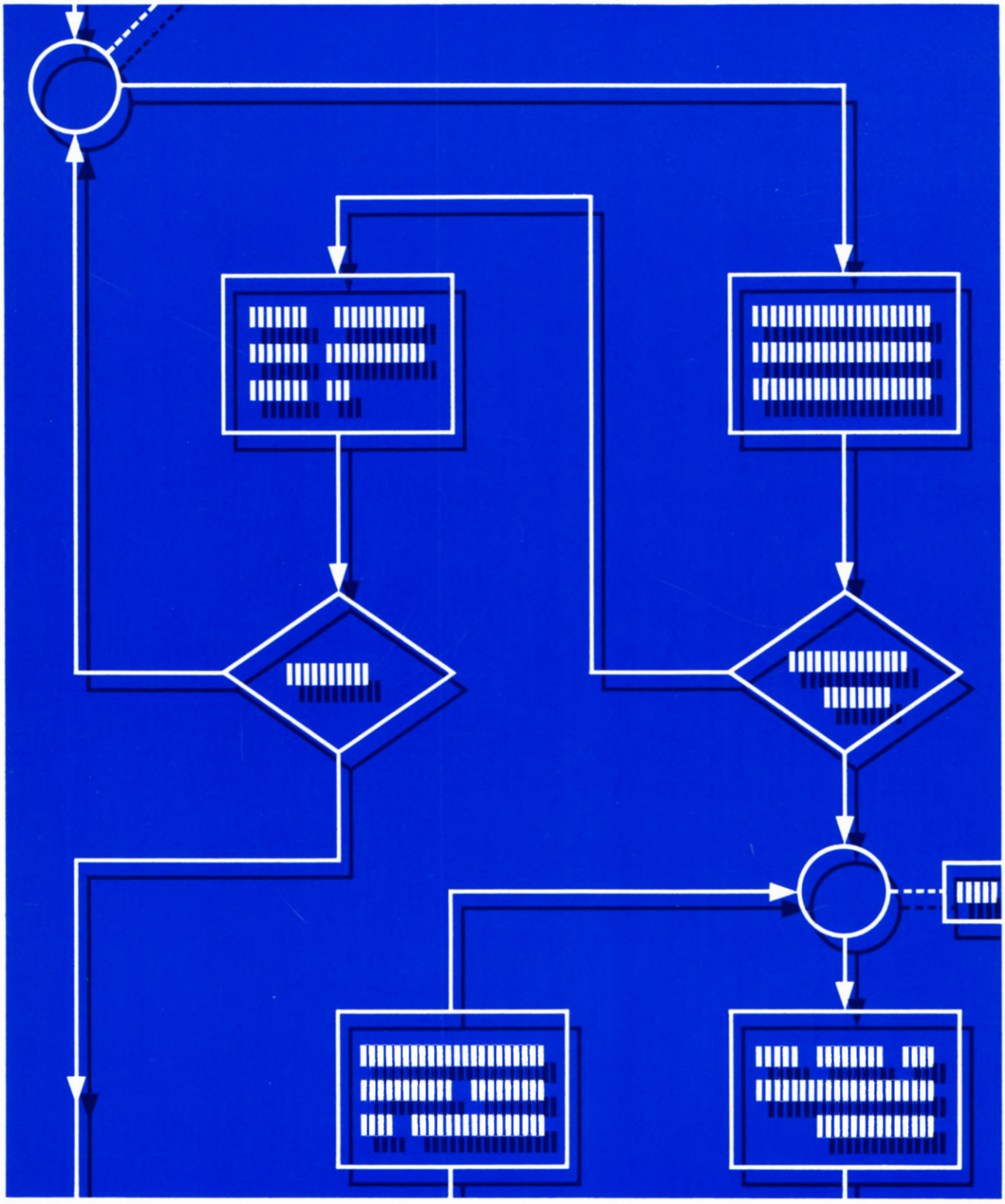
Mikroprozessortechnik

Einführung mit dem Micro-Professor

1



Christiani

Lehrgang MIKROPROZESSORTECHNIK

Herausgeber: R. Christiani

Stichwörter STICH-WÖRTER

Hardware H

Software S

Prüfungsaufgaben P

Übungen Ü

Tabellen T

Stichwörter

Stichwörter

J

K

L

M

N

O

P

R

S

T

U

V

W

Z

MIKROPROZESSORTECHNIK 1

Herausgeber: R. Christiani

Inhaltsverzeichnis

Fehler: H2 S20

TECHNISCHES LEHRINSTITUT DR.-ING. P. CHRISTIANI KONSTANZ

2.v.A.830303

MIKROPROZESSORTECHNIK 2

Herausgeber: R. Christiani

Inhaltsverzeichnis

MIKROPROZESSORTECHNIK 3

Herausgeber: R. Christiani

Inhaltsverzeichnis

TECHNISCHES LEHRINSTITUT DR.-ING. P. CHRISTIANI KONSTANZ

Dr.-Ing. P. Christiani GmbH · Technisches Lehrinstitut und Verlag

7750 Konstanz
Hermann-Hesse-Weg 2
Telefon (0 75 31) 58 01-0
Telex 7 33 304
Btx-Nr. •1380 #

Dr.-Ing. P. Christiani GmbH · Hermann-Hesse-Weg 2 · 7750 Konstanz Durchwahl-Nr. 58 01-

Herrn
Grams

Ihre Zeichen	Ihre Nachricht vom	Unsere Nachricht vom	Unsere Zeichen	7750 Konstanz
150 069 086			Re/ha	12.06.84

Sehr geehrter Herr Grams,

Sie haben uns auf drei Druckfehler aufmerksam gemacht - vielen Dank.

Selbstverständlich muß es heißen:

Seite 2/46 LD B,E
" 3/2 LD (adr), A
" 3/30 OUT (Ø2), A
" " IN A, (ØØ)

Wir bitten Sie, diese Fehler zu entschuldigen. Vor dem Druck einer neuen Auflage werden wir eine entsprechende Korrektur vornehmen.

Mit freundlichen Grüßen

Dr.-Ing. P. Christiani GmbH
Techn.Lehrinstitut u.Verlag
Abt.Teilnehmerbetreuung

J.Reize

MIKROPROZESSORTECHNIK 4

Herausgeber: R. Christiani

Inhaltsverzeichnis

S **Software**

H **Hardware**

Übungen

Prüfungsaufgaben

2.v.A.830803

Vorwort

Wenn von Mikroprozessoren gesprochen wird, dann löst dieses Wort sehr unterschiedliche Reaktionen aus. Für ganz unbedarfte Leute ist es ein Reizwort, das ganz automatisch mit dem Begriff „Jobkiller" und ähnlichen unerfreulichen Dingen in Verbindung gebracht wird. Bei etwas sachlicherer Betrachtungsweise erinnert man sich vielleicht daran, daß es so etwas wie Heim-Computer (sogenannte Personal-Computer) gibt, mit denen man am Feierabend hübsche Spielchen treiben oder gar Schach spielen kann. Vielleicht erinnert man sich auch daran, daß solche Dinger in manchen Büros stehen und von wichtig aussehenden Damen und Herren mit geheimnisvollen Zahlen gefüttert werden. Sicher hat dann auch irgendjemand mal behauptet, daß es Leute geben soll, die so etwas sogar programmieren können. Was immer es mit diesem „Programmieren" auch auf sich haben mag – jedenfalls ist das bestimmt so eine Art schwarzer Kunst, von der sich normale Sterbliche gefälligst fernzuhalten haben.

Die Tatsache, daß Sie sich in diesem Lehrgang etwas genauer über den Mikroprozessor informieren wollen zeigt, daß Sie über so primitive Betrachtungsweisen weit hinaus sind. Sicher haben Sie sogar eine ganz bestimmte Vorstellung, was man mit einem Mikroprozessor anstellen kann. Wir möchten sogar annehmen, daß Ihr Interesse noch viel weiter reicht: Sie wollen eine Anlage mit einem Mikroprozessor nicht als einen „schwarzen Kasten" ansehen, dem man in primitivem Englisch Anweisungen zum Rechnen oder zum Drucken gibt und der dann tatsächlich – wahrscheinlich mit Hilfe des elektrischen Stroms – die geforderten Aktionen vornimmt.

Sie möchten gern genau wissen, wie ein Mikroprozessor-System arbeitet und in welcher Form der Mikroprozessor seine Anweisungen entgegennimmt.

Ihnen die Grundlagen dieses Wissen zu vermitteln, das ist das Anliegen unseres Lehrgangs. Sie sollen dem Mikrocomputer ins Herz schauen und den Mikroprozessor verstehen lernen. Dazu müssen Sie ein wenig elektronisch denken können und die eigentliche Sprache des Mikroprozessors kennenlernen. Diese Sprache bezeichnet man als **Assembler**, oder in ihrer noch ursprünglicheren Form einfach als **Maschinensprache**.

Mit dem Assembler und der Maschinensprache ist es das gleiche wie mit anderen Fremdsprachen auch: Es hat keinen Zweck, eine solche Sprache allein aus einem klugen Lehrbuch lernen zu wollen. Man muß eine solche Sprache auch wirklich sprechen und sie anwenden, wenn man sie beherrschen will. Assembler und Maschinensprache spricht man mit einem Mikroprozessor-System, und deshalb gehört zu unserem Lehrgang ein solches System, an dem Sie Ihre jeweils erworbenen Kenntnisse sogleich ausprobieren können.

Das System zu unserem Lehrgang enthält einen sehr komfortablen Mikroprozessor, der die Bezeichnung Z 80 hat. (In seinem Heimatland Amerika spricht man das wie „sie äiti" aus.) Es soll und kann nicht die Absicht unseres einführenden Lehrgangs sein, Sie mit sämtlichen Möglichkeiten vertraut zu machen, die dieser Mikroprozessor bietet.

Der Z 80 hat allerdings eine ganz besondere Eigenschaft: Er ist abwärtskompatibel zu den in der Industrie sehr weit verbreiteten

Mikroprozessoren mit den Bezeichnungen 8080 und 8085. Das heißt: All' das Wissen, das Ihnen unser Lehrgang mit Hilfe des Z-80-Systems vermittelt, ist grundsätzlich auch auf Systeme anwendbar, in denen Mikroprozessoren mit den Bezeichnungen 8080 und 8085 verwendet werden. Wenn wir an der einen oder anderen Stelle unseres Lehrgangs einmal auf die komfortableren Möglichkeiten des Z-80-Mikroprozessors hinweisen oder von ihnen Gebrauch machen, dann werden wir Sie ausdrücklich darauf hinweisen.

Sie brauchen keineswegs zu fürchten, daß Sie wegen der hier genannten Beschränkung unseres Lehrgangs nur unvollkommen über den Mikroprozessor informiert werden. Am Ende des Lehrgangs verfügen Sie über das geistige Handwerkszeug, mit dem Sie sich sehr leicht auch mit dem höheren Komfort des Z-80-Mikroprozessors zurechtfinden.

Hardware

Hardware

Mikroprozessoren sind vorzugsweise Produkte der amerikanischen Industrie. Diese Tatsache bringt es mit sich, daß sich in der Umgangs-Fachsprache eine Menge englischer Ausdrücke eingebürgert hat. Man kann sich zwar bemühen, einen Teil dieser Ausdrücke durch entsprechende deutsche Wörter zu ersetzen. Das birgt jedoch die Gefahr in sich, bei Wortungetümen wie Viertopf-Zerknalltreibling (Vierzylinder-Explosionsmotor) oder Meuchelpuffer (Revolver) zu landen, oder aber sich einfach mit Fachkollegen nicht mehr verständigen zu können.

Wir wollen uns in diesem Lehrgang bemühen, dort, wo es sinnvoll erscheint, deutsche Ausdrücke zu verwenden und den oft nützlichen, entsprechenden englischen Ausdruck mit anzugeben. Wenn jedoch ein englischer Ausdruck fester Bestandteil der deutschen Fachsprache geworden ist, dann soll dieser Ausdruck beibehalten und seine Bedeutung eingehend erläutert werden.

Hardware (sprich: haadwär) ist „harte Ware", die man zum Beispiel bei einem Eisenwaren-Händler kauft. Man versteht darunter ursprünglich Montage-Bauelemente, wie Schrauben, Muttern usw. In die Computer-Fachsprache hat man dieses Wort in übertragener Bedeutung übernommen: Es kennzeichnet im Gegensatz zur Software alles das, was so hart ist, daß es beim Hinunterfallen klappert oder klirrt. Wir wollen darunter die elektrischen und elektronischen Bauelemente der Mikroprozessor-Technik verstehen.

Was kann ein Mikroprozessor?

Fragt man den Fachmann ganz unbefangen, was man denn nun eigentlich mit einem solchen Mikroprozessor machen kann – mit einiger Sicherheit bekommt man die Antwort: Alles. Und allenfalls dazu ein hintergründiges Lächeln.

Wir wollen mit dieser Antwort ein klein wenig vorsichtiger sein und lieber sagen: Fast alles. Denn glücklicherweise bleiben da einige Dinge, die ein unbeseeltes Produkt eben nicht kann. Diese Dinge können wir dann unter „fast" einordnen.

Spaß beiseite: Was kann er denn nun wirklich? Lassen Sie sich nicht verblüffen: Nichts kann er. Gar nichts.

Nehmen Sie (in Gedanken!) ein Vierfach-NAND-IC, speisen Sie es mit der vorgeschriebenen Spannung und schalten Sie zwischen den Ausgang eines der NAND-Glieder und den positiven Pol der Speisespannung eine Lumineszenzdiode. Jeden der beiden Eingänge beschalten Sie mit einem Widerstand und einem Schließer, der beispielsweise beim Öffnen einer Tür betätigt wird. Im Bild H2.1 haben wir die Schaltung skizziert.

Hier haben wir eins der billigsten digitalen Bauelemente verwendet und mit verschwindend kleinem, zusätzlichen Aufwand eine Schaltung aufgebaut, die wirklich etwas „kann".

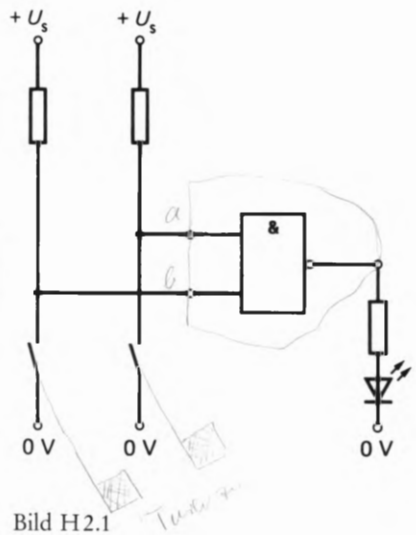


Bild H2.1
Ein NAND-Glied mit einer einfachen Beschaltung liefert bereits eine technisch brauchbare Schaltung.

Aufgabe H2.1

Was „kann" die im Bild H2.1 dargestellte Schaltung?

Jeder der beiden Schließer wird beim Öffnen einer von zwei Türen betätigt.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü1.

Der Mikroprozessor ist – wie das Vierfach-NAND-IC – nichts anderes als ein digitales Bauelement. Mit zwei Unterschieden: Er ist ein klein wenig teurer (50- bis 100-facher Preis!). Und er ist mit einer ähnlich einfachen Beschaltung wie das Vierfach-NAND-IC keinesfalls zu bewegen, ein ähnliches Können an den Tag zu legen.

Niemand kann Ihnen die Frage verübeln: Ja, was soll denn der Unsinn? Der Mikroprozessor ist ein digitales Bauelement, ein relativ teures dazu. Nach Auskunft der Fachleute soll er fast alles können – und jetzt heißt es, man könne ihn nicht einmal wie ein einfaches NAND-Glied verwenden!

Um es vorwegzunehmen: Selbstverständlich kann man **mit** einem Mikroprozessor auch Verknüpfungen von zwei Signalen vornehmen. In einem der nächsten Abschnitte werden Sie sich in einigen Versuchen mit Ihrem Mikroprozessor-System davon überzeugen können.

Das Bauelement Mikroprozessor ist eine digitale integrierte Schaltung in einem Dual-In-Line-Gehäuse mit sehr häufig 40 Anschlüssen, wie es das Bild H2.2 zeigt. Die Innenschaltung ist so ausgelegt, daß binäre Signale auf mannigfaltige Weise manipuliert werden können. Welche Manipulationen man vornehmen kann, wird eines der Hauptthemen dieses Lehrgangs sein. Gerade diese Möglichkeit, ganz nach Wunsch eine solche Vielzahl unterschiedlicher Manipulationen (z. B. Verknüpfungen) von binären Signalen vornehmen zu können, macht den Mikroprozessor so sehr interessant und vielseitig. Nur: Wenn man dem Mikroprozessor nicht sagt, was man von ihm verlangt, welche Manipulation er also gerade vornehmen soll, dann kann er natürlich auch nichts tun. Dann ist er dumm und unbrauchbar.

Bereits diese einfache Überlegung zeigt, daß der Mikroprozessor erst dann arbeitsfähig wird, wenn man ihm außer den zu verarbeitenden Signalen auch Anweisungen zukommen läßt, die ihm sagen, wie er die Signale verarbeiten soll. Eine Anweisung besteht aus **Befehlen**, (vgl. Bild H3.1). Eine zeitlich aufeinanderfolgende Reihe solcher Befehle wird als **Programm** bezeichnet.

Bild H2.2
Ein Mikroprozessor ist eine integrierte digitale Schaltung im Dual-In-Line-Gehäuse.

In der Fachsprache wird das im Bild H2.2 dargestellte Bauelement Mikroprozessor einfach als **CPU** bezeichnet. Er ist die Abkürzung der englischen Bezeichnung **C***entral* **P***rócessing* **U***nit* (sprich: ssentrel prósessing junit), die am besten mit der ebenfalls üblichen deutschen Bezeichnung **Zentraleinheit** übersetzt wird.

Sie sehen: Die Mikroprozessor-CPU muß unbedingt mit irgendwie gearteten anderen Bauelementen kombiniert werden, damit eine sinnvolle Signalverarbeitung überhaupt möglich ist. Arbeitsfähig wird die Sache erst dann, wenn der Mikroprozessor zum Mikroprozessor-System erweitert wird. Man kann zwar einige der zusätzlichen Bau-

elemente mit der CPU zu einem „Ein-Chip-Mikroprozessor" integrieren, aber ganz ohne zusätzliche Bauelemente geht's im allgemeinen nicht. Ein solches System nennt man dann **Mikrocomputer** (sprich: mikrokompjuhter).

Dies ist ein ebenso vielversprechender wie irreführender Name. Das englische Wort *computer* bedeutet ganz wörtlich Rechner. Rechnen aber ist – abgesehen von den einfachen Grundrechnungsarten – eine Tätigkeit, die der Mikrocomputer gar nicht gern mag. Gewiß, man kann ihm auch befehlen, eine Wurzel zu ziehen oder eine Gleichung mit trigonometrischen Funktionen auszurechnen. Das ist jedoch eine Aufgabe für ihn, bei der er fast zu dampfen beginnt; gerade wie Sie, wenn Sie im Zeitalter des Taschenrechners etwa die Wurzel aus 237,5 „zu Fuß" ziehen sollten.

Wir wollen es ehrlich und laut sagen: Im Rechnen sind die meisten Taschenrechner einem Mikrocomputer weit überlegen. Zwar enthalten auch Taschenrechner eine Art Mikroprozessor; dann aber einen solchen, der durch seinen inneren Aufbau speziell auf Rechnen gedrillt wurde.

Wenn Sie jetzt kurz davor stehen, die ganze Angelegenheit Mikroprozessor in das Grab des Vergessens zu versenken, dann müssen wir Sie dringend bitten, die Ausführung Ihres Entschlusses noch ein paar Minuten zu verschieben.

Gewiß: Für eine primitive NAND-Verknüpfung braucht der Mikroprozessor mindestens noch einen irgendwie gearteten Befehlsgeber und muß damit zum Mikrocomputer ausgebaut werden. Und rechnen mag sogar solch ein komplexes System nicht gern. Aber wer sagt denn, daß die Fähigkeiten des Mikrocomputers bei der NAND-Verknüpfung von zwei Signalen ein Ende haben?

Im Gegenteil: Hier fängt die Sache an interessant zu werden. Das gleiche System nämlich, das man für eine einzelne NAND-Verknüpfung höchst unwirtschaftlich einsetzen würde, kann ohne wesentlichen Mehraufwand an Bauelementen, gesteuert von entsprechenden Befehlen, die gleiche Aufgabe lösen wie sehr komplizierte Schaltnetze und Schaltwerke mit vielen Eingangs- und Ausgangssignalen.

Können Sie sich ein digitales Schaltwerk zur Steuerung mehrerer Personenaufzüge in einem Hochhaus vorstellen? Aufgebaut mit normalen TTL-ICs ergibt sich da sicher eine sehr umfangreiche Elektronik mit vielen Verknüpfungsgliedern und Flipflops. Eine solche Steuerung ist geradezu ein Leckerbissen für ein Mikroprozessor-System, dessen Hardware-Aufwand wesentlich kleiner ist als der bei einer Lösung mit normalen ICs.

Wir gehen noch einen Schritt weiter. Nehmen Sie an, Sie selbst hätten die Aufzugsteuerung mit den vielen TTL-ICs entwickelt und auch die Leiterplatten dazu entworfen. Und jetzt kommt jemand und sagt Ihnen: Ach was, ich verzichte auf die Aufzüge. Bauen Sie die Elektronik doch bitte so um, daß sie zur Steuerung der Klimaanlage des Hochhauses verwendet werden kann.

Als Elektroniker würden Sie dieses Ansinnen vermutlich nur mit einem müden Lächeln quittieren und sich bestenfalls daran machen, eine völlig neue Elektronik zu entwickeln. Die Hardware müßte in neuer und ganz anderer Form aufgebaut werden.

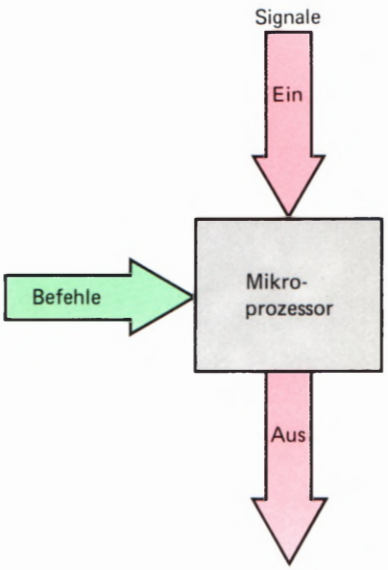


Bild H3.1
Der Mikroprozessor ist erst dann arbeitsfähig, wenn ihm Befehle für die Signalverarbeitung zugeführt werden.

Das Problem sähe ganz anders aus, wenn Sie zur Steuerung der Aufzüge ein Mikroprozessor-System verwendet hätten. Wir wollen nicht behaupten, daß dieses System ganz unverändert auch zur Steuerung der Klimaanlage dienen könnte. Die Grundkonzeption jedoch müßte keineswegs unbedingt umgestellt werden. Vielleicht ließen sich sogar einige Leiterplatten unverändert übernehmen. Geändert werden müßten die Befehle. Man müßte ein neues Programm entwickeln. Und das ist Schreibtischarbeit, zu der man keinen Lötkolben braucht.

Grundsätzlich ist mit einem Mikroprozessor-System alles das machbar, was sonst mit einer Schaltung digitaler Einzel-ICs aufgebaut würde. Zu bedenken ist höchstens, daß ein Mikroprozessor etwas langsamer arbeitet als eine konventionelle Schaltung. Normalerweise ist dieser Gesichtspunkt jedoch von untergeordneter Bedeutung und muß nur dann berücksichtigt werden, wenn eine sehr schnelle Signalverarbeitung verlangt wird.

Schwerwiegender sind rein wirtschaftliche Überlegungen bei dem Entscheid, ob man ein Problem mit einem Mikroprozessor löst, oder ob man eine „Hardwired Logic" (sprich: haadweierd lodschik) bevorzugt: Eine festverdrahtete Schaltung mit herkömmlichen digitalen Standard-ICs.

Wir können bei diesen Überlegungen den Hobby-Elektroniker außer Betracht lassen. Sein Ziel ist ja nicht klingende Münze für jede Entwicklungs-Stunde.

Im kommerziellen Bereich muß aber sehr wohl bedacht werden, daß sich beim Einsatz des Mikroprozessors die Entwickungskosten weitgehend auf die Erstellung der Befehlsfolge, auf die **Software**, verlagern.

Wir brauchen hier keine Abgrenzung der Wirtschaftlichkeit von Mikroprozessor und konventioneller Technik anzustreben. Wenn Ihnen am Ende des Lehrgangs der Mikroprozessor ein vertrautes Bauelement geworden ist, dann fällt Ihnen der Entscheid ebenso leicht wie der, ob man bei einem sonst reinen NAND-Schaltnetz ein Vierfach-NOR-IC einsetzt oder ob man die NOR-Funktion besser durch mehrere NAND-Glieder verwirklicht.

Erahnen läßt sich die Wirtschaftlichkeit von Mikroprozessoren bereits, wenn man die unsinnige Überlegung anstellt, ob für eine einzelne NAND-Funktion ein Mikroprozessor einzusetzen ist. Ab einem bestimmten Umfang der gestellten Aufgabe wird sich der Einsatz eines Mikroprozessors fast immer lohnen; besonders dann, wenn nach Abschluß der Schaltungsentwicklung Änderungen der Aufgabenstellung zu erwarten sind. Solche Änderungen lassen sich durch Ändern der Befehle im Programm berücksichtigen.

Fassen wir zusammen: Der Mikroprozessor ist ein elektronisches Bauelement, das erst im Rahmen eines Mikroprozessor-Systems arbeitsfähig ist. Ein solches System nennt man Mikrocomputer. Der Mikrocomputer bedarf einer Reihe aufeinanderfolgender Befehle, eines Programms, um eine bestimmte Aufgabe lösen zu können. Er ist nahezu universell einsetzbar. Die Einsatzmöglichkeit wird im wesentlichen durch wirtschaftliche Gesichtspunkte bestimmt.

Inbetriebnahme des Mikro-Professors

Einen Professor kann man ernennen, wenn er die notwendige Qualifikation aufweist. Man kann vor einer Prüfung vor ihm zittern und man kann ihn (wenn man zufällig das richtige Geschlecht hat) sogar heiraten. Aber daß man ihn in Betrieb nehmen kann, das ist schon eine recht seltene Gelegenheit. Sie können – und sollen! – es trotzdem tun, denn Sie haben es mit einem winzig kleinen Professor zu tun, mit einem **Mikro**-Professor nämlich, der allerdings eine nicht zu unterschätzende Intelligenz und sicher ein schier unbegrenztes Lernvermögen hat.

Diese beiden Eigenschaften haben dem Mikroprozessor-System, das wir unserem Lehrgang zugrundelegen, seinen werbeträchtigen Namen gegeben. – Wir wollen uns hier keine Gedanken darüber machen, ob ein solches System wirklich intelligent ist und lernen kann. (Das ist eine Frage der Definition, um die sich schon sehr kompetente Leute heftig gestritten haben.) Für Sie ist es wahrscheinlich viel interessanter, das System endlich einmal einzuschalten, nachdem wir Sie vorab mit einer Menge (hoffentlich nicht allzu langweiliger) Theorie eingedeckt haben.

Wie sich der Mikro-Professor meldet

In der Verpackung Ihres Mikroprozessor-Systems finden Sie außer dem in einer Art Buchhülle angeordneten System selbst eine Stromversorgungs-Einheit und ein Begleitbuch.

Auf den Inhalt dieses Begleitbuchs, das Ihnen nach dem Durcharbeiten unseres Lehrgangs ein wertvoller Begleiter sein kann, wollen wir zunächst nicht zurückgreifen. Das Buch setzt teilweise ein Grundwissen voraus, über das Sie vermutlich jetzt noch nicht verfügen, und das wir Ihnen in unserem Lehrgang vermitteln wollen.

Wichtiger ist im Augenblick die Stromversorgungs-Einheit, die Sie mit der Netzsteckdose verbinden können. Am Stecker der aus der Einheit herausgeführten Verbindungsleitung steht eine Gleichspannung von etwa 9 V zur Verfügung, die auf der Leiterplatte des Systems mit einem Stabilisierungs-IC zu einer Gleichspannung von 5 V umgesetzt wird.

Im Bild H5.1 ist dargestellt, mit welcher Polarität die 9-V-Gleichspannung am Stecker der Stromversorgungs-Einheit zur Verfügung steht. Diese Feststellung ist wichtig, weil es eine Menge ähnlicher Stromversorgungen für 9-V-Gleichspannung mit gleichem oder ähnlichem Stecker gibt. Bei der Verwendung der zu Ihrem System gehörenden oder ähnlichen Stromversorgungs-Einheiten muß jeweils sorgfältig geprüft werden, ob die Polarität der Gleichspannung am Stecker zum entsprechenden Gerät paßt. Bei falscher Polarität der Gleichspannung werden die Halbleiter des Geräts mit großer Wahrscheinlichkeit zerstört.

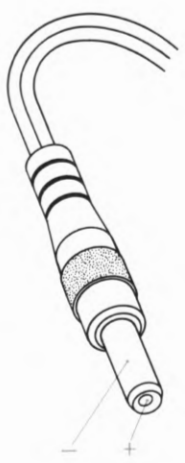


Bild H5.1
Bei dem zur Stromversorgung unseres Systems verwendeten Anschlußstecker steht die 9-V-Gleichspannung mit der hier angegebenen Polarität zur Verfügung.

Am Ende dieses Lehrgangs finden Sie auf der Seite T1 eine Darstellung Ihres Systems, in die wir nur die wichtigsten Teile eingetragen haben. Sie finden in der rechten, oberen Ecke eine mit „9 V“ bezeichnete

13 V Leerlauf
9,7 V am 20-2 (185mA)

H
6

Buchse, die zum Stecker der Stromversorgungs-Einheit paßt. Orientieren Sie sich anhand unserer Darstellung, wo sich diese Buchse auf Ihrem System befindet.

Verbinden Sie die Stromversorgungs-Einheit mit der Netzsteckdose und stecken Sie den 9-V-Stecker in die zugehörige Buchse Ihres Systems! – Beobachten Sie die sechs mit einer roten Scheibe abgedeckten Sieben-Segment-Anzeigestellen auf Ihrem System!

Die Anzeige ist nicht sehr hell. Sie können sie aber gut ablesen, wenn Sie eine allzu starke Beleuchtung der Anzeige vermeiden. Sie erkennen, wie die im Bild H6.1 gezeigte Schrift von rechts nach links in die Anzeige hineingeschoben wird. – Welche Bewandtnis es mit den beiden rechts etwas abgesetzten Anzeigestellen hat, wird später deutlich.

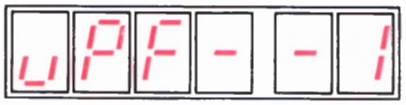


Bild H6.1
Nach dem Einschalten des Systems und nach dem Betätigen der RESET-Taste erscheinen in der Anzeige die hier dargestellten Zeichen als System-Meldung.

Die jetzt in der Anzeige stehenden Zeichen nennen wir die **System-Meldung**. Das ganz links stehende Zeichen bedeutet den griechischen Buchstaben μ (besser kann es eine Sieben-Segment-Anzeige nicht!). Er steht als Abkürzung für das Wort MIKRO, so daß die drei Zeichen links das Wort MIKRO-PROFESSOR abkürzen. Die Zeichen --1 bedeuten einfach eine Typen-Nummer.

Mit der nach dem Einschalten erscheinenden System-Meldung zeigt das System, daß es zunächst einmal ordnungsgemäß arbeitet, und daß es nun auf weitere Anweisungen wartet. Sie erinnern sich: Ohne solche Anweisungen ist der Mikroprozessor ganz dumm und kann überhaupt nichts tun.

Die einzige Möglichkeit, dem System Anweisungen zukommen zu lassen, besteht mit Hilfe des Tastenfeldes im unteren Drittel des Systems. Sie werden jedoch gleich sehen, daß in Ihrem System eine Menge Anweisungen verborgen sind, die lediglich wiederum mit Anweisungen über das Tastenfeld aktiviert werden müssen.

Ehe Sie überhaupt wissen, was das System „kann" und wie man dieses Können sinnvoll ausnutzt, sollen Sie vorab einmal feststellen, daß sich seine Fähigkeiten durchaus nicht mit der System-Meldung erschöpfen. Dazu sollen einige der im System verborgenen Anweisungen aktiviert werden.

Im Bild H7.1 haben wir noch einmal in verkleinerter Form das Tastenfeld des System abgebildet. Dabei haben wir einige Tasten grün markiert und in der Reihenfolge unter dem Tastenfeld numeriert, in der Sie sie nacheinander betätigen sollen.

Versuch H6.1

Ein Demonstrations-Programm

Sorgen Sie dafür, daß das auf der Seite T1 mit (Demonstration) bezeichnete EPROM so in der zur Adresse 2000 gehörenden Fassung steckt, wie es auf der Seite T1 dargestellt ist. Dieses EPROM kann Ihrer Materialsendung lose beiliegen.

Betätigen Sie in der im Bild H7.1 dargestellten Reihenfolge nacheinander die Tasten:

ADDR, 2, 3, 0, 0

Bei jeder Tasten-Betätigung gibt der Lautsprecher – ähnlich wie eine moderne Registrierkasse – einen kurzen Ton von sich, mit dem das System meldet, daß es die Tasten-Anweisung richtig verstanden hat. Beobachten Sie während der Eingabe die Anzeige! Sie sehen: Es tut

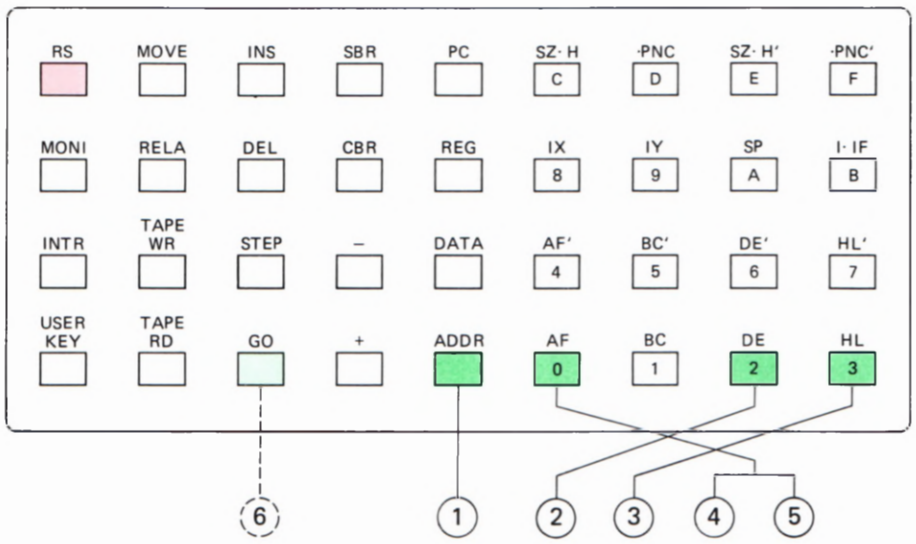


Bild H7.1
Tastenfeld des Mikro-Professors. Zum Start des Demonstrations-Programms im Versuch H6.1 müssen die hier farbig markierten Tasten in der angegebenen Reihenfolge betätigt werden. – Betätigen Sie die Taste GO erst dann, wenn im Text darauf hingewiesen wird.

sich was. Was es mit den gezeigten Zeichen auf sich hat, wollen wir hier noch nicht erläutern. Jedenfalls erkennen Sie, daß die nach der Betätigung der Taste ADDR eingetasteten Ziffern 2, 3, 0 und 0 von der (von links gezählt) vierten Anzeigestelle her nach links in die Anzeige eingeschoben werden. Die beiden rechten Anzeigestellen brauchen uns zunächst nicht zu interessieren.

Wir können Ihnen bereits hier zwei wichtige Hinweise für die Bedienung Ihres Systems geben. Zunächst:

> Unabhängig von vorher eingegebenen oder ausgeführten Anweisungen bewirkt die im Tastenfeld rot markierte Taste RS (RESET) immer das Erscheinen der System-Meldung.

Das englische Wort *reset* (sprich: riessett) bedeutet Zurücksetzen. Gemeint ist hier damit, daß das System wieder in seinen Anfangszustand gebracht wird, in dem es neue Anweisungen erwartet. Wenn Ihnen also z. B. bei einer Tasten-Eingabe ein Fehler unterlaufen ist, oder wenn später ein von Ihnen eingegebenes Programm das System „in die Wüste laufen" läßt, so daß scheinbar gar nichts mehr geht, dann ist die RESET-Taste RS eine unfehlbare Rettung, mit der das System wieder „auf den Teppich" geholt werden kann.

Wenn Sie bei den Eingaben zu unserem Versuch gleich am Anfang statt der Taste ADDR eine falsche Taste erwischt haben, dann betätigen Sie zweckmäßig die RS-Taste und beginnen die Eingabe von neuem.

Wenn Sie sich bei der Eingabe der nachfolgenden Ziffern vertun, dann brauchen Sie nicht die ganze Eingabe nach Betätigung der RS-Taste zu wiederholen. Sie brauchen nur der Reihe nach die notwendigen Ziffern noch einmal in der richtigen Reihenfolge einzugeben. Die falschen Ziffern werden nacheinander links aus der Anzeige hinausgeschoben. Machen Sie einen Versuch. Geben Sie nacheinander folgende Ziffern ein und beobachten Sie die Anzeige:

4, 5, 6 (hoppla, das war ja falsch! Also:) 2, 3, 0, 0

H
8

Am Ende dieser Eingabe stehen an den linken vier Anzeigestellen wie vorher richtig die Ziffern 2300.

Die beschriebene Korrektur-Möglichkeit gilt allerdings nur für die auf dem Tastenfeld im rechten Teil weiß markierten Tasten. Wenn Sie es irrtümlich mit einer der grauen Tasten versucht haben, dann passieren womöglich unvorhersehbare Dinge. In diesem Fall müssen Sie die RS-Taste betätigen und die ganze Eingabe neu vornehmen.

Nun aber zurück zu unserem Versuch mit dem Demonstrations-Programm. In der Anzeige stehen jetzt die Zeichen

2.3.0.0. XX

Die hier angegebenen Zeichen X für die Anzeige an den beiden rechten Anzeigestellen bedeuten, daß wir nicht wissen, welche Zeichen an diesen beiden Stellen Ihrer Anzeige stehen, und daß diese beiden wirklich angezeigten Zeichen bedeutungslos sind.

Betätigen Sie jetzt als 6. Eingabe die Taste GO! – Welche Wirkung diese Eingabe hat, brauchen wir nicht eigens zu beschreiben; es ist nicht zu überhören.

Aufgabe H8.1

Überlegen Sie, wie Sie die mit der Taste GO gestartete Aktivität Ihres Systems wieder anhalten können! (Netzstecker ziehen gilt nicht!)

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü2

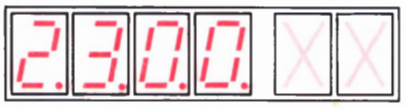


Bild H8.1
Die Eingabe-Möglichkeit in das Adreßfeld der Anzeige meldet das System durch Dezimalpunkte an den linken vier Stellen.

Mit dem soeben durchgeführten Versuch haben Sie Anweisungen aktiviert, die (auf zunächst geheimnisvolle Weise) im System verborgen sind. Am Ende dieses Abschnitts wollen wir Ihnen zeigen, wie Sie selbst Anweisungen für ein Programm in das System eingeben können, das sich anschließend wieder mit der Taste GO starten läßt.

Das englische Wort *go* (sprich: gou) bedeutet wörtlich gehen oder auch den Befehl: Geh'! Über die mit GO bezeichnete Taste gibt man dem System den Befehl Geh'! und meint damit: Lauf los und führe die Anweisungen aus, die Dir auf irgendeine Weise vorher übermittelt worden sind.

Im vorangegangenen Versuch haben Sie vor der Taste GO noch der Reihe nach die Tasten ADDR und 2300 betätigt. Diese Tasten haben offenbar dafür gesorgt, daß ganz bestimmte, im System verborgene Anweisungen nach dem Betätigen der Taste GO ausgeführt werden. Tatsächlich ist die Ziffernfolge 2300 für diese Anweisungen zuständig. Mit der vorangegangenen Taste ADDR haben Sie bewirkt, daß Ihr System die Ziffernfolge 2300 als Kommando für die angesprochenen Anweisungen interpretiert. Sie werden noch sehen, daß Ihr System solche Ziffern auch ganz anders interpretieren kann, z. B. als Befehle oder auch als Buchstaben.

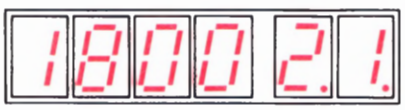


Bild H8.2
Bei der Daten-Eingabe erscheinen Dezimalpunkte an den rechten beiden Stellen der Anzeige.

Die Ziffernfolge 2300 ist die Nummer der ersten Anweisung des im System verborgenen Programms, das Sie soeben haben ablaufen lassen. Solche Nummern werden im Laufe unseres Lehrgangs noch eine wichtige Rolle spielen, und wir wollen bereits jetzt dafür den in der

Rechnertechnik üblichen Namen einführen: Man bezeichnet diese Nummern als **Adressen.**

In unserem System werden eingegebene Ziffernfolgen durch die vorangestellte Betätigung der Taste ADDR als Adressen gekennzeichnet. ADDR ist die Abkürzung des englischen Wortes **ADDR***ess*. Adressen werden in unserem System stets vierstellig eingegeben; sie erscheinen an den linken vier Stellen der Anzeige.

Wenn die ADDR-Taste betätigt wird, dann zeigt das System seine Bereitschaft zur Annahme von Adressen dadurch an, daß in den linken vier Anzeigestellen Dezimalpunkte erscheinen. Im Bild H8.1 ist die Anzeige nach der Eingabe ADDR 2300 dargestellt. Die beiden Zeichen XX erscheinen nicht wirklich in der Anzeige. Sie deuten an, daß sich die an den beiden rechten Anzeigestellen erscheinenden Ziffern in diesem Fall nicht vorhersagen lassen.

Wir wollen Ihnen jetzt zeigen, wie Sie in Ihr System Anweisungen eingeben können, die nicht von vornherein vorhanden sind.

Versuch H9.1

Eine Tick-Tack-Uhr mit Ziffernanzeige

Schließen Sie Ihr System an die Stromversorgung an. Wenn aus irgendeinem Grund in der Anzeige nicht die System-Meldung steht, dann betätigen Sie bitte die RESET-Taste RS.

Teilen Sie Ihrem System mit, daß die erste Anweisung des nun folgenden Programms bei der Adresse 18Ø0 stehen soll. (Wir wollen ab jetzt – wie in der Rechnertechnik üblich – die Ziffern 0 zur Unterscheidung vom großen Buchstaben O im Text schräg durchstreichen.) Betätigen Sie der Reihe nach folgende Tasten:

ADDR, 1, 8, 0, 0

Nach dieser Eingabe zeigt die Anzeige

1.8.0.0. XX (Die Zeichen XX bedeuten wieder nicht vorhersagbare Ziffern.)

Auf der vorhergehenden Seite wurde bereits angedeutet, daß bei einem Mikroprozessor nicht nur Adressen, sondern auch Befehle als Ziffern eingegeben werden. Im Bild H9.1 haben wir die Ziffern aufgelistet, die die Befehle für die Programm-Anweisungen bedeuten. Wenn Sie jetzt diese Ziffern über das Tastenfeld eingeben wollen, dann müssen Sie dem System zunächst mitteilen, daß die folgenden Ziffern nicht mehr Adressen bedeuten, sondern eben Befehle.

Man bezeichnet solche Befehls-Ziffern als **Daten**. – Das Tastenfeld enthält genau über der ADDR-Taste eine mit DATA bezeichnete Taste. Beobachten Sie die Anzeige und betätigen Sie diese Taste! Geben Sie gleich danach die beiden Ziffern 2 und 1 für den ersten Befehl ein:

DATA, 2, 1

Im Bild H8.2 haben wir die Anzeige nach dieser Eingabe dargestellt. Sie haben bemerkt, daß nach dem Betätigen der DATA-Taste die vier Dezimalpunkte im Adressenfeld der Anzeige verschwunden und dafür an den rechten beiden Anzeigestellen erschienen sind. Diese beiden

Adresse	1. Eingabe	2. Eingabe	3./4. Eingabe
18ØØ	21	Ø8	ØØ
18Ø3	ØE	ØF	
18Ø5	CD	E4	Ø5
18Ø8	CD	14	18
18ØB	Ø6	5E	
18ØD	CD	2C	18
1810	10	FB	
1812	18	EC	
1814	Ø6	Ø3	
1816	11	Ø2	1A
1819	21	47	18
181C	37		
181D	1A		
181E	CE	ØØ	
1820	27		
1821	12		
1822	96		
1823	38	Ø1	
1825	12		
1826	3F		
1827	2B		
1828	1B		
1829	10	F2	
182B	C9		
182C	C5		
182D	11	Ø2	1A
1830	21	Ø3	19
1833	Ø6	Ø3	
1835	1A		
1836	CD	78	Ø6
1839	1B		
183A	10	F9	
183C	DD	21	Ø3
183F			19
1840	CD	24	Ø6
1843	C1		
1844	C9		
1845	24		
1846	60		
1847	60		

Bild H9.1
Die in die drei rechten Spalten dieser Auflistung eingetragenen Ziffernpaare müssen für den Versuch H9.1 in Ihr System eingegeben werden.

Anzeigestellen sind für Befehls- und andere Daten in unserem System zuständig, und die Dezimalpunkte zeigen an, daß das System zur Aufnahme eingetasteter Daten bereit ist.

Für die Eingabe von Daten gelten die gleichen Regeln, die wir vorher für die Eingabe von Adressen genannt haben: Die Daten werden von rechts nach links in das Datenfeld der Anzeige hineingeschoben. Bei einer irrtümlichen Eingabe brauchen nur die richtigen Daten nachgeschoben zu werden. Gültig sind jeweils die letzten beiden Eingaben. Wir wollen festhalten:

> In unserem System werden Adressen immer vierstellig eingegeben. Adressen erscheinen an den vier linken Anzeigestellen.
>
> Daten werden immer zweistellig eingegeben. Daten erscheinen an den beiden rechten Anzeigestellen.

Die beiden von Ihnen bisher eingegebenen Ziffern 21 stellen noch keinen vollständigen Befehl dar. Im Bild H9.1 haben wir (mit einer Ausnahme) die zu einem Befehl gehörenden Ziffernpaare jeweils in einer Zeile angeschrieben. Sie erkennen, daß bei dem in unserem System verwendeten Mikroprozessor Befehle aus unterschiedlich vielen Ziffernpaaren bestehen können.

Der erste Befehl des hier einzugebenden Programms beginnt bei der Adresse 18ØØ; bei dieser Adresse steht das erste Ziffernpaar dieses ersten Befehls. Das zweite Ziffernpaar dieses Befehls steht bei der Adresse 18Ø1. Wir haben der Einfachheit halber diese Adresse nicht eigens angeschrieben. Wenn Sie von der Adresse 18ØØ des ersten Ziffernpaares aus weiterzählen, dann erkennen Sie leicht, daß das erste Ziffernpaar des zweiten Befehls bei der Adresse 18Ø3 stehen muß.

Sie müssen jetzt – ausgehend von der im Bild H8.2 dargestellten Anzeige – das zweite Ziffernpaar des ersten Befehls eingeben (Ø8). Es muß bei der Adresse 18Ø1 eingetragen werden. Eigentlich müßte jetzt wieder die Taste ADDR betätigt werden, damit diese Adresse angewählt werden kann; anschließend müßte zur Eingabe der Daten Ø8 vorher wieder die Taste DATA gedrückt werden.

Unser System macht es uns viel leichter. Betätigen Sie der Reihe nach folgende Tasten und beobachten Sie dabei die Anzeige:

+, 0, 8

(Die Taste + finden Sie gleich links neben der Taste ADDR.) Sie stellen fest, daß sich die Adresse im Adreßfeld der Anzeige nach dem Betätigen der Taste + automatisch um eins auf 18Ø1 erhöht; die Dezimalpunkte blieben im Datenfeld der Anzeige stehen und es erschien zunächst die Anzeige

1801 X.X.

Anschließend wurde das zweite Ziffernpaar des ersten Befehls von rechts nach links in das Datenfeld eingeschoben.

Das Prinzip der Eingabe der zu den einzelnen Befehlen gehörenden Ziffernpaare, die bei jeweils aufeinanderfolgenden Adressen stehen, ist nun deutlich: Die Eingabe erfolgt nach dem Schema

Erste Ziffer, zweite Ziffer, +, erste Ziffer, zweite Ziffer, +...

bzw.:

Ziffernpaar, +, Ziffernpaar, +, Ziffernpaar, +... usw.

Über der Taste + finden Sie übrigens eine mit – bezeichnete Taste, die Sie versuchsweise einmal betätigen können. Sie werden feststellen, daß diese Taste bei jeder Betätigung die Adresse im Adreßfeld der Anzeige um eins erniedrigt; die Eingabemöglichkeit von Daten in das Datenfeld bleibt dabei unbeeinflußt. Wenn Ihnen also nach dem Betätigen der Taste + einfällt, daß Sie bei der vorhergehenden Eingabe ein falsches Ziffernpaar eingegeben haben, dann ist nichts verloren. Sie brauchen nur die Taste – zu betätigen und sehen dann im Datenfeld, welches Ziffernpaar sie bei dieser Adresse eingegeben hatten. Bei Bedarf können Sie dann dieses Ziffernpaar korrigieren und nach Betätigung der Taste + das nächstfolgende Ziffernpaar eingeben.

Tasten Sie jetzt nach den hier beschriebenen Eingabe-Regeln die im Bild H9.1 aufgelisteten Ziffernpaare für das Uhren-Programm ein! Kümmern Sie sich bitte zunächst nicht darum, daß bei den Daten die höchste Ziffer nicht wie gewohnt die Ziffer 9 ist, sondern daß im weißen Ziffernfeld der Eingabe-Tastatur auf die Ziffer 9 noch die Ziffern A bis F folgen! Wir werden uns mit dieser merkwürdigen Tatsache noch näher beschäftigen.

Kontrollieren Sie während der Eingabe ab und zu, ob Sie das jeweilige Ziffernpaar bei der richtigen Adresse eingeben! Wenn die Zuordnung von Daten und Adressen nicht mehr mit der Liste im Bild H9.1 übereinstimmt, dann können Sie mit Hilfe der Taste – die Adressen bis zu einem Ziffernpaar erniedrigen, das wieder bei der richtigen Adresse steht. Ab dann können Sie die Daten dann neu eingeben.

Nach der Eingabe des letzten Ziffernpaares (6Ø) bei der Adresse 1847 brauchen Sie die Taste + nicht noch einmal zu betätigen. Rufen Sie statt dessen mit der RESET-Taste RS die System-Meldung auf. – Das System ist jetzt für den Ablauf des Uhren-Programms vorbereitet.

Ehe die Uhr gestartet wird, muß sie natürlich auf die richtige Zeit gestellt werden. Die Start-Zeit wird bei den Adressen 1AØØ, 1AØ1 und 1AØ2 abgelegt. Diese Adressen enthalten wieder die für Zahlenangaben merkwürdigen Ziffern A, aber das soll Sie ja im Augenblick nicht beunruhigen.

Im Bild H11.1 haben wir schematisch dargestellt, wie die Start-Zeit in Ihrem System abgelegt wird. Stellen Sie sich einfach vor, die drei dargestellten Kästen seien eine Art von Schubladen, die mit den angegebenen Adressen numeriert sind. In jeder der Schubladen wird ein Ziffernpaar abgelegt, das der Stunde, der Minute oder der Sekunde der Start-Zeit entspricht. – Die hier gewählte Art der Darstellung für die Unterbringung von Ziffernpaaren in „Schubladen" unseres Systems werden wir im Laufe unseres Lehrgangs noch recht häufig verwenden.

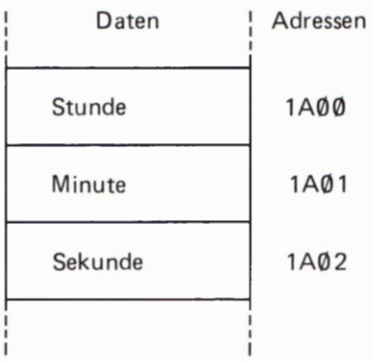


Bild H11.1
Die Start-Zeit der Uhr wird bei den Adressen 1AØØ bis 1AØ2 in Ihrem System abgelegt.

H
12

Aufgabe H12.1

Wir wollen annehmen, Ihre programmierte Uhr solle um 18 Uhr, 8 Minuten, Ø Sekunden loslaufen.

Überlegen Sie bitte, welche Eingaben Sie über das Tastenfeld Ihres Systems vornehmen müssen, damit die Uhr bei dieser Start-Zeit loslaufen kann.

Ein Hinweis: Sofort nach dem Start des Programms schaltet Ihre Uhr um eine Sekunde weiter! Sie wartet mit diesem Weiterschalten also nicht erst den Ablauf einer Sekunde ab.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü2.

Tasten Sie nun ab der Adresse 1AØØ entsprechend dem als Lösung der vorhergehenden Aufgabe gefundenen Schema die Zeit ein, mit der Ihre Uhr loslaufen soll. Beachten Sie, daß die Stunden, Minuten und Sekunden jeweils als Ziffern**paare**, also zweistellig eingegeben werden müssen!

Nach der Eingabe der zweiten Sekunden-Ziffer können Sie mit der RESET-Taste RS die System-Meldung aufrufen. Notwendig ist das nicht unbedingt.

Gerade wie beim Demonstrations-Programm im Versuch H6.1 müssen Sie jetzt zunächst dem System die Start-Adresse mitteilen, bei der es das erste Ziffernpaar der ersten Anweisung findet. Betätigen Sie die folgenden Tasten:

ADDR, 1, 8, 0, 0

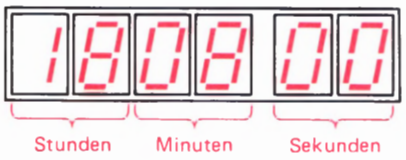


Bild H12.1
So wird die Uhrzeit in der Anzeige Ihres Systems beim Versuch H9.1 angezeigt.

Wenn jetzt Ihre Vergleichs-Uhr die in Ihr System eingegebene Zeit erreicht hat, dann können Sie die Taste GO betätigen und in der Anzeige Ihres Systems erscheint die voreingestellte Uhrzeit (Bild H12.1). Bei jedem Wechsel der Sekunde hören Sie im Lautsprecher ein Knacken, mit dem das Ticken einer großen Uhr imitiert wird.

Ihre nun programmierte Uhr ist zwar eine Quarzuhr, aber sie muß trotzdem nicht unbedingt genau gehen. Wir haben sie nicht abgeglichen und wir wollen hier darauf verzichten, Ihnen genaue Abgleich-Anweisungen anzugeben. Es sei nur darauf hingewiesen, daß ein Grob-Abgleich mit dem in der Liste im Bild H9.1 rot eingetragenen Ziffernpaar bei der Adresse 18ØC vorgenommen werden kann. Der Fein-Abgleich kann mit dem ebenfalls rot eingetragenen Ziffernpaar bei der Adresse 18Ø4 erreicht werden.

In unserem Uhren-Programm haben wir teilweise auf die komfortablen Möglichkeiten des in unserem System verwendeten Mikroprozessors Z8Ø zurückgegriffen. Das Programm ist also auf 8Ø8Ø- und 8Ø85-Systemen nicht ohne weiteres lauffähig. Hier, am Anfang unseres Lehrgangs, kam es uns auch nur darauf an, Ihnen die Möglichkeiten Ihres Systems an zwei Beispielen vorzuführen.

Der Aufbau des Mikroprozessors

Beim Durcharbeiten des ersten Lehrbriefs haben Sie sich mit der Handhabung Ihres Mikroprozessor-Systems vertraut gemacht und versuchsweise auch ein paar kleine Programme laufen lassen. Sie haben gemerkt, daß der Mikroprozessor etwas kann, aber richtig überzeugt von seinem Können sind Sie sicher noch nicht.

Der Sinn unseres Lehrgangs soll es nun nicht sein, Ihnen unverstandene Programme vorzusetzen, die Sie staunend abspielen können. So richtig reizvoll wird die Sache doch erst, wenn Sie selbst Programme nach eigenem Geschmack entwerfen können. – Alles, was wir bisher erarbeitet haben, ist nur die Vorbereitung zum Erreichen dieses Ziels.

Programmieren können Sie erst dann, wenn Sie den Aufbau des Mikroprozessors durchschaut haben. Wir wollen jetzt anfangen, uns diesen Aufbau anzusehen.

Ehe wir aber diesen Aufbau – **Architektur** nennt der Fachmann das – erläutern, wollen wir gleich die Grenzen unserer Bemühungen festlegen. Als Hardware-Elektroniker sind Sie es gewohnt, auch bei integrierten Schaltungen bis in die Halbleiter-Struktur der von Ihnen benutzten IC's vorzudringen.

Wenn Sie sich mit der Handhabung des Mikroprozessors vertraut machen wollen, dann sind Sie in der gleichen Situation wie jemand, der ein Auto kaufen und nun zunächst das Fahren lernen will. Natürlich müssen Sie außer den Verkehrsregeln auch wissen, wie man bremst, wie man rückwärts in eine Parklücke rangiert und was es mit der Kupplung und mit den Gängen auf sich hat. Aber sogar die Gangschaltung können Sie vergessen, wenn Ihr Auto ein automatisches Getriebe hat. Jedenfalls wird Sie bei der Führerscheinprüfung kaum jemand nach der Funktion des Vergasers oder nach Einzelheiten des Unterschiedes zwischen Zweitakt- und Viertaktmotor fragen. Wenn Sie Ihr Auto sicher handhaben und die Verkehrsregeln beherrschen, dann werden Sie auch ohne Detailkenntnisse ein guter Autofahrer.

Das Ziel unseres Lehrgangs soll der Erwerb des Führerscheins zum Betrieb des Mikroprozessors sein. Sie werden noch sehen, daß dazu die Kenntnis seiner Halbleiterstruktur keineswegs notwendig ist. Denn ganz sicher wollen Sie genau so wenig eine neue CPU (Seite H 2) entwerfen, wie der Fahrschüler ein Auto bauen will. Und der Umgang mit dem Mikroprozessor ist genauso schön wie das Autofahren. Was noch schöner ist, gehört nicht zum Thema dieses Lehrgangs.

Akkumulator, B- und C-Register

Ein Register ist ein Verzeichnis oder eine Ablage für wichtige Dinge, die man sich merken möchte. Genau diese Funktion erfüllen auch die Register, die in Mikroprozessoren meist in mehr oder weniger großer Anzahl anzutreffen sind. Der in unserem System verwendete Mikroprozessor enthält insgesamt zweiundzwanzig Register. Auch ein

Akkumulator ist nichts anderes als ein solches Register. Da er jedoch in allen Mikroprozessoren ein zentrales Register darstellt, hat er seinen besonderen Namen bekommen.

Was sind für einen Mikroprozessor wichtige Dinge? Es sind Dinge, von und mit denen ein Mikroprozessor lebt, also Zahlen. In seinen Registern werden Zahlen abgelegt, die in einer der sedezimalen Schreibweise verwandten Art dargestellt sind. Wir tun zunächst so, als würden in den Registern tatsächlich sedezimal dargestellte Zahlen verwaltet. In Wirklichkeit werden diese Zahlen in den Registern binär dargestellt, mit einem Zeichenvorrat, der nur zwei unterschiedliche Zeichen enthält. Die sedezimale Darstellung von Zahlen ist nichts anderes, als eine für uns recht bequeme Art, mit nur zwei unterschiedlichen Zeichen dargestellte Zahlen zu handhaben. Doch davon später.

Zunächst wollen wir den Mikroprozessor als ein Bauelement betrachten, in dessen Registern sedezimal zweistellig dargestellte Zahlen verwaltet werden. Verwalten bedeutet dabei das, was auch ein Buchhalter mit seinen Zahlen tut: Er schreibt sie in die gehörigen Register, verwahrt sie dort, holt sie bei Bedarf hervor und legt sie – aus welchen Gründen auch immer – in einem anderen Register ab. Manchmal addiert er auch zwei Zahlen aus verschiedenen Registern, streicht vielleicht ein paar Stellen weg, legt das Ergebnis wieder in einem Register ab usw.

Ganz genau das gleiche geschieht auch im Mikroprozessor. Eines ist allerdings bei diesem Bauelement bemerkenswert: Das, was der Mikroprozessor mit den in seinen Registern abgelegten Zahlen tun soll, wird ihm durch eine Anweisung, also durch Befehle mitgeteilt, die ihrerseits wieder als sedezimal dargestellte Zahlen codiert sind.

Wie die im Mikroprozessor enthaltenen Register tatsächlich aussehen, soll uns an dieser Stelle noch nicht interessieren. Wir stellen sie zunächst einfach wie in einem Blockschaltplan als Kästen dar. Wir wollen auch nicht fragen, durch wieviele elektrische Leitungen die Register untereinander verbunden sind. Wir stellen nur fest, daß zwischen diesen Registern sedezimal dargestellte Zahlen transportiert oder ausgetauscht werden können. Wichtig ist allerdings, daß jedes der hier betrachteten Register solche Zahlen faßt, die sedezimal mit zwei Ziffern dargestellt werden können.

In einem der nächsten Abschnitte erläutern wir, daß diese zwei Ziffern binär durch acht Zeichen dargestellt werden. Daher kommt die Bezeichnung 8-Bit-Mikroprozessor.

Aufgabe H14.1

a) Wieviel unterschiedliche Zahlen können in einem Register eines 8-Bit-Mikroprozessors abgelegt werden?

b) Geben Sie in sedezimaler Darstellung die größte Zahl an, die in einem der hier behandelten Register abgelegt werden kann.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 5.

Am Ende dieses Lehrbriefs haben wir auf der Seite T 2 die Architektur des Z 80-Mikroprozessors dargestellt, der in Ihrem System verwendet

wird. Das einzige, was Sie im Augenblick erkennen können, ist die Tatsache, daß unser Mikroprozessor eine scheinbar verwirrende Menge unterschiedlicher Register enthält und daß ein Teil dieser Register auch noch „breiter" ist als die anderen. All' das soll Sie aber jetzt nicht stören. Wir wollen Sie aber bereits jetzt darauf hinweisen, daß die in der Architektur grau unterlegten Register auch im 8085-Mikroprozessor vorhanden sind. Mit diesen Registern wollen wir uns in unserem Lehrgang vorzugsweise beschäftigen. Die nicht grau unterlegten Register sind Z 80-spezifisch. Sie erkennen, daß der Z 80 offenbar gegenüber dem 8085 einen zweiten Registersatz enthält, der genau dem Haupt-Registersatz entspricht.

Wir betrachten zunächst nur den **Akkumulator** und das **B-Register**. Im Bild H15.1 sind diese beiden Register als Blockschaltzeichen angegeben und mit ihren Kennbuchstaben (A, B) gekennzeichnet. Der eingetragene, rote Verbindungspfeil soll nicht eine elektrische Verbindung darstellen (die zweifellos im Inneren der CPU in irgendeiner Weise existiert); der Pfeil soll lediglich aussagen, daß die im Akkumulator abgelegte Zahl ins B-Register übertragen werden kann. Diese Übertragung wird von einem Befehl ausgelöst, der mit den Ziffern 47 codiert wird.

Bild H15.1
Der Inhalt (A) des Akkumulators kann mit dem Befehl 47 (LD B, A) in das B-Register übertragen werden.

Sie werden im Laufe des Lehrgangs noch eine ganze Reihe anderer solcher Befehle zur Übertragung des Inhalts eines Registers in ein anderes Register kennenlernen. Alle diese Befehle gehören zur Gruppe der **Lade-Befehle**.

Den Vorgang, der durch den mit den Ziffern 47 codierten Befehl ausgelöst wird, drückt man in der Fachsprache so aus:

Mit dem Befehl 47 wird das B-Register mit dem Inhalt des Akkumulators geladen.

Versuch H15.1

Anzeige und Verändern von Register-Inhalten.

Schließen Sie Ihr System an und warten Sie die System-Meldung ab; betätigen Sie ggf. die Taste RS, um die System-Meldung in die Anzeige zu bekommen.

Betätigen Sie die Taste REG, die Sie im Tastenfeld in der zweiten Reihe als (von links gezählt) fünfte Taste finden. Das Bild H16.1a stellt die Anzeige dar, die die Bereitschaft des Systems meldet, anschließend die Inhalte beliebiger Register anzuzeigen. – Wir interessieren uns zunächst für den Inhalt des Akkumulators.

Betätigen Sie die mit AF bezeichnete, weiße Taste an der (von rechts gezählt) vierten Stelle der unteren Tastenreihe! Diese Taste hat – wie alle sechzehn weißen Tasten im rechten Teil des Tastenfeldes – eine Doppelfunktion. Die über der mit 0 bezeichneten Taste angegebene Funktion AF wird automatisch dann wirksam, wenn vorher die Taste REG betätigt worden ist.

Das Bild H16.1b zeigt, welche Anzeige Sie erhalten: An den rechten beiden Anzeigestellen erscheinen die beiden Buchstaben A und F; welche Werte an den linken vier Stellen erscheinen, können wir nicht voraussagen. Wir haben deshalb in unser Bild die (nicht wirklich erscheinenden) Werte X eingetragen.

Der Buchstabe A an der (von rechts gezählt) zweiten Anzeigestelle meldet, daß an den (von links gezählt) ersten und zweiten Anzeigestellen der derzeitige Inhalt des Akkumulators angegeben wird. Der Buchstabe F an der ganz rechten Anzeigestelle sagt, daß an den (von links gezählt) dritten und vierten Anzeigestellen der derzeitige Inhalt des Flag-Registers gezeigt wird. Sie finden dieses Flag-Register auf der Seite T2 oberhalb vom Akkumulator. Was es mit dem Flag-Register auf sich hat, werden wir Ihnen später erläutern. Interessant ist im Augenblick nur, daß in der Anzeige die Inhalte von zwei Registern gleichzeitig erscheinen. Sie erinnern sich: In den hier betrachteten Registern können jeweils sedezimal zweistellig darstellbare Zahlen abgelegt werden (Seiten H10 und H14). Die vier linken Anzeigestellen bieten sich also zur gleichzeitigen Darstellung der Inhalte von zwei Registern an.

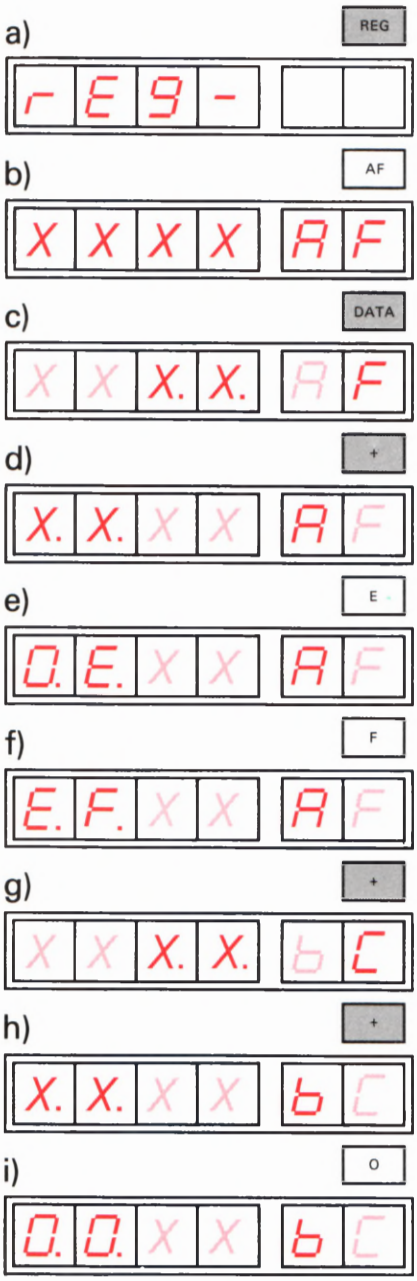


Bild H16.1
Tasten und Anzeigen beim Laden von A und Löschen von B.

Betätigen Sie jetzt die Ihnen bereits bekannte Taste DATA und beobachten Sie die Anzeige! Die angezeigten Zeichen ändern sich nicht, abgesehen davon, daß die Anzeige ganz kurzzeitig verlöscht. Zusätzlich erscheinen jedoch an den beiden mittleren Anzeigestellen Dezimalpunkte (Teilbild c). Diese Dezimalpunkte sind Ihnen bereits von der Eingabe von Programm-Daten her bekannt (Seite H9). Sie melden, daß der Inhalt des Flag-Registers geändert werden kann.

Betätigen Sie versuchsweise beliebige der weißen Tasten in der rechten Hälfte des Tastenfeldes! Sie erkennen, daß die eingegebenen Ziffern von rechts nach links an den für den Inhalt des Flag-Registers zuständigen beiden Stellen durch die Anzeige geschoben werden. Die beiden für den Akkumulator-Inhalt zuständigen Anzeigestellen bleiben bei diesen Eingaben unbeeinflußt.

Weil wir uns für das Flag-Register und seinen Inhalt hier noch nicht interessieren, muß dafür gesorgt werden, daß Zifferneingaben für den Inhalt des Akkumulators wirksam werden. – Betätigen Sie die (Ihnen ebenfalls bereits bekannte) Taste + ! Bei dieser Eingabe bleiben die Ziffern in der Eingabe unverändert; lediglich die beiden Dezimalpunkte erscheinen an den beiden linken Anzeigestellen. Jetzt kann der Inhalt des Akkumulators geändert werden (Teilbild d. Wir haben in unserer Darstellung die jeweils nicht interessierenden Ziffern schwächer dargestellt. In der Anzeige sind natürlich alle Ziffern immer gleich hell.)

Wie vorher beim Flag-Register können Sie jetzt durch Betätigen der weißen Ziffern-Taste beliebige, sedezimal zweistellig dargestellte Zahlen in den Akkumulator eintasten. Beachten Sie bitte, daß nach dem Eintasten der ersten Ziffer nach der Taste + in der ganz linken Anzeigestelle die Ziffer 0 erscheint! – In den Teilbildern e und f haben wir die Anzeige nach der aufeinanderfolgenden Eingabe der Ziffern E und F dargestellt. Bei diesem Inhalt EFH des Akkumulators wollen wir es zunächst bewenden lassen.

Jetzt soll noch der Inhalt des B-Registers zu 00 gemacht werden. In der Fachsprache sagt man: Das B-Register soll gelöscht werden.

Rechts neben der mit AF bezeichneten weißen Taste finden Sie eine Taste, für die als zweite Funktion BC eingetragen ist. Diese Taste ist für die Inhalte der B- und C-Register (vgl. Seite T2) zuständig. Zum Löschen des B-Register-Inhalts könnten Sie nun entsprechend dem

soeben beschriebenen Verfahren vorgehen und der Reihe nach die Tasten REG, BC, DATA und + betätigen. Tun Sie das aber nicht, denn es geht viel einfacher.

Wir nehmen an, daß Sie zuletzt die zwei weißen Ziffern-Tasten E und F zur Eingabe in den Akkumulator betätigt haben. Die Dezimalpunkte stehen also noch in den beiden linken Anzeigestellen.

Betätigen Sie die Taste +! Das Bild H16.1g zeigt die jetzt entstandene Anzeige: In den rechten beiden Anzeigestellen sind die Buchstaben B und C erschienen. (Es soll Sie nicht stören, daß die Anzeige das B als Kleinbuchstaben bringt. Das ist notwendig, damit Verwechslungen mit der Ziffer 8 vermieden werden.) Die linken vier Anzeigen melden Ihnen jetzt die Inhalte der B- und C-Register. Die Dezimalpunkte stehen in den mittleren beiden Anzeigen; die beiden dort stehenden Ziffern bilden den Inhalt des C-Registers ab, der durch Eingaben mit den weißen Tasten geändert werden kann. Machen Sie einen entsprechenden Versuch!

Wie vorher der Inhalt des Flag-Registers, so interessiert uns hier der Inhalt des C-Registers nicht. Betätigen Sie noch einmal die Taste +! Die Dezimalpunkte stehen jetzt in den beiden linken Anzeigestellen (Teilbild h), die für den Inhalt des B-Registers zuständig sind. – Betätigen Sie die Taste 0! Das Teilbild i zeigt, was passiert: In der (von links gezählt) zweiten Anzeigestelle erscheint die eingetastete Ziffer 0; gleichzeitig wird automatisch an die ganz linke Anzeigestelle ebenfalls eine Ziffer 0 eingetragen. Sie können sich also für Ihr System merken:

> Bei der Eingabe von Daten nach vorher betätigter DATA-Taste genügt das einmalige Betätigen der Taste 0, um beide Daten-Ziffern auf die Werte 0 zu bringen.

Mit den jetzt erfolgten Eingaben haben Sie den Wert EF**H** in den Akkumulator und den Wert ØØ**H** in das B-Register eingetragen. Es ist üblich, den Inhalt eines Registers durch die in Klammern gesetzte Bezeichnung dieses Registers zu kennzeichnen. Die jetzt eingestellten Inhalte von Akkumulator und B-Register können demnach so dargestellt werden:

(A) = EF**H**; (B) = ØØ**H**

Aufgabe H17.1

Überlegen Sie bitte, welche Tasten Sie betätigen müssen, wenn Sie sich davon überzeugen wollen, daß der eben in den Akkumulator eingegebene Wert immer noch im Akkumulator steht!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü5.

Wir wollen uns jetzt die Aufgabe stellen, mit dem vorhin beschriebenen Befehl 47 (Bild H15.1) das B-Register mit dem Inhalt des Akkumulators zu laden. Für den folgenden Versuch ist Ihnen bekannt, daß

im Akkumulator EF**H** steht und daß nach der richtigen Ausführung des Befehls 47 im B-Register ebenfalls EF**H** stehen muß. Grundsätzlich muß aber nach der richtigen Ausführung des Befehls 47 – unabhängig vom Akkumulator-Inhalt – im B-Register der gleiche Wert wie im Akkumulator stehen.

Versuch H18.1

B-Register mit dem Inhalt des Akkumulators laden

Der Befehl 47 wird bei der Adresse 18ØØ abgelegt. Es ist Ihnen jetzt bereits geläufig, daß Sie dazu diese Adresse zuerst anwählen und anschließend – nach der Umschaltung des Systems auf Daten-Eingabe – das Ziffernpaar 47 eintasten müssen:

ADDR, 1, 8, 0, 0	Anwählen der Adresse
DATA, 4, 7	Eingabe der Daten

Mit diesen Eingaben haben Sie den Mikroprozessor programmiert, obwohl dieses Programm nur aus einer einzigen Anweisung besteht. Wenn Sie das Programm jetzt mit der Taste GO starten (halt, tun Sie's nicht!), wie Sie es bei den Versuchen im Lehrbrief 1 getan haben, dann läuft das Programm los und läuft und läuft... „in die Wüste". Denn Sie wissen ja nicht, welche Befehle unser Mikroprozessor-System außer dem gewollten Lade-Befehl sonst noch enthält, die sich beim Einschalten zufällig eingestellt haben! Vielleicht bleibt das „Programm" irgendwann und irgendwo stehen; vielleicht ändert es aber auch ganz unsinnig die soeben eingegebenen Werte.

Wir wollen erreichen, daß der Mikroprozessor nach einem eingegebenen Kommando nur einen einzigen Befehl, nämlich den absichtlich eingegebenen Lade-Befehl mit dem Code 47, ausführt und dann darauf wartet, ob mit dem gleichen Kommando die Ausführung des nächstfolgenden Befehls veranlaßt wird. Diese Art der Abarbeitung eines Programms wird als **Einzelschritt**-Programmablauf bezeichnet. Die englische Bezeichnung dafür ist *single step* (sprich: ssingel step, mit „spitzem" st). Zuständig für diese Art des Programmablaufs ist die graue Steuertaste STEP, die Sie an der (von links gezählt) dritten Stelle der dritten Reihe des Tastenfeldes finden.

Überzeugen Sie sich, daß im Adreßfeld der Anzeige (linke vier Anzeigestellen) die Adresse 18ØØ mit dem dort eingetragenen Ziffernpaar 47 steht. Wenn das aus irgendwelchen Gründen nicht der Fall ist, dann wählen Sie bitte mit den Tasten ADDR, 1, 8, 0, 0 diese Adresse neu an und betätigen Sie dann einmal die Taste STEP!

In der Anzeige erscheint jetzt

1801 XX

XX haben wir hier für Daten mit nicht voraussagbarem Wert angeschrieben. Wesentlich ist, daß im Adreßfeld der Anzeige die Adresse 18Ø1 steht. Das System hat den Befehl 47 bei der Adresse 18ØØ ausgeführt und wartet jetzt, ob es vielleicht das Kommando erhält, den Befehl XX bei der Adresse 18Ø1 auszuführen.

Bevor wir uns nun ansehen, was die Ausführung des Befehls 47 bewirkt hat, wollen wir eine wichtige Feststellung treffen:

Nach der Ausführung eines Befehls wird CPU-intern automatisch die Adresse des nächstfolgenden Befehls angewählt.

Die Adressen sind in einem Doppelregister abgelegt, das als **Programmzähler** bezeichnet wird. Die englische Bezeichnung dieses Registers ist ***Program-Counter*** mit der Abkürzung **PC**.

Sie finden das Register „Programmzähler" in der Darstellung auf der Seite T2 ganz oben. Es ist ein Doppelregister, weil Adressen immer sedezimal vierstellig dargestellt werden (vgl. Seite H10).

Jetzt interessiert uns natürlich, ob der soeben ausgeführte Befehl 47 erwartungsgemäß seine Pflicht getan hat. Hat er das, dann muß jetzt im vorher gelöschten B-Register der gleiche Wert EF stehen wie im Akkumulator, denn wir haben dem Mikroprozessor ja die Anweisung gegeben, das B-Register mit dem Inhalt des Akkumulators zu laden.

Sehen Sie sich zunächst noch einmal den Inhalt des Akkumulators an:

```
REG, AF
```

Dort steht nach wie vor der Wert EF**H**. – Jetzt sehen wir nach, was im B-Register steht:

```
REG, BC
```

An den linken beiden Anzeigestellen, die jetzt für den Inhalt des B-Registers zuständig sind, und die vor der Ausführung des Befehls 47 den Wert (B)=ØØ**H** aufwiesen, steht jetzt ebenfalls der Wert EF**H**.

Bitte merken Sie sich:

Lade-Befehle sind Kopier-Befehle. Sie bewirken die Kopie des Inhalts eines Registers in ein anderes Register.

Wir werden den Begriff der Lade-Befehle später noch etwas erweitern.

Der Befehlssatz unseres Mikroprozessors verfügt über eine große Anzahl unterschiedlicher Befehle, deren jeder durch eine sedezimal zweistellig dargestellte Zahl codiert ist. Etwas anderes als Zahlen kann der Mikroprozessor ja nicht verarbeiten. – Sie werden beim Umgang mit dem Mikroprozessor mit Sicherheit eine Reihe dieser Zahlen ganz automatisch auswendig lernen. Grundsätzlich wäre es jedoch eine Zumutung, wenn Sie sämtliche Zahlen für alle Befehle auswendig lernen müßten.

Nur zur Vereinfachung beim Schreiben eines Programms ist man übereingekommen, die einzelnen Befehle in leicht merkbarer, abgekürzter Schreibweise anzugeben. Wohlgemerkt: Mit diesen leicht merkbaren Abkürzungen kann der Mikroprozessor nicht das geringste anfangen. Er versteht nichts anderes als Zahlen. Aber für Sie selbst sind diese Zahlen viel schwieriger zu behalten, als aussagekräftige, leicht

merkbare Abkürzungen. Diese Abkürzungen sind sogar dann besser zu handhaben, wenn sie aus der englischen Sprache kommen.

Die Darstellung der Befehle in der leicht merkbaren Schreibweise bezeichnet man als **mnemonischen Code**, während man die Darstellung in der dem Mikroprozessor allein verständlichen Schreibweise als **Operationscode** bezeichnet.

Beim Aufstellen eines Programms bedient man sich zweckmäßig der leicht merkbaren mnemonischen Codes. Ehe man den Mikroprozessor mit dem Programm füttert, nimmt man dann anhand einer Liste die Umcodierung der mnemonischen Codes in die Operationscodes vor.

In das Bild H15.1 haben wir außer dem Operationscode 47 für den Befehl zum Kopieren des Akkumulator-Inhalts in das B-Register auch den zugehörigen mnemonischen Code eingetragen:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD B, A	47	B ← (A)	B-Register mit Inhalt von A laden

Die mnemonische Abkürzung LD kommt vom englischen L*oa*D, was laden bedeutet: Das eine Register wird mit dem Inhalt des anderen Registers geladen. Die anschließende Angabe B, A sagt aus, wohin die Zahl geladen werden soll und woher sie kommt.

Bitte merken Sie sich:

> In den mnemonischen Codes der Lade-Befehle wird stets zuerst das Ziel-Register angegeben und, abgetrennt durch ein Komma, danach das Herkunfts-Register.

Ein ganz ähnliches Register wie das B-Register ist das C-Register. Das Bild H20.1 zeigt die Architektur der drei Register A, B und C. Es wird deutlich, daß der Akkumulator A als zentrales Register eine Sonderstellung einnimmt, während das B- und das C-Register gleichberechtigt nebeneinander stehen. Sie werden später sehen, daß diese beiden Register auch untereinander eine besondere Beziehung haben. Die zwischen den Registern hin und her weisenden Pfeile deuten an, wie Inhalte zwischen den Registern übertragen werden können. Sie werden die entsprechenden Befehle in einem der nächsten Abschnitte kennenlernen. Hier sollen Sie sich nur noch die Befehle LD C, A und LD B, A ansehen.

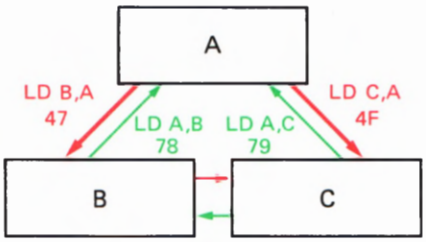


Bild H20.1
Architektur des Akkumulators und der Register B und C. Die Pfeile zeigen die Übertragungsmöglichkeiten der Register-Inhalte. In diesem Abschnitt werden nur die Befehle LD B, A und LD C, A behandelt.

Aufgabe H20.1

Vor dem Start des nächsten Programms soll der Akkumulator mit dem Wert 1A**H** geladen werden. Außerdem sollen die Inhalte der Register B und C gelöscht werden.

Überlegen Sie bitte, welche Tasten Sie betätigen müssen!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü5.

Aus der Beschreibung des mnemonischen Codes LD für die Lade-Befehle ergibt sich, daß mit dem Befehl LD C, A das C-Register mit dem Inhalt des Akkumulators geladen wird:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD C, A	4F	C ← (A)	C-Register mit Inhalt von A laden

Mit Hilfe dieses und des bereits erläuterten Befehls LD B, A wollen wir eine Folge von zwei Befehlen anschreiben, die nacheinander das Laden des B-Registers und des C-Registers mit dem Inhalt des Akkumulators bewirken. Dafür bietet sich folgendes Schema an:

Nr.	Befehl	Adresse	Opcode	Bemerkungen
1	LD B, A	19AB	47	B ← (A)
2	LD C, A	19AC	4F	C ← (A)

In diesem Schema fällt zweierlei auf: Wir haben statt des Worts Operationscode die häufig benutzte Abkürzung **Opcode** verwendet. Außerdem haben wir den ersten Befehl statt wie bisher bei der Adresse 18Ø0 diesmal bei der Adresse 19AB als Anfangsadresse und den nächsten Befehl entsprechend bei der Adresse 19AC eingetragen. Einen besonderen Grund gibt es dafür nicht; wir wollen einfach mal probieren, ob das wohl auch funktioniert.

Versuch H21.1

B- und C-Register mit dem Inhalt des Akkumulators laden

Laden Sie zuerst den Akkumulator mit dem Wert 1A**H** und löschen Sie die Inhalte der B- und C-Register. Bei der Lösung der Aufgabe H20.1 haben Sie gesehen, wie das gemacht wird.

Tragen Sie anschließend die Operationscodes 47 und 4F für die Befehle LD B, A und LD C, A bei den Adressen 19AB und 19AC ein. Für diesen Vorgang brauchen wir Ihnen jetzt nicht mehr die einzelnen Tastenbetätigungen anzugeben. Im Zweifelsfall können Sie z. B. beim Versuch S25.1 nachlesen, wie die Eingabe vorgenommen wird.

Für den Ablauf der Befehlsfolge gelten die gleichen Überlegungen, die wir bereits beim Versuch H18.1 angestellt haben: Die Abarbeitung der Befehle darf nicht mit der Taste GO gestartet werden, sondern muß in Einzelschritten erfolgen. Dieses Einzelschritt-Verfahren hat zudem den großen Vorteil (und dazu ist es eigens vorgesehen!), daß man den Ablauf des Programms nach jedem einzelnen Befehl genau kontrollieren kann.

Wählen Sie die Start-Adresse 19AB des Programms an und lassen Sie dann den Befehl LD B, A in einem Einzelschritt ausführen:

ADDR, 1, 9, A, B	Anwahl der Start-Adresse
STEP	Ausführung des ersten Befehls

Die Anzeige 19AC 4F meldet jetzt, daß der Mikroprozessor auf das Kommando wartet, den bei der Adresse 19AC abgelegten Befehl LD C, A mit dem Operationscode 4F auszuführen. Wir tun ihm diesen Gefallen aber zunächst noch nicht, sondern wollen uns erst überzeu-

gen, ob er den vorhergehenden Befehl LD B, A richtig ausgeführt hat. Dazu sehen wir uns den Inhalt des vorher gelöschten B-Registers an:

REG, BC

Richtig! Die linken beiden Anzeigestellen sind jetzt für den Inhalt des B-Registers zuständig und aus der Anzeige 1A00 BC erkennen wir, daß im B-Register jetzt der vorher in den Akkumulator eingetragene Wert 1AH steht. – Die Ziffern 0 an den beiden mittleren Anzeigestellen melden, daß sich am Inhalt des C-Registers noch nichts geändert hat.

Jetzt soll der zweite Befehl (LD C, A) unseres Programms bei der Adresse 19AC als Einzelschritt ausgeführt werden. Es liegt fast nahe, diese Adresse mit der Taste ADDR anzuwählen und dann die Taste STEP zu betätigen. Unser System macht es uns aber einfacher. Sie erinnern sich, daß jeweils die Adresse des nächsten, auszuführenden Befehls im Programmspeicher der CPU abgelegt wird. Es muß uns also nur gelingen, diesen Wert aus dem PC-Register (Seite H19) wieder zu aktivieren und in die Anzeige zu holen. Das geht ganz ähnlich wie das Aufrufen anderer Register mit Hilfe der Taste REG. Allerdings hat der Programmzähler eine so wichtige Funktion, daß ihm im Tastenfeld unseres Systems eine eigene Taste zugeordnet wurde. Sie finden diese Taste an der (von links gezählt) fünften Stelle in der ersten Reihe mit der Bezeichnung PC.

Betätigen Sie diese Taste PC! In der Anzeige erscheint links die Adresse 19AC des nun auszuführenden Befehls mit dem Operationscode 4F. Jetzt brauchen Sie nur noch die Taste STEP für den nächsten Programm-Einzelschritt zu betätigen. Tun Sie's! – Was geschieht? Zunächst nichts anderes, als daß die Adresse 19AD mit dem Inhalt XX angezeigt wird, der rein zufällig ist und den wir nicht voraussagen können, denn bei dieser Adresse haben wir ja vorher nichts eingegeben. Die CPU kann das natürlich nicht wissen. Sie nimmt an, daß dieser Wert XX der Operationscode eines Befehls ist und harrt gehorsam der Dinge, die in Form unseres nächsten Kommandos kommen.

Unser nächstes Kommando besteht jetzt nur noch darin, daß wir uns ansehen, ob auch der Befehl LD C, A richtig ausgeführt worden ist. Das ist einfach: Betätigen Sie nacheinander die Tasten REG und BC und sehen Sie sich die Anzeige an: 1A1A BC. Das C-Register ist jetzt wie vorher das B-Register mit dem Wert 1AH aus dem Akkumulator geladen worden.

Aufgabe H22.1

Lesen Sie bitte noch einmal auf der Seite H20 über den Aufbau der mnemonischen Codes der Lade-Befehle nach und betrachten Sie das Bild H20.1!

Überlegen Sie jetzt, ob es vielleicht möglich ist, einen anderen Befehl als LD C, A einzusetzen, um nach dem B-Register auch das C-Register mit dem Inhalt des Akkumulators zu laden.

Wir haben einen solchen Befehl zwar noch nicht besprochen, aber vielleicht gibt es ihn? Lassen Sie Ihre Phantasie spielen!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü5.

Der Aufbau des Mikroprozessors (Fortsetzung)

Auf der Seite T2 haben wir in vereinfachter Form die Architektur (Seite H13) des in Ihrem System benutzten Mikroprozessors Z 80 aufgezeichnet. Dabei haben wir nur die Register dargestellt, die beim Erstellen einfacher Programme wichtig sind.

In den folgenden Abschnitten werden wir uns zunächst mit den Ihnen bereits bekannten Registern A, B und C und zusätzlich mit den Registern D, E, H und L des Haupt-Registersatzes beschäftigen. Später werden wir dann erläutern, was es mit dem Programmzähler, dem Stackpointer und dem Flag-Register auf sich hat. Sie werden sehen, daß diese Register nicht nur einfach der Ablage sedezimal dargestellter Zahlen dienen. Jedes dieser Register hat vielmehr im Zusammenhang mit den anderen Registern ganz spezielle und äußerst nützliche Eigenschaften.

Der Akkumulator und die Register B, C, D, E, H und L

Wir haben bereits auf der Seite H13 darauf aufmerksam gemacht, daß die Kenntnis der Architektur eines Mikroprozessors Voraussetzung für das Programmieren ist. Neben der Beherrschung des Befehlssatzes muß man beim Programmieren die auf der Seite T2 für unseren Mikroprozessor dargestellte Architektur vor seinem geistigen Auge haben. Machen Sie sich jedoch bitte nicht die Mühe, dieses Bild auswendig zu lernen. Die dauernde Beschäftigung wird Ihnen die Register-Anordnung ganz von selbst geläufig machen.

Damit Sie nicht immer umzublättern brauchen, haben wir die Anordnung der hier interessierenden Register im Bild H23.1 noch einmal dargestellt. Der obere Teil des Bildes entspricht der Darstellung im Bild H20.1 mit dem Akkumulator A als zentrales Register und den Registern B und C als zusätzliche Ablage-Register. Neu sind in der Darstellung die Register D, E, H und L.

Grundsätzlich sind zunächst – abgesehen vom Akkumulator – alle im Bild dargestellten Register untereinander gleichartig: Jedes dieser Register kann ganz einfach zur Ablage einer sedezimal zweistellig dargestellten Zahlen dienen. Die Art der Anordnung in unserem Bild läßt jedoch bereits jetzt besondere Zusammenhänge zwischen den einzelnen Registern vermuten. Offenbar stehen jeweils die Register B und C, D und E sowie H und L zueinander in einem besonderen Zusammenhang. Außerdem zeigt die aus der alphabetischen Reihenfolge fallende Bezeichnung der Register H und L, daß diese Register eine Sonderstellung einnehmen.

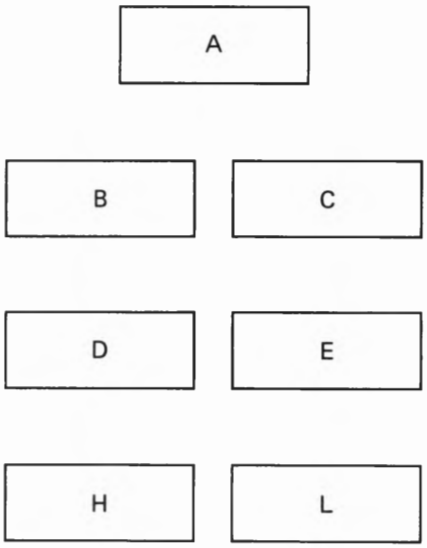


Bild H23.1
Anordnung der Register A, B, C, D, E, H und L im Haupt-Registersatz des Z80-Mikroprozessors.

Diese Besonderheiten sollen hier nur angedeutet werden. Sie werden die daraus folgenden, komfortablen Eigenschaften unseres Mikroprozessors noch kennenlernen. Im Augenblick interessieren uns die Register nur einzeln und in ihrer Eigenschaft als Ablage-Speicher. In dieser Eigenschaft werden sie auch häufig in Programmen benutzt.

Bei der Beschäftigung mit den Registern A, B und C haben Sie Befehle kennengelernt, mit denen sedezimal dargestellte Zahlen aus dem

Akkumulator A wahlweise in das B- und in das C-Register geladen (kopiert) werden können. Selbst herausgefunden haben Sie, daß es einen Befehl geben muß, mit dem eine Zahl auch direkt vom B-Register in das C-Register übertragen werden kann (Aufgabe H22.1 und deren Lösung auf der Seite Ü5). Vor den jeweiligen Versuchen haben Sie den vom Akkumulator in eins der anderen Register zu übertragenden Werte zunächst von Hand mit Hilfe der Tasten REG und DATA in den Akkumulator geladen. Selbstverständlich enthält der Befehlssatz des Mikroprozessors auch einen Befehl, der das Laden des Akkumulators mit einer beliebigen, sedezimal zweistellig dargestellten Zahl ermöglicht.

Diesen Befehl wollen wir Ihnen zwischendurch vorstellen und bei den anschließenden Versuchen benutzen:

Mnemonischer Code	Operationscode	Operation	Erläuterung
LD A,Konst	3E	A ← Konst	A mit konstantem Wert laden

Der mnemonische Code LD (von **L***oa***D** = laden) entspricht dem, den Sie beim Lade-Befehl LD B,A auf der Seite H20 kennengelernt haben. Im Gegensatz zu einem solchen Lade-Befehl weist der Befehl LD A,Konst jedoch eine Besonderheit auf. Der Operationscode 3E dieses Befehls sagt dem Mikroprozessor, daß der Akkumulator mit einer bestimmten, sedezimal zweistellig dargestellten Zahl Konst geladen werden soll. Er sagt dem Mikroprozessor aber nicht, welche Zahl in den Akkumulator gebracht werden soll.

Die bereits vorgestellten Lade-Befehle bestanden nur aus einem Operationscode, also aus einer einzigen, sedezimal zweistellig dargestellten Zahl. Eine solche Zahl bezeichnet man in der Fachsprache als **Byte** (sprich: bait). Was es damit auf sich hat, werden Sie in einem der nächsten Abschnitte erfahren. Hier wollen wir nur feststellen, daß es sich bei den Lade-Befehlen zum Kopieren des Inhalts eines Registers in ein anderes Register um **Ein-Byte-Befehle** handelt. Der LD A,Konst-Befehl dagegen ist ein **Zwei-Byte-Befehl**. Das erste Byte enthält den Operationscode; das zweite Byte gibt die Zahl mit dem konstanten Wert Konst an, die in das A-Register geladen werden soll. Dieses zweite Byte wird als **Operand** bezeichnet.

Hier haben Sie den Grund, warum in unseren Programm-Auflistungen (z. B. im Bild H9.1) in vielen Zeilen mehr als eine sedezimal zweistellig dargestellte Zahl steht. Die Auflistungen lassen erkennen, daß der Befehlssatz unseres Mikroprozessors auch Drei-Byte-Befehle (und speziell der Mikroprozessor Z80 sogar Vier-Byte-Befehle) enthält. Die Drei-Byte-Befehle bestehen aus einem Operationscode, einem 1. Operanden und einem 2. Operanden. (Die Vier-Byte-Befehle des Z80 enthalten einen Zwei-Byte-Operationscode und zwei zusätzliche Operanden.) Die Zusammensetzung der Drei-Byte-Befehle stellen wir Ihnen später vor. Von den Vier-Byte-Befehlen des Z80 werden wir ab und zu Gebrauch machen. Diese Befehle sind im sonst gleichen Befehlssatz des 8085-Mikroprozessors nicht enthalten.

Versuch H25.1

Akkumulator mit konstantem Wert laden

Sorgen Sie dafür, daß in Ihrem System der Inhalt des Akkumulators (A) = ØØ angezeigt wird. (Tasten REG, AF, DATA, +, 0 vgl. Versuch H 15.1.)

Wir stellen uns die Aufgabe, per Programm den Akkumulator mit 67**H** zu laden. Der dafür zuständige Befehl lautet:

Operation	Operand	Adresse	Opcode	1. Operand
LD	A,67	18ØØ	3E	67

Im Gegensatz zur Auflistung der beiden Befehle auf der Seite H21 haben wir hier beim mnemonischen Code des Befehls LD A,67 – 67 ist der konstante Wert Konst – eine Unterteilung in Operation und Operand vorgenommen. Das erscheint in Hinblick auf die Tatsache vernünftig, daß Sie später noch eine Reihe anderer LD-Befehle kennenlernen. – In das Schema haben wir auch gleich die Adresse 18ØØ für den Befehl eingetragen. Diese Eintragung bedeutet, daß der Operationscode 3E bei der Adresse 18ØØ stehen soll und der erste (und in diesem Fall einzige) Operand bei der Adresse 18Ø1.

Laden Sie Ihr System ab der Adresse 18ØØ mit den beiden Bytes 3E und 67! Notfalls können Sie beim Versuch S25.1 noch einmal nachlesen, wie das gemacht wird.

Damit das System nach dem Start des nur aus einem einzigen Befehl bestehenden Programms nicht „in die Wüste" läuft, soll der Lade-Befehl für den Akkumulator in einem Einzelschritt ausgeführt werden:

ADDR, 1, 8, 0, 0	Befehlsadresse anwählen
STEP	Befehl im Einzelschritt ausführen

Sehen Sie sich die Wirkung des soeben ausgeführten Befehls mit Hilfe der Tasten REG und AF an: Im vorher gelöschten Akkumulator steht jetzt das Byte 67**H**.

Wir können uns jetzt wieder den im Bild H23.1 dargestellten Registern zuwenden. Im Bild H26.1 haben wir ein Programm mit sieben Befehlen aufgelistet, das Sie in einem Versuch ausprobieren sollen.

Aufgabe H25.1

Lesen Sie bitte noch einmal auf der Seite H20 nach und überlegen Sie dann, was das im Bild H26.1 aufgelistete Programm bewirkt.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü7.

Versuch H25.2

Die Register B, C, D, E, H und L

Geben Sie bitte in Ihr System das im Bild H26.1 aufgelistete Programm ab der Adresse 18ØØ ein (vgl. Versuch S25.1!)

Vor dem Start des Programms, das wir in Einzelschritten ablaufen lassen wollen, sollen die Register A, B, C, D, E, H und L mit Hilfe der

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand
1		LD	A, AB	18ØØ	3E	AB
2		LD	B, A	18Ø2	47	
3		LD	C, A	18Ø3	4F	
4		LD	D, A	18Ø4	57	
5		LD	E, A	18Ø5	5F	
6		LD	H, A	18Ø6	67	
7		LD	L, A	18Ø7	6F	

Bild H26.1
Programm zum Versuch H25.1. Die in dieser Liste noch nicht benutzte Spalte werden Sie bei späteren Programmen kennenlernen. – Was bewirkt dieses Programm?

Tasten REG, AF, DATA und den anschließend abwechselnd betätigten Tasten 0 und + gelöscht werden. (Lesen Sie noch einmal beim Versuch H15.1 nach!) Bei diesem Vorgang wird auch das noch nicht weiter erwähnte Flag-Register gelöscht, aber das braucht Sie hier nicht zu interessieren. – Sie können den Vorgang nach dem Löschen des H-Registers (Anzeige 0.0.0.0 HL) abbrechen.

Die Tatsache, daß beim Aufrufen oder Ändern des Inhalts der CPU-internen Register jeweils die Inhalte von zwei Registern gleichzeitig angezeigt werden, kommt nicht von Ungefähr. Insbesondere die Register B und C, D und E sowie H und L stehen in einem besonderen Zusammenhang zueinander. Diese Register werden als **Registerpaare** bezeichnet. Die Register können jeweils ganz normal als Einzelregister behandelt werden, in die man sedezimal zweistellig dargestellte Zahlen eintragen kann; es können aber auch zwei Register eines Registerpaares gemeinsam gehandhabt werden und sedezimal vierstellig dargestellte Zahlen aufnehmen. Wir werden Ihnen noch zeigen, was es damit auf sich hat.

Jetzt können Sie das eingegebene Programm in Einzelschritten „abfahren" und die Inhalte der einzelnen Register nach der Ausführung eines jeden Befehls ansehen.

Wählen Sie die Start-Adresse 18ØØ des Programms an und lassen Sie dann den ersten Befehl (LD A, AB) durch Betätigen der Taste STEP ausführen. – Die Wirkung dieses Befehls kennen Sie schon. Mit den Tasten REG und AF können Sie sich unmittelbar nach Ausführung des Befehls davon überzeugen, daß (A) = AB ist. Der gleichzeitig angezeigte Inhalt des Flag-Registers interessiert hier nicht. Durch jeweils zweimaliges Betätigen der Taste + können Sie die Inhalte der übrigen Register (immer als Registerpaare angezeigt) in die Anzeige bringen und feststellen, daß der Befehl LD A, AB tatsächlich nur den Akkumulator mit dem Byte AB geladen und die übrigen Register unbeeinflußt gelassen hat

Nach dem Prüfen des HL-Registerpaares können Sie – wiederum mit der Taste STEP – den nächstfolgenden Befehl des Programms (LD B, A) ausführen lassen. Dazu müssen Sie allerdings zuerst Ihr System wieder aus dem Register-Modus (Anzeigen oder Ändern von Register-Inhalten) in den Programm-Modus umschalten. Das geschieht einfach dadurch, daß die Adresse 18Ø2 des nächsten Befehls in die Anzeige gebracht wird. Die umständlichste, aber auch funktionierende Methode wäre es, diese Adresse nach dem Betätigen der Taste ADDR neu anzuwählen. Viel eleganter ist es, den Inhalt des Programmzählers mit der Taste PC zu aktivieren (Seiten H19 und

H 22). In der Anzeige erscheint dann wieder die Adresse 18Ø2 für den Befehl LD B, A mit dem Ihnen bereits bekannten Operationscode 47.

Betätigen Sie wieder die Taste STEP und lassen Sie diesen Befehl in einem Einzelschritt ausführen! Wiederholen Sie danach den eben beschriebenen Vorgang des Abfragens der Register-Inhalte. Sie stellen erwartungsgemäß fest, daß jetzt durch das Kopieren des Akkumulator-Inhalts in das B-Register auch dieses Register das Byte AB enthält. Die Inhalte der übrigen Register sind nach wie vor gleich ØØ.

Lassen Sie auf die hier beschriebene Weise nacheinander auch die übrigen Befehle im Bild H 26.1 ausführen und sehen Sie sich nach der Ausführung eines jeden Befehls die Register-Inhalte an! Nach der Ausführung des letzten Befehls steht in sämtlichen, hier betrachteten Registern der ursprünglich nur in den Akkumulator geladene Wert AB:

(A) = AB; (B) = AB; (C) = AB; (D) = AB, (E) = AB;
(H) = AB; (L) = AB

Der jetzt ausgeführte Versuch hat Sie mit folgenden Befehlen bekannt gemacht:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD D, A	57	D ← (A)	D-Register mit Inhalt von A laden
LD E, A	5F	E ← (A)	E-Register mit Inhalt von A laden
LD H, A	67	H ← (A)	H-Register mit Inhalt von A laden
LD L, A	6F	L ← (A)	L-Register mit Inhalt von A laden

Wir haben bereits mehrfach angedeutet, daß es nicht nur Kopier-Befehle zum Übertragen des Akkumulator-Inhalts in ein anderes Register gibt. Der Befehlssatz unseres Mikroprozessors bietet die Möglichkeit, den Inhalt jedes der im Bild H 23.1 dargestellten Register in jedes andere Register zu kopieren. Da wir es hier mit sieben Registern zu tun haben, läßt sich leicht ausrechnen, daß es insgesamt 7 x 7 = 49 solcher Kopier-Befehle geben muß. Im mnemonischen Code lassen sich diese Befehle auf die allgemeine Form

LD r, r'

bringen, wobei r das Zielregister und r' das Herkunftsregister ist.

Wir brauchen diese 49 Befehle nicht sämtlich in eigenen Versuchen vorzustellen. Das Prinzip haben Sie mit den Versuchen zu den Befehlen LD r, A bereits kennengelernt. Anmerken müssen wir jedoch, daß unter den 49 Befehlen zum Kopieren von Register-Inhalten sieben Befehle sind, die den eigenen Register-Inhalt zu sich selbst kopieren, z. B. der Befehl LD B, B. Solche scheinbar unsinnigen Befehle können in manchen Programmen trotzdem eine vernünftige Funktion erfüllen.

Wenn wir uns für einen Versuch die Aufgabe stellen, den Inhalt des Registers C in das H-Register zu kopieren, dann müssen wir vorab natürlich das C-Register mit einem definierten Wert, also mit einer Konstanten, laden. Dazu kennen Sie bisher zwei Methoden:

Sie können den konstanten Wert über das Tastenfeld mit Hilfe der Taste DATA laden, oder – was eleganter ist –
Sie können den konstanten Wert zunächst mit dem LD A, Konst in den Akkumulator laden und ihn von dort mit dem Befehl LD C, A ins C-Register kopieren.

Der Befehlssatz unseres Mikroprozessors bietet noch eine bequemere Methode. Er enthält insgesamt sieben Befehle, deren mnemonische Codes folgende, allgemeine Form haben:

LD r, Konst

Dabei ist r das Zielregister für die nachfolgende Konstante. r kann ein beliebiges der im Bild H23.1 dargestellten Register sein. Von diesen sieben möglichen Befehlen ist Ihnen der Befehl LD A, Konst mit dem Operationscode 3E bereits bekannt. Wie dieser Befehl sind auch die entsprechenden anderen sechs Befehle Zwei-Byte-Befehle: Sie müssen außer dem Operationscode noch ein zweites Byte als Operanden mit dem Wert der in das Register zu bringenden Konstanten enthalten.

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD A, Konst	3E	A ← Konst	
LD B, Konst	Ø6	B ← Konst	
LD C, Konst	ØE	C ← Konst	Das betreffende Re-
LD D, Konst	16	D ← Konst	gister wird mit einer
LD E, Konst	1E	E ← Konst	Konstanten geladen
LD H, Konst	26	H ← Konst	
LD L, Konst	2E	L ← Konst	

Sicher leuchtet Ihnen hier die Nützlichkeit der mnemonischen Codes ein. Sie haben jetzt schon soviele Befehle kennengelernt, daß es fast unmöglich ist, all die vielen, zugehörigen Operationscodes im Kopf zu behalten. Die mnemonischen Codes dagegen sprechen für sich selbst und sind leicht zu behalten. Häufig gebrauchte Operationscodes prägen sich Ihnen auf die Dauer ganz von selbst ein. Es ist aber wenig sinnvoll, diese Codes systematisch auswendig zu lernen. Programme schreibt man mit Hilfe mnemonischer Codes und sieht dann die zugehörigen Operationscodes in einer Liste nach.

Versuch H28.1

Inhalt des C-Registers in das H-Register kopieren

Tasten Sie bitte ab der Adresse 18ØØ die folgenden Befehle ein:

Operation	Operand	Adresse	Opcode	1. Operand
LD	C,34	18ØØ	ØE	34
LD	H,C	18Ø2	61	

Diese beiden Befehle sollen wieder in Einzelschritten abgearbeitet werden, damit Sie die Wirkungen der Befehle einzeln abfragen können. Löschen Sie dazu – wie im vorhergehenden Versuch beschrieben – mit Hilfe der Tasten REG, AF, DATA, 0 und + die Inhalte der Register A, B,

C, D, E, H und L und lassen Sie dann nach der Anwahl der Startadresse 18ØØ den Befehl LD C,34 in einem Einzelschritt ausführen. Die anschließende Abfrage der Register-Inhalte mit den Tasten REG, AF und + liefert

(A) = ØØ; (B) = ØØ; (C) = 34; (D) = ØØ; (E) = ØØ;
(H) = ØØ; (L) = ØØ

Holen Sie anschließend mit der Taste PC wieder die Adresse des nächsten Befehls (18Ø2, LD H,C, Operationscode 61) in die Anzeige und lassen Sie auch diesen Befehl in einem Einzelschritt ausführen. Die anschließende Kontrolle der Register-Inhalte liefert:

(A) = ØØ; (B) = ØØ; (C) = 34; (D) = ØØ; (E) = ØØ;
(H) = 34; (L) = ØØ

Wichtig ist bei der Ausführung dieser Befehlsfolge, daß die Inhalte von zwei Registern manipuliert wurden, ohne daß sich der Inhalt des zentralen Registers im Akkumulator geändert hat. Der Versuch zeigt, daß ein direkter Datentransport zwischen den einzelnen Registern möglich ist.

Wir empfehlen Ihnen, nach dem Durcharbeiten dieses Abschnitts, selbständig eine Reihe ähnlicher Versuche zum Datentransport zwischen den einzelnen Registern durchzuführen.

Aufgabe H29.1

Im Bild H26.1 haben wir ein Programm aufgelistet, das das Laden der Register A bis L mit dem konstanten Wert AB bewirkt (vgl. Versuch H25.1). Mit den LD r,Konst-Befehlen haben Sie eine Möglichkeit kennengelernt, jedes der Register unabhängig vom Inhalt eines anderen Registers mit einem konstanten Wert zu laden.

Tragen Sie in die Tabelle im Bild H29.1 eine Befehlsfolge ein, die unter Verwendung von LD r,Konst-Befehlen das Laden der Register A bis L mit dem konstanten Wert AB bewirkt. Ergänzen Sie auch die Adressen-Spalte!

Kontrollieren Sie Ihre Lösung durch einen Versuch, indem Sie das Programm in Einzelschritten abfahren.

Welches der Programme in den Bildern H26.1 und H29.1 würden Sie bevorzugen, obwohl beide Programme das gleiche bewirken? Warum?

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü7.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand
1		LD	B,ØACH	1800	06	AB
2			C	1802	0E	AB
3			D	1804	16	AB
4			E	1806	1E	AB
5			H	1808	26	AB
6			L	180A	2E	AB
7			A	180C	3E	

Bild H29.1
Zu Aufgabe H29.1. Tragen Sie in diese Tabelle eine Befehlsfolge ein, die mit LD r,Konst-Befehlen das Laden der Register A bis L mit dem Wert AB bewirkt.

Sicher vermissen Sie jetzt noch die Auflistung der Operationscodes für die 49 LD r,r'-Befehle. Eine solche Auflistung ergäbe an dieser Stelle des Lehrbriefs eine unpraktisch lange Liste. Wir machen es hier etwas übersichtlicher: Im Bild H30.1 finden Sie eine gedrängte Tabelle, der Sie die gewünschten Operationscodes entnehmen können.

Nun mag es sein, daß Sie am Ende dieses Abschnitts zumindest etwas verwundert sind – und haben dabei vielleicht nicht einmal ganz unrecht. Erschöpft sich denn tatsächlich die ganze Fähigkeit des Mikroprozessors darin, Zahlen in seinen diversen Registern umeinander zu schieben?

Ganz so ist es zwar nicht, aber viel Wahres ist daran. Wird denn in einer ordentlichen Buchhaltung viel anderes gemacht? Oh, gewiß, da wird addiert und subtrahiert und da werden vor allem schwerwiegende Entscheidungen gefällt, welche Zahlen unter welchen Bedingungen aus welchen Registern geholt und in welchem Register abgelegt werden müssen. Nur Mut: Für solch überaus schwierige Entscheidungen enthält unser Mikroprozessor auch noch ein paar Befehle. Und denen folgt er dann blitzschnell, mit absoluter Sicherheit und ohne Managerkrankheit. Und davon, daß unser Mikroprozessor auch rechnen kann, werden Sie sich noch überzeugen können.

Aufgabe H30.1

Lösen Sie mit drei Befehlen die Aufgabe, die Inhalte der B- und E-Register miteinander zu vertauschen.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü8.

	A	B	C	D	E	H	L
LD A,	7F	78	79	7A	7B	7C	7D
LD B,	47	40	41	42	43	44	45
LD C,	4F	48	49	4A	4B	4C	4D
LD D,	57	50	51	52	53	54	55
LD E,	5F	58	59	5A	5B	5C	5D
LD H,	67	60	61	62	63	64	65
LD L,	6F	68	69	6A	6B	6C	6D

Bild H30.1
Die Operationscodes der Befehle LD r,r'. Den Operationscode des Befehls LD C,H=4C z.B. finden Sie nach dem eingetragenen Muster.

Der Aufbau des Mikroprozessors (Fortsetzung)

Es besteht durchaus kein Widerspruch zwischen unseren Behauptungen, daß jeder Taschenrechner bei der Lösung umfangreicher Rechenaufgaben einem Mikroprozessor weit überlegen ist, und daß doch Zahlen die wichtigsten Dinge für einen Mikroprozessor sind.

Wir werden Ihnen im folgenden Abschnitt zeigen, daß sedezimal und dual dargestellte Zahlen elegante Hilfsmittel zur Darstellung und zum Verständnis der Arbeitsweise des Mikroprozessors sind.

Die Register als binäre Zahlenspeicher

Den Akkumulator und die Register B bis L (Bild H 23.1) haben Sie bisher schlicht als Ablagen für sedezimal dargestellte Zahlen kennengelernt. Wie diese Register, die doch offenbar elektronische Gebilde sein müssen, die Zahlen aufbewahren und wie sie beschaffen sind – darüber haben wir uns bisher noch keine Gedanken gemacht.

Auf der Seite H 14 haben wir angedeutet, daß die Zahlen in den Registern binär dargestellt werden. Was heißt **binär**? Das Wort kommt vom lateinischen *bini* und bedeutet ein Paar. Gemeint ist hier ein Paar von zwei unterschiedlichen Zeichen, mit denen sich Zahlen im Dualsystem darstellen lassen. Die dabei verwendeten Zeichen Ø und 1 werden deshalb als **Binärzeichen** bezeichnet: Es sind Zeichen aus einem Zeichenvorrat mit zwei unterschiedlichen Zeichen.

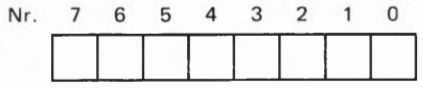


Bild H 31.1
In dieser Darstellung eines Registers symbolisiert jedes der von 0 bis 7 numerierten Kästchen ein Flipflop.

Zur Ablage solcher Zeichen in einem Register ist das elektronische Bauelement Flipflop als Speicherglied mit zwei stabilen Zuständen bestens geeignet. Wie die Halbleiterstruktur eines Registers tatsächlich aussieht und aus welcher Art von Flipflop die Register bestehen, ist hier ganz unwichtig. Es reicht die Vorstellung, daß die Ihnen bis jetzt bekannten Register aus jeweils acht einzelnen Flipflops bestehen, die gesetzt oder rückgesetzt sein können. Sie liefern im gesetzten Zustand 1-Signale und im rückgesetzten Zustand Ø-Signale.

Diese Vorstellung führt uns zu einer Darstellungsweise der Register, wie es im Bild H 31.1 gezeigt ist. Jedes der acht Kästchen stellt ein Flipflop dar. Bei Bedarf können in die Kästchen die Ziffern Ø und 1 eingetragen werden, mit denen man die Zustände der Flipflops darstellt. Zunächst nur zur leichteren Verständigung haben wir die einzelnen Kästchen von rechts nach links – mit der Nr. Ø beginnend – numeriert.

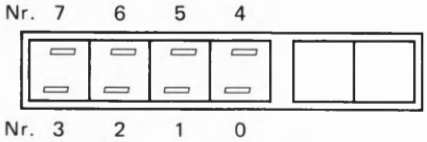


Bild H 31.2
In den vier linken Anzeigestellen werden die hier gezeigten Segmente zur Anzeige der Flipflop-Zustände in den Registern des Mikroprozessors benutzt.

Mit einem entsprechenden Programm können wir den Mikroprozessor veranlassen, uns die Zustände in einem seiner Register – wir wählen den Akkumulator – optisch vorzuführen. Wir benutzen dazu die linken vier Stellen der Anzeige, wie es das Bild H 31.2 zeigt. Interessant sind dabei nur die im Bild als nicht leuchtende Balken angedeuteten Segmente. Die eingetragenen Nummern zeigen, wie die Segmente den Flipflops in den Registern zugeordnet sind. Ein leuchtendes Segment

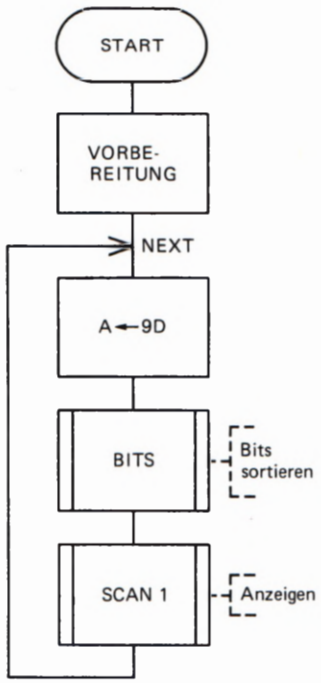


Bild H32.1
Programmablaufplan zum Versuch H32.1. Das Unterprogramm BITS ist im Bild H33.2 aufgelistet. Das Unterprogramm SCAN1 ist in Ihrem System intern abgelegt.

stellt ein gesetztes Flipflop dar (1-Signal), ein dunkles Segment ein rückgesetztes Flipflop (Ø-Signal).

Versuch H32.1

Die Flipflops im Akkumulator

Tasten Sie ab der Adresse 19ØØ das im Bild H33.2 aufgelistete Programm ein, das die optische Anzeige der Flipflop-Zustände im Akkumulator ermöglicht. Starten Sie dieses Programm aber bitte noch nicht, denn zunächst haben wir ja noch kein Byte in den Akkumulator eingetragen. Wir nennen dieses Programm BITS. Es kann als Unterprogramm (Seite S6) aufgerufen werden.

Das Bild H32.1 zeigt den Ablaufplan des Hauptprogramms, in dem das Unterprogramm BITS aufgerufen wird. Die erste dort eingetragene Anweisung braucht uns nicht zu interessieren. Sie ist die Vorbereitung für das Unterprogramm SCAN1, das in Ihrem System zur Bedienung der Anzeige fest abgelegt ist und dessen wir uns hier bedienen, um die Flipflop-Zustände in die Anzeige zu bekommen.

Die folgende Anweisung sorgt dafür, daß der Akkumulator mit einem definierten Byte geladen wird. Wir haben willkürlich das Byte 9D gewählt. Anschließend wird das Unterprogramm BITS aufgerufen, mit dem die Flipflop-Zustände für die anschließende Anzeige mit dem Unterprogramm SCAN1 sortiert werden.

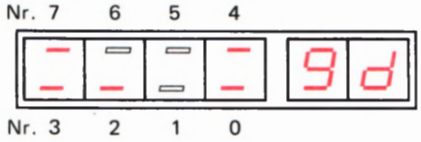
Nach dem Abarbeiten dieser Anweisungen ist das Programm eigentlich beendet. Damit die Flipflop-Anzeige aber dann nicht durch die System-Meldung überschrieben wird, lassen wir das Programm in einer **Schleife** (Seite S5) laufen: Abgesehen von der Anweisung VORBEREITUNG wird das Programm immer von neuem ausgeführt. Am Ende des Programms wird dazu ein **Sprungbefehl** programmiert, der das Programm bei der Adresse der Lade-Anweisung für den Akkumulator (LD A,9D) neu starten läßt. Da diese Adresse für das Programm von besonderer Wichtigkeit ist, geben wir ihr einen eigenen Namen: Wir nennen sie NEXT (Abkürzung des englischen *NEXT time*, nächstesmal). Solche Namen für wichtige Adressen werden beim Programmieren recht oft verwendet. Man bezeichnet sie als **Label** (sprich: läibel). Dieses englische Wort bedeutet Etikett.

Für die Sprunganweisung versteht unser Mikroprozessor einen eigenen Befehl, dessen mnemonischer Code JP (Abkürzung des englischen **J***um***P** – sprich dschamp –, springe) ist. Auf diesen mnemonischen Code folgen dann – in umgekehrter Reihenfolge – die beiden Bytes der Adresse NEXT. Es handelt sich um einen Drei-Byte-Befehl, aber darauf kommen wir noch zurück.

Unbekannt ist Ihnen auch noch der Befehl mit dem mnemonischen Code CALL zum Aufrufen von Unterprogrammen. Das englische Wort *call* (sprich kool, jedoch mit „offenem" o, wie Rolle) bedeutet wörtlich rufen. Es handelt sich ebenfalls um einen Drei-Byte-Befehl. Das zweite und dritte Byte enthält – wieder in umgekehrter Reihenfolge – die Startadresse des (auf)gerufenen Unterprogramms.

Im Bild H33.1 haben wir das Hauptprogramm in vollständiger Form aufgelistet. Tasten Sie dieses Programm jetzt bitte in Ihr System ein.

a)



b)

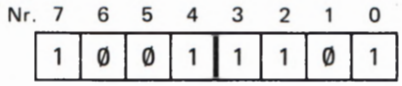


Bild H32.2
a) So sieht die Anzeige aus, die der Versuch H32.1 liefert.
b) Der Akkumulator enthält das hier dargestellte Bitmuster.

Nr.	Label	Operation	Operand	Adresse	Opcode	Opcode Operand	Operand	Operand	Bemerkungen
1	START	LD	IX, BUFF	18ØØ	DD	21	ØØ	1A	Vorbereitung
2	NEXT	LD	A, 9D	18Ø4	3E	9D			(A) = Byte
3		CALL	BITS	18Ø6	CD	ØØ	19		Bits sortieren
4		CALL	SCAN1	18Ø9	CD	24	Ø6		Anzeigen
5		JP	NEXT	18ØC	C3	Ø4	18		Wiederholen

Bild H33.1
Hauptprogramm zur Anzeige der Flipflop-Zustände im Akkumulator im Versuch H32.1.

Für die Eingabe interessieren nur die Werte im rot unterlegten Teil der Liste. Starten Sie das Programm bei der Adresse 18ØØ. Sie erhalten die im Bild H32.2a dargestellte Anzeige.

Überträgt man die Anzeige in eine Darstellung des Registers Akkumulator, dann zeigt das Teilbild b die Zustände der acht Flipflops. Ganz offenbar besteht ein enger Zusammenhang zwischen der sedezimal durch 9D**H** dargestellten Zahl, die wir in den Akkumulator geladen haben und dem Binärzeichen-Muster im Bild H32.2b.

Ehe wir diesen – sehr einfachen – Zusammenhang aufdecken, wollen wir uns noch auf eine Abkürzung einigen. Statt des langen Worts Binärzeichen, das wir noch oft gebrauchen müßten, verwenden wir in Zukunft einfach die Abkürzung **Bit** des englischen Worts **BI***nary digi***T** für Binärzeichen. Auch die Zeichen Ø und 1 aus dem Dualsystem, die ja ebenfalls Binärzeichen sind, werden als Bit bezeichnet.

Der Inhalt des Akkumulators, der jetzt in der Anzeige Ihres Systems dargestellt ist, ist also – wie auch der Inhalt jedes anderen Registers – ein **Bitmuster**, gebildet durch jeweils einen der beiden Zustände von Flipflops. Dieses Bitmuster steht im Zusammenhang mit der Sedezimaldarstellung einer Zahl.

9DH ist die Sedezimaldarstellung der Zahl einhundertsiebenundfünfzig. Genau diese Zahl wird auch durch das von Ihrem System angezeigte Bitmuster mit 1ØØ111Ø1**B** dargestellt. Es ist also:

1ØØ111Ø1**B** = 9D**H**

Daraus folgt:

Adresse	Opcode	Opcode Operand	Operand
19ØØ	4F		
1901	21	ØØ	1A
1904	CD	78	Ø6
1907	Ø6	Ø4	
1909	CB	Ø9	
19ØB	79		
19ØC	E6	88	
19ØE	77		
19ØF	23		
1910	1Ø	F7	
1912	C9		

Bild H33.2
Unterprogramm BITS zum Sortieren der Bits im Akkumulator für die Anzeige.

> Die Sedezimaldarstellung der Zahlen, die in den Registern des Mikroprozessors abgelegt sind, entspricht den Bitmustern, die von den Flipflops der Register geliefert werden.

Beim Programmieren betrachtet man die Bitmuster in den CPU-Registern als dual dargestellte Zahlen und operiert mit diesen Zahlen in sedezimaler Schreibweise. Das ist bequem, weil sich zwei Sedezimalzeichen, die das Muster der acht Bits darstellen können, viel leichter handhaben lassen als das Bitmuster selbst.

Vier Binärzeichen lassen sich gerade zu sechzehn unterschiedlichen Kombinationen zusammenstellen. Im Bild H34.1 haben wir diese Kombinationen aufgelistet. Wir betrachten jede dieser Kombinationen als die Dualdarstellung einer Zahl, wobei vorgestellte Zeichen Ø die

Dual				Sede-zimal	Dezi-mal
Ø	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	1	1
Ø	Ø	1	Ø	2	2
Ø	Ø	1	1	3	3
Ø	1	Ø	Ø	4	4
Ø	1	Ø	1	5	5
Ø	1	1	Ø	6	6
Ø	1	1	1	7	7
1	Ø	Ø	Ø	8	8
1	Ø	Ø	1	9	9
1	Ø	1	Ø	A	10
1	Ø	1	1	B	11
1	1	Ø	Ø	C	12
1	1	Ø	1	D	13
1	1	1	Ø	E	14
1	1	1	1	F	15

Bild H34.1
Jede der sechzehn möglichen Kombinationen von vier Binärzeichen läßt sich eindeutig einem der sechzehn unterschiedlichen Zeichen des Sedezimalsystems zuordnen.

dargestellte Zahl nicht beeinflussen (ØlØl = lØl). Sie erkennen, daß sich bei der Sedezimaldarstellung immer gerade ein Zeichen für jede Kombination der vier Binärzeichen ergibt. Hier liegt der Vorteil gegenüber der Dezimal-Darstellung, die in sechs Fällen die Kombination von zwei Zeichen erfordert.

> Jedes der sechzehn Zeichen des Sedezimalsystems läßt sich einer der sechzehn Vier-Bit-Kombinationen des Dualsystems zuordnen.

Die Inhalte der Acht-Bit-Register des Mikroprozessors lassen sich in zwei Vier-Bit-Gruppen einteilen; jede dieser Vier-Bit-Gruppen läßt sich eindeutig durch ein Zeichen des Sedezimalsystems beschreiben. Das Bild H34.2 zeigt das am Beispiel (A) = 1ØØ1 11Ø1. Jedem einzelnen Bit wird eine Nummer zugeteilt, die der Potenz von zwei im dualen Stellenwertsystem entspricht. Häufig wird das Bit Nr. 7 als **M***ost* **S***ignificant* **B***it* (sprich: moust ssignifikent bit, höchstwertiges Bit) oder abgekürzt als **MSB** bezeichnet. Dementsprechend nennt man das ganz rechts stehende Bit **L***east* **S***ignificant* **B***it* (sprich: liesst .., niedrigstwertiges Bit) und kürzt das mit **LSB** ab. (Unsere Numerierung im Bild H31.1 war also nicht willkürlich!)

Das ganze Bitmuster in einem Register bezeichnet man als **Wort**.

> Ein Wort ist eine Folge von Binärzeichen (Bits), die in einem bestimmten Zusammenhang als Einheit betrachtet wird.

Ein Acht-Bit-Wort wird als **Byte** (sprich: beit) bezeichnet; es ist durch zwei Sedezimalzeichen charakterisiert.

Zur Eingabe eines Bytes in unser System müssen immer zwei Zifferntasten nacheinander betätigt werden. Die erste Taste liefert das höherwertige Halbbyte; mit der zweiten Tastenbetätigung wird das niederwertige Halbbyte eingegeben. Das System setzt diese beiden Halbbytes automatisch zu einem vollständigen Byte zusammen.

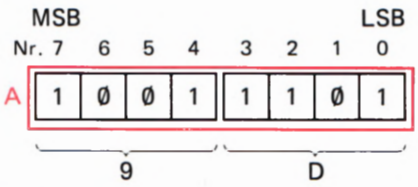


Bild H34.2
Der Inhalt eines Acht-Bit-Registers läßt sich durch zwei Zeichen des Sedezimalsystems angeben. Die Nummern der Bits entsprechen den Potenzen von zwei im dualen Stellenwertsystem.

Versuch H34.1

Eingabe von zwei Halbbytes

Dieser Versuch entspricht dem vorhergehenden Versuch, mit dem Unterschied, daß der Akkumulator nicht per Programm, sondern über das Tastenfeld mit einem Byte geladen wird. Sie brauchen dazu nur das Hauptprogramm entsprechend zu ändern; das im Bild H33.2 aufgelistete Unterprogramm BITS kann unverändert weiter verwendet werden.

Das Bild H35.1 zeigt das in diesem Versuch ab der Adresse 18ØØ verwendete Hauptprogramm. Die Funktion dieses Programms können Sie jetzt noch nicht übersehen. Interessant ist, daß wir statt des Anzeige-Unterprogramms SCAN1 jetzt für jede Eingabe einmal das System-interne Unterprogramm SCAN aufrufen, das außer der Bedienung der Anzeige auch die Abfrage des Tastenfelds übernimmt. Auch dieses Hauptprogramm läuft in einer Schleife. Beachten Sie bitte,

Nr.	Label	Operation	Operand	Adresse	Opcode	Opcode Operand	Operand	Operand	Bemerkungen
1	START	LD	IX, BUFF	18ØØ	DD	21	ØØ	1A	Vorbereitung
2	NEXT	CALL	SCAN	18Ø4	CD	FE	Ø5		1. Ziffer
3		CALL	BITS	18Ø7	CD	ØØ	19		Bits sortieren
4		CALL	SCAN	18ØA	CD	FE	Ø5		2. Ziffer
5		OR	C	18ØD	B1				(A) = 2 Ziffern
6		CALL	BITS	18ØE	CD	ØØ	19		Bits sortieren
7		JR	NEXT	1811	18	F1			Nächstes Byte

Bild H35.1
Dieses Hauptprogramm zum Versuch H34.1 ist nur dann lauffähig, wenn auch das Unterprogramm BITS im Bild H33.2 eingegeben wurde.

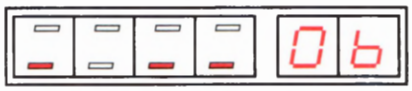
daß wir diesmal als letzten nicht den Drei-Byte-Sprungbefehl JP programmiert haben, dessen zweites und drittes Byte in umgekehrter Reihenfolge die Bytes der Adresse NEXT enthält; wir haben den Zwei-Byte-Sprungbefehl mit dem mnemonischen Code JR verwendet. Der Operand dieses Befehls sagt dem Programm, daß es seine Arbeit fünfzehn Adressen weiter zurück wieder aufnehmen soll. JR ist die Abkürzung des englischen ***J**ump **R**elative* (sprich: dschamp relativ): Springe relativ (zur derzeitigen Adresse).

Tasten Sie das Programm entsprechend dem rot unterlegten Feld im Bild H35.1 ein und starten Sie es bei der Adresse 18ØØ! Übersehen Sie die jetzt erscheinende Anzeige und betätigen Sie die Zifferntaste B! Die sich jetzt einstellende Anzeige ist im Bild H35.2a dargestellt. Ihr System interpretiert die Eingabe als niederwertiges Halbbyte. Die unteren Balken der linken Anzeigestellen zeigen das zugehörige Bitmuster.

Tasten Sie jetzt die Ziffer A ein! Ihr System interpretiert auch diese Eingabe als niederwertiges Halbbyte. Vorher wird jedoch die zuerst eingegebene Zahl Elf zum höherwertigen Halbbyte befördert und auch das zugehörige Bitmuster nach oben geschoben. Sie haben jetzt ein komplettes Byte entsprechend der Darstellung im Teilbild b eingegeben.

Das Bild H35.3 erläutert diesen Vorgang noch einmal anhand der Darstellung eines Registers. – Tasten Sie nacheinander unterschiedliche Bytes ein und sehen Sie sich die Bitmuster an!

a)



b)

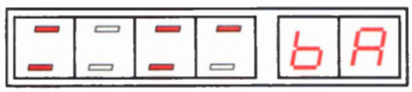


Bild H35.2
a) Anzeige nach der 1. Tasten-Eingabe, b) Anzeige nach der 2. Tasten-Eingabe für ein Byte im Versuch H34.1.

Registerpaare für 16 Bits

Der in unserem System verwendete Mikroprozessor wird als Acht-Bit-Mikroprozessor bezeichnet, weil die meisten seiner Register acht Datenbits verarbeiten und auch die Daten-Kommunikation mit der Peripherie, also mit der Außenwelt des Mikroprozessors, über acht parallele Datenleitungen, den sogenannten **Datenbus**, vorgenommen wird. Die Darstellung der Architektur auf der Seite T2 zeigt Ihnen jedoch, daß unser Mikroprozessor auch über vier sechzehn Bit „breite" Register verfügt, von denen uns in diesem Lehrgang jedoch nur zwei beschäftigen werden. Diese beiden Register haben allerdings etwas andere Aufgaben als die Ihnen jetzt bekannten Register A bis L und als das später noch vorzustellende Flag-Register. Der Programmzähler und der Stackpointer dienen zur Manipulation von Adressen, die sedezimal vierstellig dargestellt werden.

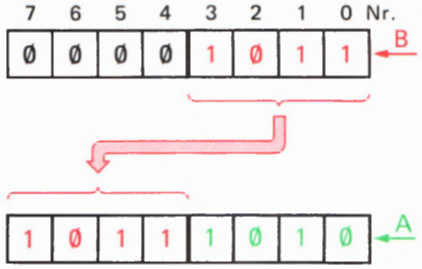


Bild H35.3
So wird beim Versuch H34.1 ein Byte aus zwei Tasten-Eingaben zusammengesetzt.

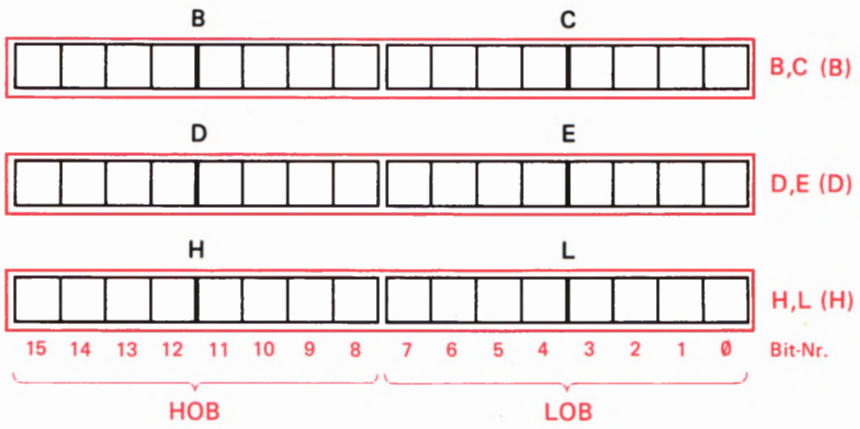


Bild H36.1
Der Z80-Mikroprozessor kann die hier dargestellten Registerpaare wie Sechzehn-Bit-Register behandeln.

Aufgabe H36.1

Welches ist die größte Zahl, die in einem sechzehn Bit „breiten" Register abgelegt werden kann?

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü8.

Trotz seiner grundsätzlichen Acht-Bit-Struktur bietet unser Mikroprozessor die Möglichkeit, intern auch sechzehn Bit „breite" Daten zu verarbeiten. Wir haben das bereits auf der Seite H23 angedeutet und es auch bei der Darstellung der Architektur berücksichtigt (Bild H23.1 und Seite T2): Die Register B und C, D und E sowie H und L können von bestimmten Befehlen jeweils als **Registerpaare** betrachtet und – sozusagen miteinander vereint – als ein einziges, sechzehn Bit „breites" Register behandelt werden. Diese Registerpaare erhalten in den mnemonischen Befehlscodes Doppelnamen, die sich aus den Namen der Einzelregister zusammensetzen: BC, DE und HL.

Jedes dieser drei Registerpaare (rot im Bild H36.1) nimmt zwei Bytes auf, die in der Fachsprache folgende Bezeichnungen haben:

LOB = **L***ower* **O***rder* **B***yte* (sprich: louer order beit)
Byte niederer Ordnung mit den Stellenwerten 16^0 und 16^1

HOB = **H***igher* **O***rder* **B***yte* (sprich: haier order beit)
Byte höherer Ordnung mit den Stellenwerten 16^2 und 16^3

Es sei deutlich darauf hingewiesen: Die Tatsache, daß die Register B bis L Bestandteile von Registerpaaren sind, beeinträchtigt in keiner Weise ihre Eigenständigkeit als Acht-Bit-Register! Diese Register können also im einen Befehl als Bestandteile eines Registerpaares angesprochen werden; im nächsten Befehl kann dann eins dieser Register als selbständiges Acht-Bit-Register gemeint sein.

Die einfachsten Befehle, die ein Registerpaar betreffen, entsprechen den LD r,Konst-Befehlen für Acht-Bit-Register:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD BC,Konst	Ø1	BC ← Konst	Das Registerpaar wird
LD DE,Konst	11	DE ← Konst	mit der sedezimal vier-
LD HL,Konst	21	HL ← Konst	stellig dargestellten Zahl Konst geladen.

Bei diesen drei Befehlen handelt es sich um **Drei-Byte-Befehle**. Das erste Byte ist jeweils der Operationscode. Es folgen dann zwei Operanden, deren jeder ein Byte für das zwei Byte breite Registerpaar enthält.

Für die beiden Operanden in drei-Byte-Befehlen gilt für unseren Mikroprozessor eine **sehr wichtige Regel**:

> In Drei-Byte-Befehlen bilden der 1. und der 2. Operand insgesamt eine sedezimal vierstellig dargestellte Zahl (zwei Bytes).
>
> Der **erste Operand** enthält das **LOB**, also die 16^0 - und die 16^1 - Stellen dieser Zahl.
>
> Der **zweite Operand** enthält das **HOB**, also die 16^2 - und die 16^3 - Stellen dieser Zahl.

In den beiden Operanden von Drei-Byte-Befehlen sind die Ziffern einer sedezimal dargestellten Zahl paarweise vertauscht.

Sie sollen in einem Versuch sich als Beispiel für diese Befehle den Befehl LD BC,12AB ansehen. Dieser Befehl bewirkt das Laden des BC-Registerpaares mit einer Zahl, die sedezimal durch 12AB**H** dargestellt wird. Wohlgemerkt: In der mnemonischen Schreibweise von Drei-Byte-Befehlen wird die Zahl im Operanden ganz normal angeschrieben.

Versuch H37.1

Das Laden eines Registerpaares

Löschen Sie zunächst die Inhalte der Register A bis L unter Benutzung der Tasten REG, AF, DATA und abwechselnd der Tasten 0 und +.

Tasten Sie anschließend die zum folgenden Befehl gehörenden Bytes ab der Adresse 18ØØ ein:

Operation	Operand	Adresse	Opcode	1. Operand	2. Operand
LD	BC, 1 2 (HOB) A B (LOB)	18ØØ	Ø1	AB (LOB)	12 (HOB)

Lassen Sie den Befehl in einem Einzelschritt ablaufen (ADDR, 1, 8, 0, 0, STEP) und sehen Sie sich anschließend die Inhalte der Register an! Sie brauchen dazu nur einmal die Taste REG und anschließend der Reihe nach die Tasten AF, BC, DE und HL zu betätigen.

Der Versuch zeigt, daß Sie ohne Inanspruchnahme des zentralen Registers Akkumulator mit einem einzigen Drei-Byte-Befehl das Register B mit dem Wert 12**H** und das Register C mit dem Wert AB**H** geladen haben. Das Bild H37.1 zeigt das mit 12AB**H** geladene Registerpaar BC.

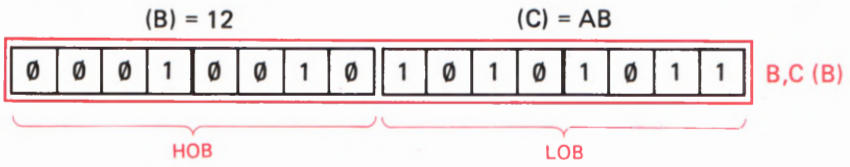


Bild H37.1
Im Versuch H37.1 wird das Registerpaar BC mit dem hier dargestellten Bitmuster geladen.

H
38

Der Versuch macht außerdem die Zweckmäßigkeit der Anzeige der Register-Inhalte in Ihrem System deutlich: Es werden jeweils die Inhalte von Registerpaaren angezeigt. Sie werden noch erfahren, warum es auch zweckmäßig ist, den Akkumulator und das Flag-Register in Form eines Registerpaares anzuzeigen, obwohl es sich bei diesen beiden Registern nicht um ein echtes Registerpaar handelt.

Für die hier betrachteten Sechzehn-Bit-Registerpaare innerhalb des Haupt-Registersatzes steht nur ein Befehl zur Verfügung, der den Befehlen LD r,r' bei den Acht-Bit-Registern entspricht:

Mnemonischer Code	Operations-code	Operation	Erläuterung
EX DE,HL	EB	(DE) ⇄ (HL)	Vertauschen der Inhalte der Registerpaare DE und HL

Der mnemonische Code EX kommt vom englischen **EX***change* (sprich: ixtschensch). Das bedeutet wörtlich: Austauschen.

Da die auszutauschenden Werte bei diesem Befehl bereits als Register-Inhalte vorliegen, braucht dieser Befehl keinen Operanden. Er ist – gerade wie die LD r,r'-Befehle – ein Ein-Byte-Befehl. Das Bild H38.1 erläutert die Wirkung dieses Befehls.

Versuch H38.1

Inhalte der Registerpaare DE und HL vertauschen

Löschen Sie die Inhalte der Register A bis L und tasten Sie die folgende Befehlsfolge ein:

Nr.	Operation	Operand	Adresse	Opcode	1.Operand	2.Operand
1	LD	DE,8E3B	18ØØ	11	3B	8E
2	LD	HL,Ø1FA	18Ø3	21	FA	Ø1
3	EX	DE,HL	18Ø6	EB		

Lassen Sie das Programm in Einzelschritten ablaufen und kontrollieren Sie nach jedem Einzelschritt die Register-Inhalte! Nach dem zweiten Befehl sind die Register so geladen, wie es das Bild H38.1 oben zeigt. Das Bild zeigt in der unteren Hälfte die Register-Inhalte nach der Ausführung des dritten Befehls.

Bild H38.1
Mit dem Befehl EX DE,HL werden die Inhalte der Registerpaare DE und HL miteinander vertauscht.

Speicher und Adressen im Mikroprozessor-System

Sicher haben Sie sich schon gefragt: Wo bleiben eigentlich die bei einer Programmeingabe eingetasteten Befehle? Offenbar werden sie im Mikroprozessor-System gespeichert, denn sie können ja nach der Eingabe einer Adresse jederzeit wieder „aufgerufen" und in der Anzeige sichtbar gemacht werden. Wo und wie werden diese Acht-Bit-Worte gespeichert? – Diese Frage berührt ein wichtiges Thema, mit dem wir uns beschäftigen wollen.

Die Daten im Mikroprozessor-System

Der Haupt-Registersatz Ihres Systems enthält acht Acht-Bit-Register, einschließlich des in seiner Funktion noch nicht besprochenen Flag-Registers. Zusätzlich gibt es die sechzehn Bit „breiten" Register Programmzähler und Stackpointer, deren Funktion wir Ihnen ebenfalls später vorstellen. Auf der Seite T 2 erkennen Sie, daß in der CPU noch zwei weitere Sechzehn-Bit-Register, acht Register des zweiten Registersatzes und außerdem die Register I und R existieren, die uns in diesem Lehrgang nicht beschäftigen können.

Wenn Sie nachzählen, dann stellen Sie fest, daß in der CPU insgesamt 26 Bytes verarbeitet werden können. Welche Funktion all' diese Bytes auch immer haben mögen, auf keinen Fall können sie die vielen Funktionen der Befehlsbytes übernehmen, die z. B. zum Uhrenprogramm (Bild H 9.1) gehören, abgesehen davon, daß die Register in der CPU ganz andere Funktionen haben und zum Bearbeiten der laufenden Befehle dringend benötigt werden.

Unsere Überlegung zeigt, daß ein Mikroprozessor-System neben der CPU eines zusätzlichen Speichers bedarf, in dem zumindest die Bytes aufbewahrt werden können, die das Programm selbst darstellen.

> Ein Speicher ist eine Funktionseinheit innerhalb eines digitalen Rechensystems, die Daten aufnimmt, aufbewahrt und abgibt.

Was sind denn nun Daten? Sie lassen sich so definieren:

> Daten sind Zeichen, die zum Zweck der Verarbeitung irgendwelche Mitteilungen enthalten.

Das Mikroprozessor-System ist ein (wenn auch kleines) Datenverarbeitungs-System. Beim Arbeiten mit dem System füttern wir es mit Daten irgendwelcher Art. Das System verarbeitet diese Daten und gibt selbst wieder Daten mit dem von uns gewünschten Ergebnis aus.

In unserem Mikroprozessor werden Daten nur in Form von Acht-Bit-Worten verarbeitet. Gleichgültig, mit welcher Art von Daten wir unser

System füttern: Immer handelt es sich um Acht-Bit-Worte. Auch dann, wenn von diesen Bits nur ein einziges Bit interessant ist.

Bei Eingaben in unser System handelt es sich um ganz unterschiedliche Daten: Zunächst wird das aus Daten bestehende Programm eingegeben und anschließend Zahlen, die – beim Versuch H34.1 z. B. für eine Anzeige – verarbeitet werden.

Offensichtlich arbeitet das System mit zwei unterschiedlichen Arten von Daten: Mit Programm-Daten und mit solchen, die zu verarbeitende Zahlen oder Bitmuster darstellen.

Beide Arten von Daten muß das System speichern und beide Arten können in so großen Mengen anfallen, daß sie von den Arbeitsregistern der CPU (Akkumulator, Register B bis L usw.) niemals aufbewahrt werden können.

Für den wie auch immer gearteten Datenspeicher, den das System benötigt, können wir zweierlei feststellen:

1. Gleichgültig, ob es sich um Programm-Daten handelt oder um Daten, die zu verarbeitende Zahlen oder Bitmuster enthalten: Die grundsätzliche Struktur des Speichers ist in beiden Fällen gleich. Der Speicher muß eine Menge von Acht-Bit-Worten fassen können.

2. Der Inhalt eines Speichers, der nur Programm-Daten enthält, braucht sich beim Ablauf eines Programms nicht zu ändern. Das Programm ist ja für einen bestimmten Zweck festgelegt. – Der Inhalt eines Speichers für Daten von zu verarbeitenden Zahlen oder Bitmustern wird sich beim Ablauf eines Programms dauernd ändern. Solche Daten sind z. B. eingegebene und als Bitmuster anzuzeigende Bytes.

Festspeicher

Unser System enthält dementsprechend zwei unterschiedliche Arten von Speichern: **Festspeicher** und **Schreib-Lese-Speicher**. Der grundsätzliche Aufbau beider Speicherarten ist gleich: Sie können eine Menge von Acht-Bit-Worten aufnehmen. Ihre Unterschiede werden durch ihre Bezeichnungen gekennzeichnet:

> Der Inhalt eines Festspeichers wird vor der Inbetriebnahme festgelegt und kann – von Ausnahmen abgesehen – nicht mehr geändert werden. Sein Inhalt kann im Betrieb nur gelesen werden.

Festspeicher werden mit der Abkürzung des englischen **R***ead* **O***nly* **M***emory* (sprich: riid ounli memorie) als **ROM** bezeichnet. Der Inhalt dieser ROMs wird bereits bei der Herstellung nach den Wünschen des Anwenders unveränderbar festgelegt.

In der Praxis werden weit häufiger sogenannte **EPROM**s verwendet. Diese Abkürzung kommt vom englischen **E***rasable,* **P***rogrammable* **R***ead* **O***nly* **M***emory:* Löschbarer, programmierbarer Festspeicher. Sie können vom Anwender mit einem festen Inhalt versehen werden, sind aber mit besonderen Maßnahmen wieder löschbar und können dann mit einem anderen Inhalt gefüllt werden. Während des Betriebs im

System bleibt der Inhalt dieser EPROMs allerdings unbeeinflußbar. Und – besonders wichtig! – ihr Inhalt ändert sich auch dann nicht, wenn die Betriebsspannung abgeschaltet wird.

Aus dieser Beschreibung der Festspeicher geht hervor, daß sie nur zur Speicherung von Programm-Daten geeignet sind, dafür dann aber auch besonders gut. Ein in einem solchen Speicher festgelegtes Programm braucht nicht jedesmal nach dem Einschalten des Systems neu eingegeben zu werden.

In Ihrem System ist das **Betriebsprogramm** in einem Festspeicher enthalten (vgl. Seite T1). Dieses Programm ermöglicht die Eingabe von Daten über das Tastenfeld, steuert die Sieben-Segment-Anzeige, macht die Anzeige von Daten in den CPU-Registern möglich usw. Die Unterprogramme SCAN und SCAN1 in den Versuchen H32.1 und H34.1 sind z. B. Teile des Betriebsprogramms. Ein solches Betriebsprogramm muß unbedingt in einem Festspeicher abgelegt sein, damit das System nach dem Einschalten überhaupt arbeiten kann.

Wenn man ein Mikroprozessor-System praktisch einsetzen will, z. B. für die auf der Seite H3 genannte Aufzugsteuerung, dann gehört zur entsprechenden Hardware unbedingt ein auf diese Aufgabe zugeschnittenes Programm. Ist dieses Programm einmal fertig, dann wird man es in einem Festspeicher hinterlegen. Vielleicht ist es gar so vielseitig geraten, daß es unverändert für fünfhundert oder gar tausend gleiche Steuerungen geeignet ist. Dann kann man beim Hersteller ein ROM mit diesem Programm fertigen lassen. Bei kleineren Stückzahlen verwendet man vorzugsweise selbst programmierte EPROMs.

Schreib-Lese-Speicher

Ehe man daran denken kann, ein Programm in einem Festspeicher festzulegen, muß es während der Entwicklung in einem Speicher abgelegt werden, dessen Inhalt mit Hilfe des Betriebsprogramms leicht veränderbar ist. Für diesen Zweck verwendet man Schreib-Lese-Speicher, die man mit der Abkürzung **RAM** des englischen **R***andom* **A***ccess* **M***emory* bezeichnet (sprich: rändem ekssess memorie). Frei übersetzt bedeutet das: Speicher mit wahlfreiem Zugriff zu den einzelnen Speicherzellen. Die deutsche Bezeichnung Schreib-Lese-Speicher kennzeichnet die Eigenschaft dieser Bauelemente weit besser.

RAMs sind sehr wendige Bauelemente. Sie können sowohl die Programm-Daten des zu entwickelnden Programms aufnehmen als auch solche Daten, die beim Ablauf eines Programms anfallen. Das betrifft z. B. Informationen über das Stockwerk, in dem sich der Aufzug gerade befindet, oder auch Daten für das Betriebsprogramm, z. B. Informationen über die gerade betätigte Taste.

In zweierlei Hinsicht sind RAMs allerdings mit Vorsicht zu genießen: Beim Abschalten der Betriebsspannung geht ihr Inhalt verloren. Und: Beim Entwickeln eines Programms kann eine nicht genau überlegte Eingabe das mühsam eingetastete Programm zerstören. Wenn man es „geschickt" anstellt, dann kann eine falsche Eingabe das Programm, das man entwickeln möchte, zum Selbstmord treiben; es programmiert sich selbst zu Tode, indem es den ganzen Speicherinhalt löscht.

Adresse	Opcode	Opcode Operand	Operand
1824	21	ØØ	18
1827	36	ØØ	
1829	2C		
182A	18	FB	

Bild H42.1
Zu Versuch H42.1. Diese Befehlsfolge kann den Selbstmord Ihres Programms verursachen.

Versuch H42.1

Ein Programm-Selbstmordprogramm

Geben Sie bitte das Programm zum Versuch H34.1 noch einmal ein (Hauptprogramm im Bild H35.1; Unterprogramm BITS im Bild H33.2). Tasten Sie einige Bytes ein. Wenn die Anzeige programmgemäß erscheint, dann können Sie das Programm mit der Taste RS anhalten.

Tasten Sie jetzt bitte die im Bild H42.1 aufgelistete Befehlsfolge ab der Adresse 1824 ein und prüfen Sie dann noch einmal, ob ihr Programm noch läuft. – Ändern Sie anschließend im Hauptprogramm für die Eingabe von zwei Halbbytes den Operanden bei der Adresse 1812 wie folgt:

Adresse	Byte	
1812	11	(statt F1)

Starten Sie jetzt das Programm wieder bei der Adresse 18ØØ. Was geschieht? – Zunächst geschieht überhaupt nichts, was Sie irgendwie überraschen könnte; in der Anzeige erscheint das vorher zuletzt eingegebene Byte mit seinem Bitmuster.

Tasten Sie ein Halbbyte ein! Auch das funktioniert programmgemäß. Aber jetzt kommt's: Wenn Sie das zweite Halbbyte eintasten, dann verlöscht die Anzeige und es geht nichts mehr.

Schauen Sie nach, was da los ist und kontrollieren Sie Ihr Programm ab der Adresse 18ØØ (RS, ADDR, 1, 8, 0, 0). Ihr Programm ist verschwunden; statt seiner finden Sie lauter Bytes ØØ.

Der Grund für diesen Programm-Selbstmord ist offenbar das jetzt nicht mehr korrekte Byte bei der Adresse 1812. Dabei braucht uns hier nicht zu interessieren, wie diese Zerstörung des Programms durch ein einziges falsches Byte zustande kommt. Wir stellen einfach fest, daß die Bytes aus dem Bild H42.1 vielleicht als Rest eines irgendwann vorher eingegebenen Programms noch im Speicher herumgeistern.

Zugegeben: Unser Beispiel ist natürlich konstruiert. Solches Pech kann Ihnen aber auch ganz unbeabsichtigt widerfahren.

Der in einem System benötigte Festspeicher muß immer so groß sein, wie es ein spezielles Programm erfordert. Für den RAM-Speicher gelten andere Überlegungen. Sein Umfang hängt davon ab, wie viele veränderliche Daten beim Ablauf des System-Programms anfallen. Bei einem Uhren-Programm kann man auf den RAM-Speicher unter Umständen ganz verzichten. Bei einem Aufzug-Programm braucht man nur einen recht kleinen Speicher. Setzt man das System dagegen z. B. zur Buchführung über ein umfangreiches Lager irgendwelcher Einzelteile ein, dann kann der RAM-Speicher gar nicht groß genug sein.

Das Betriebsprogramm Ihres Systems kommt im Prinzip mit einem sehr kleinen RAM-Speicher aus. In einem solchen System zur Entwicklung von Programmen fallen aber noch viele andere, veränderliche Daten an: Eben die Programm-Daten der zu entwickelnden Programme.

Der RAM-Speicher Ihres Systems ist recht großzügig ausgelegt: Er faßt mehr als vierzigmal das Uhrenprogramm (Bild H9.1).

Das System der Speicher-Adressen

Weil es sich bei dem in Ihrem System verwendeten Mikroprozessor um eine Acht-Bit-Type handelt, der stets acht Bit parallel verarbeitet, muß der Speicher des Systems eine Menge Acht-Bit-Worte aufnehmen. Bei einem Sechzehn-Bit-Prozessor müßte der Speicher entsprechend eine Menge von Sechzehn-Bit-Worten aufnehmen. Es handelt sich um **wortorganisierte Speicher**.

Im Prinzip muß der in einem Mikroprozessor-System notwendige Speicher also eine Art zusätzlicher Register enthalten, die ähnlich strukturiert sind wie die Register A bis L in der CPU. Bei Speichern spricht man jedoch nicht von Registern, sondern von **Speicherzellen** oder von **Speicherplätzen**.

> Eine Speicherzelle in einem wortorganisierten Speicher ist eine Gruppe von Speicherelementen, die ein verarbeitbares Wort aufnimmt.

Ein Speicherelement kann man sich (wie auch bei den CPU-Registern) einfach als irgendwie geartetes Flipflop vorstellen.

Im Bild H43.1 haben wir einen Teil eines für Acht-Bit-Worte organisierten Speichers dargestellt. Wie im Bild H31.1 sind die acht Speicherelemente der Speicherzellen, die z. B. Flipflops sein könnten, als kleine Rechtecke dargestellt. In diese Speicherelemente sind Binärwerte eingetragen. Die neben den Speicherzellen angeschriebenen Sedezimalziffern lassen erkennen, daß es sich offenbar um Programm-Daten handelt. Nicht zu erkennen ist, ob es sich um einen Festspeicher oder um einen Schreib-Lese-Speicher handelt.

Wenn man im RAM-Speicher sowohl Programm-Daten als auch während des Programmablaufs anfallende, veränderliche Daten ablegt, dann ist eine sorgfältige Verwaltung der Speicherzellen unumgänglich. Für diesen Zweck beschränkt man sich auf eine Darstellung entsprechend dem Teilbild b. Bei veränderlichen Daten trägt man in die Darstellung die Bezeichnung dieser Daten (z. B. Stunden) ein.

Für einen Mikroprozessor ist es ganz wesentlich zu wissen, in welcher Speicherzelle bestimmte Daten abgelegt werden müssen bzw. in welcher Speicherzelle gesuchte Daten zu finden sind. Das wird einfach durch fortlaufende Numerierung der Speicherzellen möglich, die sedezimal vorgenommen wird: Sie beginnt bei ØØØØ**H** und kann bei unserem Mikroprozessor bis FFFF**H** geführt werden. Er kann also einschließlich der Speicherzelle mit der Nr. ØØØØ insgesamt 65536 Speicherzellen handhaben. – Sehen Sie sich die Aufgabe H36.1 und deren Lösung auf der Seite Ü8 an!

Wir haben bereits angedeutet, daß auch das Betriebsprogramm bei seinem Ablauf einen (verhältnismäßig kleinen) Bereich des RAM-Speichers für sich beansprucht. In diesem Bereich werden z. B. beim Abarbeiten eines Programms in Einzelschritten nach jedem Programmschritt die jeweils aktuellen Inhalte der CPU abgelegt, so daß sie

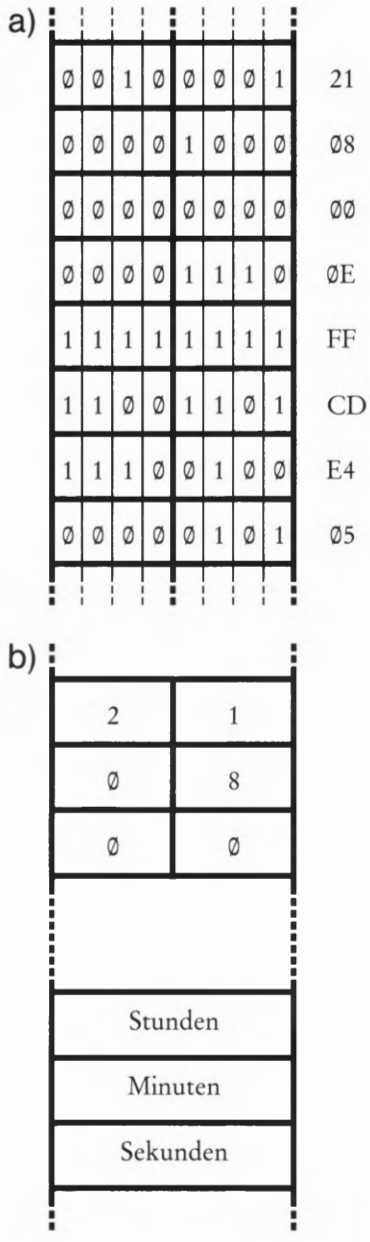


Bild H43.1
a) Im wortorganisierten Speicher enthält jede Speicherzelle soviele Speicherelemente, wie das zu speichernde Wort Bits enthält.
b) In der Praxis begnügt man sich mit der Darstellung von Speicherzellen (vgl. auch das Bild H11.1).

nach Betätigen der REG-Taste angezeigt werden können. – Ein solcher Speicherbereich, der dem System beim Abarbeiten eines Programms zum mehr oder weniger kurzzeitigen Aufbewahren solcher Daten dient, wird im Fachjargon mit dem englischen Ausdruck **Scratchpad** (sprich: sskretschpät) bezeichnet. Wörtlich bedeutet das einen Kritzelblock, also einen Notizblock für wichtige, aber nicht langlebige Daten.

Man kann die einzelnen Speicherzellen des gesamten Speichers als Häuser betrachten, deren jedes von acht Bit „bewohnt" werden kann. Manche dieser Häuser sind für Dauer-Bewohner eingerichtet (Festspeicher-Zellen, ROM); bei anderen wechseln die Bewohner häufiger oder es sind gar Asyle für Tippelbrüder (Scratchpad). Diese Häuser für wechselnde Bewohner gehören zur RAM-Gruppe. Jedenfalls aber sind alle Häuser zur besseren Kontrolle durch die hohe Obrigkeit numeriert: Jedes Haus hat seine Adresse.

Diesem Vergleich verdanken die Nummern der Speicherzellen ihre Bezeichnung: Man nennt diese Nummern Adressen.

> Eine Adresse ist ein bestimmtes Wort zur Kennzeichnung einer Speicherzelle.

Die in Ihrem System sedezimal vierstellig dargestellten Adressen werden von der CPU als Sechzehn-Bit-Worte gehandhabt; sie bestehen also aus zwei Byte. Nun wird auch die Bedeutung der sechzehn Bit „breiten" Registerpaare deutlich: Sie eignen sich besonders zum Arbeiten mit Speicher-Adressen. Ihr Inhalt kann auf eine bestimmte Speicherzelle zeigen. Zeigen heißt auf englisch *point*, und daher kommt die Bezeichnung so arbeitender Sechzehn-Bit-Register: Sie werden als **Pointer** (sprich: peunter) bezeichnet.

Speziell das sechzehn Bit breite Register **Programmzähler** (vgl. Seite T 2) ist ein solches Pointer-Register. Seine abgekürzte Bezeichnung **PC** kommt vom englischen **P***rogram* **C***ounter* (sprich: program-kaunter) und bedeutet wörtlich Programmzähler. Sein Inhalt zeigt immer auf eine bestimmte Speicherzelle, und das ist seine einzige Funktion innerhalb der CPU.

Die CPU interpretiert die Inhalte der Speicherzelle, auf die das Pointer-Register Programmzähler (PC) zeigt, immer als Programm-Daten, also als Befehle oder als Teile von Befehlen. (Daher der Name Programmzähler.) Der Inhalt dieses Sechzehn-Bit-Registers wird beim Abarbeiten eines Programms über die innere Struktur der CPU (das sogenannte Mikroprogramm) schrittweise um eins bei Ein-Byte-Befehlen, um zwei bei Zwei-Byte-Befehlen usw. erhöht.

Natürlich muß die CPU nicht nur Speicherzellen erreichen („adressieren") können, in denen Programm-Daten stehen. Sie muß auch an solche Speicherzellen heran können, in denen zu verarbeitende Daten oder Bitmuster abgelegt sind. Diese Speicherzellen werden z. B. über das als **Memory-Pointer** fungierende, sechzehn Bit breite HL-Registerpaar adressiert. Memory (sprich: memmorie) ist die englische Bezeichnung für Speicher. Wie diese Adressierung funktioniert, werden wir im anschließenden Software Abschnitt erläutern.

Das Cassetten-Interface

Unter einem Interface versteht man eine elektronische Schaltung, die zwei Geräte oder Bausteine aneinander anpaßt. Im hier betrachteten Zusammenhang ist eins dieser Geräte meistens das Mikroprozessor-System. Das andere dieser Geräte kann z.B. ein Bildschirm-Gerät (Video-Monitor) sein; in diesem Falle spricht man von einem Video-Interface. Oder es ist ein irgendwie gearteter Drucker; dann handelt es sich um ein Drucker-Interface.

Wir wollen Ihnen hier die Handhabung einer Interface-Schaltung vorstellen, die in Ihr System bereits integriert ist. Es handelt sich um eine Einrichtung, mit der Sie einmal in das System eingetastete Programme dauerhaft speichern können, ohne daß diese Programme in einem Festspeicher (PROM oder EPROM, vgl. Seite H40) abgelegt werden müssen.

Programme, die über das Tastenfeld in Ihr System eingegeben werden, sind im Schreib-Lese-Speicher (RAM, Seite H41) abgelegt. Sie bleiben dort gespeichert, solange die Stromversorgung Ihres Systems in Betrieb ist. Nach dem Abschalten der Stromversorgung verschwinden die Programme wieder aus dem Schreib-Lese-Speicher und müssen zu neuerlicher Verwendung nach dem Einschalten der Stromversorgung jedesmal neu eingetastet werden. Das ist besonders bei längeren Programmen eine äußerst lästige (und fehlerträchtige) Prozedur.

Für komfortable Mikroprozessor-Systeme stehen verschiedene Datenspeicher zur Verfügung, in denen Daten aus dem Schreib-Lese-Speicher dauerhaft, aber leicht veränderbar aufbewahrt werden können. Eins der ältesten Speichermedien stellen Magnetbänder dar, für die zum Teil recht aufwendige und schnell arbeitende spezielle Laufwerke konstruiert wurden. Eine sehr elegante, modernere Variante für die magnetische Speicherung von Daten bilden Floppy-Disks oder Plattenspeicher, die in ihrer Arbeitsweise eine entfernte Ähnlichkeit mit Schallplatten haben, auf denen die Informationen allerdings mechanisch gespeichert werden.

Solche Speichermedien sind jedoch für ein Mikroprozessor-System der Größenordnung unseres Mikro-Professors viel zu aufwendig. Eine sehr effektive Methode zur Speicherung von Daten aus Systemen dieser Größenordnung stellt die Benutzung eines normalen Audio-Cassettenrecorders dar, wie er heute in fast jedem Hause verfügbar ist und besonders von Kindern in sehr einfacher Ausführung gern verwendet wird. Zur Speicherung von Daten braucht also durchaus kein hochwertiges Cassetten-Deck genommen zu werden.

Die Daten aus dem Schreib-Lese-Speicher des Mikroprozessor-Systems werden auf der Audio-Cassette in Form akustischer Signale gespeichert. Die Umwandlung der Daten in diese akustischen Signale übernimmt das Cassetten-Interface, das zunächst an der Eingangsseite mit dem Mikroprozessor-System verbunden ist; der Ausgang der Interface-Schaltung besteht aus einer Buchse, an der niederfrequente Wechselspannungs-Signale geliefert werden, die direkt dem Mikrophon-Anschluß des Cassetten-Interface zugeführt werden können.

Das Cassetten-Interface hat aber noch eine zweite Aufgabe. Die auf der Cassette akustisch gespeicherten Daten sollen ja auch irgendwann in den Schreib-Lese-Speicher des Systems eingelesen werden können. Fast alle Cassetten-Recorder haben eine Buchse zum Anschluß eines Kopfhörers oder eines getrennten Lautsprechers. An dieser Buchse steht eine niederfrequente Wechselspannung zur Verfügung, wenn der Recorder auf Wiedergabe geschaltet ist. Verbindet man diese Buchse mit der entsprechenden Buchse des Cassetten-Interface, dann übernimmt die Interface-Schaltung die Umwandlung der jetzt als niederfrequente Wechselspannung gelieferten Daten in einer Form, die die Ablage der Daten im Schreib-Lese-Speicher des Systems möglich macht. – Die Interface-Schaltung arbeitet also als Ausgangs- und als Eingangs-Wandler für unser System.

Die Einrichtung zur Speicherung von Daten

Es wurde bereits erwähnt: Zur Speicherung von Daten aus dem Mikro-Professor auf Audio-Cassetten genügt ein Cassettenrecorder der preiswertesten Sorte. Zu beachten ist allerdings, daß der Mikro-Professor kein Produkt der deutschen Industrie ist. Die von seinem eingebauten Cassetten-Interface gelieferten Signale entsprechen deshalb auch nicht den von DIN genormten Signalen, die üblicherweise bei deutschen Recordern an die meist fünfpolige DIN-Buchse des Recorders zur Aufnahme von Sprache oder Musik aus dem Radio geliefert werden müssen. Ebensowenig verarbeitet das Cassetten-Interface die Signale, die von einem anderen Anschluß der fünfpoligen DIN-Buchse bei der Wiedergabe von der Cassette geliefert werden. – Am besten vergessen Sie also diese an Ihrem Recorder angebrachte DIN-Buchse beim Abspeichern und Einlesen von Mikroprozessor-Daten.

Verwenden Sie zum Abspeichern von Daten die Anschlußbuchse für das Mikrophon, die an Ihrem Recorder zur Verfügung stehen sollte. Die Empfindlichkeit dieses Eingangs kann am Recorder meist eingestellt werden. Wenn Ihr Recorder dafür eine Automatik besitzt – um so besser; dann brauchen Sie sich um die Einstellung nicht zu kümmern. Andernfalls müssen Sie die Einstellung der Empfindlichkeit für die Signale vom Mikro-Professor mit Hilfe der Aussteuer-Anzeige entsprechend der Bedienungsanleitung vornehmen.

Zum Einlesen der auf der Cassette gespeicherten Daten in Ihr System benutzen Sie bitte den Anschluß für einen getrennten Lautsprecher oder Köpfhörer an Ihrem Recorder. Die Amplitude der an diesem Anschluß gelieferten, niederfrequenten Signale können Sie über den Lautstärke-Einsteller Ihres Recorders vornehmen. Die richtige Einstellung der Lautstärke ist einfach, weil der Lautsprecher des Mikro-Professors die Signale wiedergibt. Stellen Sie die Lautstärke immer so ein, daß Sie die Signale mit der gleichen Lautstärke hören wie vorher beim Abspeichern der Signale.

Für das Herstellen der Verbindungsleitung zwischen dem Mikro-Professor und dem Recorder müssen wir an Ihr bastlerisches Geschick appelieren. Auf dem Mikro-Professor sind zwei Buchsen für 3,5 mm-Klinkenstecker angebracht, die mit „EAR“ (= Ohr) und „MIC“ (= Mikro, vgl. Seite T1) bezeichnet sind. Die Buchse EAR muß mit

dem Anschluß für den getrennten Lautsprecher an Ihrem Recorder verbunden werden und die Buchse MIC mit dem Anschluß für das Mikrophon. Wir verzichten hier auf eine Beschreibung für das Anfertigen entsprechender Verbindungsleitungen, weil an Cassetten-Recordern sehr unterschiedliche Buchsen für Mikrophon- und Kopfhörer-Anschluß verwendet werden. Im Zweifelsfall ist Ihnen sicher ein Bekannter mit Bastel-Erfahrung behilflich. Jedenfalls sollten Sie jedoch für die Verbindung koaxiale, also abgeschirmte Leitungen verwenden.

Das Abspeichern von Daten auf Cassette

Ehe Sie wirklich Daten auf einer Cassette abspeichern, können Sie sich in einem Versuch ansehen (bzw. anhören), wie dieses Abspeichern funktioniert.

Versuch H47.1

Akustische Daten-Darstellung im Lautsprecher des Systems

Sie brauchen für diesen Versuch noch keinen Cassetten-Recorder. Sie brauchen auch noch keine Daten über das Tastenfeld einzugeben. Wir tun einfach so, als wollten wir das im Versuch H6.1 abgespielte Demonstrations-Programm auf einer Cassette ablegen. Tatsächlich ist das natürlich Unsinn, denn dieses Programm ist in Ihrem System in einem Festspeicher fest abgelegt und braucht nicht noch einmal getrennt gespeichert zu werden.

Schalten Sie Ihr System an und betätigen Sie der Reihe nach folgende Tasten:

Tasten	Anmerkungen	Anzeige
TAPE WR	(Von links gezählt zweite Taste in der dritten Reihe des Tastenfeldes.) Frage nach der **F**ile-Nummer. (Wir erläutern gleich, was das ist):	X.X.X.X. – F
0, 0, 0, 1	Eingabe der **F**ile-Nummer.	0 .0 .0 .1 . – F
+	Frage nach der **S**tart-Adresse des Files:	X.X.X.X. – S
2, 3, 0, 0	Eingabe der Start-Adresse des Files	2 .3 .0 .0 . – S
+	Frage nach der **E**nd-Adresse des Files:	X.X.X.X. – E
2, 4, 1, 1	Eingabe der End-Adresse des Files. (Nach dieser Eingabe müßte der Recorder angeschlossen werden.)	2 .4 .1 .1 . – E
GO	Hören Sie selbst! Am Ende der Übertragung erscheint die End-Adresse und das dort abgelegte Byte in der Anzeige	dunkel 2 4 1 1 4.0.

Nach dem Betätigen der Taste GO hören Sie zunächst vier Sekunden lang einen Synchronisations-Ton mit einer Frequenz von 1KHz. Danach liefert das System für kurze Zeit einen merkwürdig schnarrenden Ton und anschließend zwei Sekunden lang einen Ton mit der

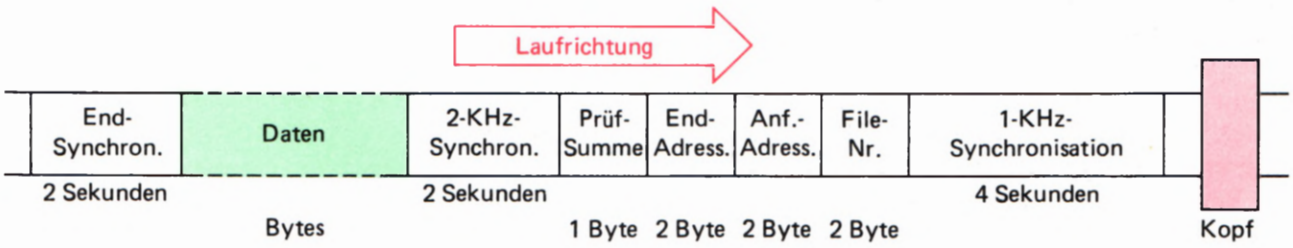


Bild H48.1
Der Mikro-Professor legt beim Abspeichern eines Files außer den Daten noch die File-Nummer, die Start- und die End-Adresse sowie ein Byte als Prüfsumme auf dem Band ab.

Frequenz von 2 KHz. Daran anschließend werden mit schnarrendem Geräusch die Daten ausgegeben und am Ende der Übertragung kommt noch einmal zwei Sekunden lang ein 2-KHz-Ton.

Im Bild H48.1 haben wir das Aufzeichnungs-Schema der Daten auf dem Band dargestellt. Dazu folgende Erläuterung:

Ein File (sprich: feil) ist eine zusammengehörige Menge von Daten. Im hier betrachteten Zusammenhang besteht ein File aus Programm-Daten; ein File kann aber auch aus Daten bestehen, die irgendwelche andere Informationen darstellen, z. B. die Buchstaben und Ziffern eines Textes. Für unseren Mikroprozessor bedeutet ein File immer eine Menge von Bytes mit jeweils acht Bit.

Die Speicherung der Daten erfolgt auf dem Band Byte-seriell, Bit-seriell. Das heißt: Auf dem Band wird ein Byte nach dem anderen gespeichert und innerhalb eines Bytes werden auch die einzelnen Bits nach bestimmten Regeln nacheinander auf das Band gebracht. Damit das Mikroprozessor-System beim späteren Zurücklesen der Daten erkennen kann, wann jeweils ein Byte beginnt und wann es zu Ende ist, werden für ein Byte außer den zugehörigen acht Daten-Bits noch ein Start-Bit mit dem Wert 1 und ein Stop-Bit mit dem Wert Ø übertragen. (Manche Systeme übertragen statt des einen Stop-Ø-Bits auch deren zwei. Für ein Byte müssen dann auf dem Band insgesamt elf Bit abgelegt werden.)

In Ihrem System werden Ø-Bits durch acht Perioden einer 2-KHz- und anschließend zwei Perioden einer 1-KHz-Wechselspannung gekennzeichnet. 1-Bits werden durch vier Perioden einer 2-KHz- und anschließend vier Perioden einer 1-KHz-Wechselspannung gekennzeichnet. Zur Übertragung eines Bits benötigt Ihr System also 6 ms und zur Übertragung eines Bytes einschließlich des Start-Ø- und des Stop-1-Bits insgesamt 60 ms.

Diese Art der akustischen Kennzeichnung von Ø- und 1-Bits ist spezifisch für das Mikro-Professor-System. Andere Systeme arbeiten bei der Aufzeichnung von Daten auf Audio-Cassetten nach anderen Prinzipien der Darstellung von Ø- und 1-Bits, die uns aber hier nicht interessieren sollen.

Aufgabe H48.1

Überlegen Sie bitte, wie lange die Übertragung der Daten des Uhren-Programms aus dem Bild H9.1 einschließlich des im Bild H48.1 dargestellten Vorspanns und des Nachspanns dauert!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü10.

Die Lösung der vorstehenden Aufgabe macht deutlich, daß Sie auf einer normalen Cassette mit einer halben Stunde Spieldauer auf jeder Seite mit Leichtigkeit eine Menge unterschiedlicher Programme hintereinander ablegen können. Zum Wiederfinden bestimmter Programme beim Einlesen in das System bietet der Mikro-Professor eine elegante Möglichkeit: Sie können den einzelnen Programmen vierstellige Kenn-Nummern zuordnen, die dann entsprechend der Darstellung im Bild H48.1 zusammen mit der Anfangs- und Endadresse des Programms als File-Nummer vor den eigentlichen Daten auf dem Band gespeichert werden.

Versuch H49.1

Speichern mehrerer Programme auf Cassette

Tasten Sie bitte noch einmal das Uhren-Programm aus dem Bild H9.1 in Ihr System ein und prüfen Sie das Programm entsprechend dem Versuch H9.1.

Halten Sie das Programm an (RS) und betätigen Sie – ähnlich wie im vorhergehenden Versuch – folgende Tasten:

Tasten	Anmerkungen	Anzeige
TAPE WR	Frage nach der File-Nummer	X.X.X.X. – F
0, 0, 0, 1	Eingabe der File-Nummer.	0 .0 .0 .1 . – F
+	Frage nach der Start-Adresse des Files:	X.X.X.X. – S
1, 8, 0, 0	Eingabe der Start-Adresse des Files:	1 .8 .0 .0 . – S
+	Frage nach der End-Adresse des Files:	X.X.X.X. – E
1, 8, 4, 7	Eingabe der End-Adresse des Files.	1 .8 .4 .7 . – E

Schließen Sie jetzt Ihren Cassetten-Recorder so an das System an, wie es im vorhergehenden Abschnitt beschrieben wurde. – Schalten Sie den Recorder entsprechend der Betriebsanweisung auf Aufnahme. – Wenn Ihr Recorder eine Automatik für die Eingangs-Empfindlichkeit hat, dann schalten Sie sie jetzt ein und starten Sie den Recorder. – Betätigen Sie die Taste

GO	Bei fehlender Automatik für die Eingangs-Empfindlichkeit haben Sie jetzt 4 Sekunden Zeit, die Einstellung der Empfindlichkeit nach der Aussteuer-Anzeige vorzunehmen.	dunkel
	Am Ende der Übertragung erscheint die End-Adresse mit dem letzten Byte des Programms:	1 8 4 7 6.0.

Halten Sie den Recorder an und lassen Sie das Band zunächst weder vor- noch zurücklaufen.

Laden Sie Ihr System mit dem im Bild S25.1 aufgelisteten Programm für den Versuch S25.1 und prüfen Sie, ob das Programm einwandfrei arbeitet! – Wiederholen Sie jetzt die für die Abspeicherung des Uhren-Programms beschriebene Prozedur, tasten Sie aber als File-Nummer ØØØ2 ein, als Start-Adresse des Programms wieder 18ØØ und als End-Adresse 1824!

Laden Sie zum Schluß Ihr System mit den Programmteilen für den Versuch H34.1 (Bilder H35.1 und H33.2) und führen Sie den Versuch zur Kontrolle noch einmal durch! – Dieses Programm wird mit der File-Nummer ØØØ3 gespeichert. Die Start-Adresse ist wieder 18ØØ; geben Sie als End-Adresse 1912 ein! Damit werden zwar die nicht zum Programm gehörenden, zufälligen Inhalte der Speicherzellen zwischen den Adressen 1813 und 18FF mit auf dem Band gespeichert. Das Abspeichern (und entsprechend das spätere Wieder-Einlesen) des Programms dauert relativ lange, aber das braucht Sie hier nicht zu stören.

Bei dieser Gelegenheit können Sie auch gleich noch die Befehlsfolge für den Versuch S53.1 aus dem Bild S53.1 von der Adresse 1F9Ø bis zur Adresse 1F9C unter der File-Nummer ØØØ4 abspeichern.

Daten von der Cassette in den Speicher

Nach dem vorangegangenen Abspeichern Ihrer Programme können Sie Ihr System nun getrost ausschalten; Sie können sich die Programme jederzeit wieder äußerst bequem in Ihr System zurückspielen.

Versuch H50.1

Daten von der Cassette in den Speicher

Von den vorher abgelegten Programmen soll das mit der File-Nummer ØØØ2 (Versuch S25.1) in den Schreib-Lese-Speicher geladen werden. Betätigen Sie dazu folgende Tasten:

Tasten	Anmerkungen	Anzeige	
TAPE RD	(Von links gezählt zweite Taste in der vierten Reihe des Tastenfeldes.)		
	Frage nach der File-Nummer:	X.X.X.X.	– F
0, 0, 0, 2	Eingabe der File-Nummer	0 .0 .0 .2 .	– F
GO	Band zurückspulen, auf Wiedergabe starten. Lautstärke ganz aufdrehen.	. . .	. .
	Die eingelesenen File-Nummern werden jeweils kurz angezeigt. – Anzeige	0 .0 .0 .1	– F
		0 .0 .0 .2	– F
	beim Lesen des richtigen Files:	– – – –	– –
	Am Ende End-Adresse und Byte:	1 8 2 4	D B

Da die Start- und End-Adressen des angewählten Files auf dem Band gespeichert sind, brauchen sie nicht noch einmal eingegeben zu werden. – Auf dem Band ist noch ein Byte mit einer Prüfsumme abgelegt (Bild H48.1). Diese wird beim Einlesen verglichen. Wenn ein Lese-Fehler auftritt, dann meldet das System das mit der **ERR***or*-(= Fehler-) Meldung „-ERR". Machen Sie einen neuen Lese-Versuch!

Hilfsfunktionen des Mikro-Professors

Im Versuch S 69.1 ergab sich die Notwendigkeit, in ein Programm, das Sie bereits in den Speicher des Systems eingetastet hatten, zusätzliche Befehle einzubauen. Eine solche Notwendigkeit versetzt den Programmierer oft in gelinde Panik, wenn ihm nicht die Möglichkeiten eines sehr viel komfortableren Programm-Entwicklungssystems zur Verfügung stehen.

Eigentlich gibt es zur Lösung einer solchen Aufgabe nur zwei Möglichkeiten: Man muß entweder zu einer (immer uneleganten) „Rucksack"-Programmierung greifen, oder man muß sich zähneknirschend damit abfinden, daß man den hinter der notwendigen Einfügung folgenden Programm-Teil ab einer höheren Adresse neu eintasten muß. Bei verhältnismäßig kleinen Programmen mag das noch angehen. Aber wenn in einem Programm mit 200 Befehlen nach dem zehnten Befehl auch nur ein einziges Byte für eine Programm-Ergänzung fehlt, dann ist es schon mehr als ärgerlich, wenn man 90 bereits eingegebene Befehle ab einer um eins höheren Adresse neu eingeben muß, um für dieses eine Byte Platz zu schaffen.

Ähnlich, wenn auch nicht ganz so dramatisch ist es, wenn man in einem Programm aus irgendwelchen Gründen einen oder mehrere Befehle unwirksam machen möchte. Im einfachsten Fall ersetzt man die nicht mehr benötigten Befehls-Bytes einfach durch NOP-Befehle (Seite S 66). Erstrebenswert wäre es natürlich, wenn man die Speicherplätze der nicht mehr benötigten Befehle mit Bytes aus dem anschließenden Programm-Teil belegen könnte. Aber das bedeutet wieder die ärgerliche Neu-Eingabe eines mehr oder weniger langen Teils des Programms.

Das Betriebsprogramm des Mikro-Professors (Seite H 41) enthält drei Routinen, mit denen sich die hier beschriebenen Probleme spielend leicht lösen lassen:

1. INSERT-Routine zum Einfügen eines einzelnen Bytes in eine bestehende Daten-Folge im Speicher.

2. DELETE-Routine zum Herausnehmen eines einzelnen Bytes aus einer bestehenden Daten-Folge im Speicher.

3. MOVE-Routine zum Verschieben eines beliebig definierten Daten-Blocks im Speicher an beliebige Stellen im Schreib-Lese-Speicher.

Wir wollen Ihnen die Verwendung dieser Routinen einzeln vorstellen.

Einfügen eines Bytes in eine Daten-Folge

Das hier beschriebene Einfügen eines Bytes wird im Englischen mit **INS***ert* (sprich: inssört) bezeichnet. Für die INSERT-Routine ist die mit INS bezeichnete Taste (von links gezählt die dritte Taste in der oberen Reihe des Tastenfeldes) vorgesehen, die Sie in ganz anderem Zusammenhang (Versuch S 64.1) bereits einmal verwendet haben.

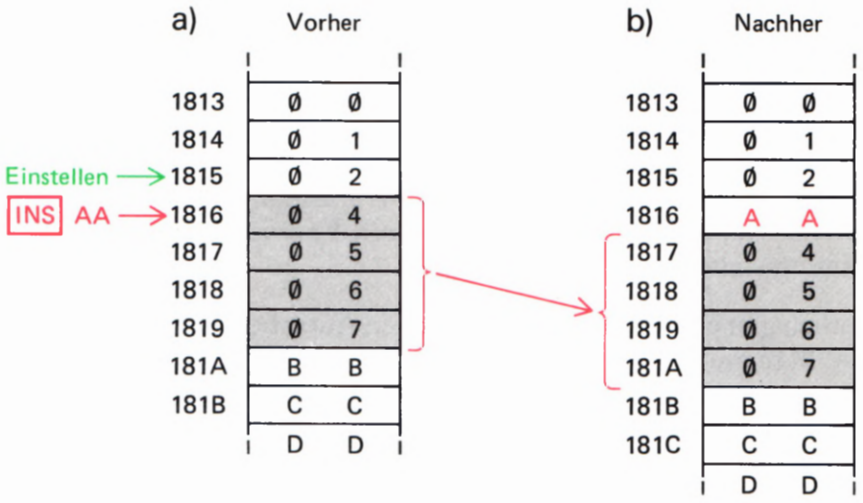


Bild H52.1
Mit Hilfe der INS-Taste kann ein Byte in eine Daten-Folge im Speicher eingefügt werden. Es muß eine Adresse angewählt werden, die um eins unter der Adresse liegt, bei der eingefügt werden soll.

Im Bild H52.1 ist dargestellt, wie die INSERT-Routine funktioniert. Mit Hilfe der Taste ADDR wird die Adresse in die Anzeige gebracht, die gerade um eins unter der Adresse der Speicherzelle liegt, in der das zusätzliche Byte abgelegt werden soll. Anschließend wird die Taste INS betätigt, und das bewirkt bereits, daß die Inhalte aller folgenden Speicherzellen (bis zur Speicherzelle mit der Adresse 1DFF) um eine Adresse im Speicher hinaufgeschoben werden. In die Speicherzelle, in die das zusätzliche Byte eingetragen werden soll, wird das Byte ØØ eingetragen.

Weil das System gleichzeitig mit dem Betätigen der Taste INS in den Daten-Eingabe-Modus bei der Adresse umgeschaltet wird, an der das zusätzliche Byte eingetragen werden soll – die Dezimalpunkte erscheinen an den rechten beiden Anzeige-Stellen –, kann nun sofort das zusätzliche Byte eingetragen werden.

Der hier beschriebene Vorgang klingt vielleicht zunächst reichlich kompliziert. Bei der Eingabe und vor allem beim Austesten von Programmen erweist sich die INSERT-Routine jedoch als große Hilfe. Sehen Sie sich den Vorgang in einem Versuch an; Sie werden feststellen, daß die Sache in Wirklichkeit sehr einfach ist.

Versuch H52.1

Einfügen eines Bytes

Tasten Sie ab der Adresse 1813 die im Bild H52.1a dargestellten Bytes – beginnend mit dem Byte ØØ und endend mit dem Byte DD – in den Schreib-Lese-Speicher ein. Wir wollen annehmen, alle mit dem Halb-Byte Ø beginnenden Bytes seien irgendwelche Daten eines beliebigen Programms; das Byte Ø7 sei das letzte dieser Programm-Daten. – Die Bytes ab der Adresse 181A (BB, CC, DD) sollen nicht mehr zum Programm gehören. Sie werden nur eingegeben, damit in diesen Speicherzellen definierte Bytes stehen.

Die Systematik der eingegebenen „Programm-Daten" zeigt, daß offenbar nach dem Byte Ø2 ein Byte fehlt, denn es folgt gleich darauf das Byte Ø4. Zwischen diesen beiden Bytes soll das fehlende Byte eingetragen werden. Wir wählen dazu nicht das Byte Ø3, sondern der deutlichen Markierung wegen das Byte AA.

Betätigen Sie nun folgende Tasten:

Tasten	Anzeige	Bemerkungen
ADDR, 1, 8, 1, 5	1.8.1.5. 0 2	Einstellen der Adresse, die um eins kleiner ist als die Adresse der Speicherzelle, bei der ein Byte eingefügt werden soll.
INS	1 8 1 6 0. 0.	Daten-Eingabe-Modus bei der Adresse, bei der ein Byte eingefügt werden soll. Dort ist das Byte ØØ eingetragen.
A, A	1 8 1 6 A.A	Das Byte AA ist eingefügt.

Kontrollieren Sie jetzt den Speicher-Inhalt ab der Adresse 1813 bis zur Adresse 181C! Sie finden die im Bild S 52.1b dargestellte Anordnung: Das Byte AA ist zwischen die Bytes Ø2 und Ø4 eingefügt worden; die Anordnung aller übrigen Bytes ist genau die gleiche geblieben. Sie stehen allerdings sämtlich um genau eine Adresse im Speicher nach oben geschoben.

Merkwürdig erscheint beim Einfügen eines Bytes, daß vor der Betätigung der Taste INS nicht die Adresse angewählt wird, bei der die Einfügung vorgenommen werden soll, sondern eine Adresse, die gerade um eins kleiner ist. Das hat seinen Grund: Wenn Sie außer dem Byte AA auch noch gleich anschließend z. B. das Byte 99 einfügen wollen, dann brauchen Sie gleich nach der Eingabe von AA nur noch einmal die Taste INS zu betätigen und sofort danach 99 einzutasten. Überzeugen Sie sich davon, indem Sie den Versuch von Anfang an wiederholen und im Anschluß an die Eingabe A, A die Tasten INS, 9, 9 betätigen! – Sehen Sie sich danach wieder den Speicher-Inhalt ab der Adresse 1813 bis zur Adresse 181C an!

Aufgabe H53.1

Beim Versuch S 69.1 mußte ein „Rucksack" programmiert werden, weil im Programm im Bild S 64.1 kein hinreichender Platz vorgesehen war, um das HL-Registerpaar auf die Speicherzelle zeigen zu lassen, in der das einzuODERnde Bitmuster abgelegt wird. Zum Programmieren des HL-Registerpaares als Zeiger werden drei Speicherzellen benötigt; der OR (HL)-Befehl braucht eine weitere Speicherzelle. Das sind insgesamt vier Speicherzellen. Das Programm im Bild S 64.1 weist aber nur zwei (mit farbig eingetragenen NOP-Befehlen markierte) verfügbare Speicherzellen auf.

Im Bild H 54.1 haben wir das Programm aus dem Bild S 64.1 ab dem Befehl Nr. 11 noch einmal wiederholt, diesmal aber mit den in das Programm eingebauten Befehlen LD HL,19ØØ zum Setzen des Zeigers und OR (HL) für die ODER-Verknüpfung. Dabei haben wir die Bytes grün markiert, für die im ursprünglichen Programm noch Platz war. Rot markiert sind die Bytes, deretwegen wir im Versuch S 69.1 den Rucksack programmieren mußten.

Das Problem ist ohne „Rucksack" lösbar, wenn ab dem Befehl Nr. 12, LD C,A, im ursprünglichen Programm alle Bytes um zwei Adressen

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
11		LD	HL,19ØØ	181Ø	21	ØØ	19	Zeiger auf 19ØØ
12		OR	(HL)	1813	B6			Bits einODERn
13		LD	C,A	1814	4F			→ C
14		CALL	DISP	1815	CD	**1A**	18	Tasten – Muster anzeigen
15		JR	START	1818	18	**E6**		nächstes Muster
16	DISP			181A	Ø6	Ø6		Bitmuster anzeigen
17	usw. entsprechend der Liste im Bild S64.2							
	Änderungen bei			1822		37 wird **39**		
				1825		3C wird **3E**		

Bild H54.1
Zu Aufgabe H53.1: In das Programm im Bild S64.1 sollen anstelle der beiden NOP-Befehle die Befehle LD HL,19Ø und OR (HL) eingefügt werden. Für den Ablauf des Programms müssen zusätzlich 4 Bytes geändert werden (im Druck hervorgehoben).

im Speicher nach oben geschoben werden. – Anders ausgedrückt: Die im Bild H54.1 grün eingetragenen Bytes 21 und ØØ können anstelle der NOP-Bytes ØØ eingesetzt werden; die rot eingetragenen Bytes müssen hinter der Speicherzelle mit der Adresse 1811 eingefügt werden.

Geben Sie bitte die Tastenfolge an, mit der nach der Eingabe der Bytes 21 und ØØ bei den Adressen 181Ø und 1811 ab der Adresse 1812 die Bytes 19 und B6 in das Programm eingefügt werden können.

Nach dem Einfügen der beiden Bytes können Sie sich in einem Versuch von der Richtigkeit Ihrer Lösung überzeugen, wenn Sie bei der Adresse 19ØØ z. B. das Byte Ø2 ablegen. Beachten Sie aber bitte, daß das Programm nur dann lauffähig ist, wenn die im Bild H54.1 bei den Adressen 1816, 1819, 1822 und 1825 durch stärkeren Druck hervorgehobenen Bytes programmiert sind.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü11.

Entfernen eines Bytes aus einer Daten-Folge

Das Entfernen eines Bytes aus einer Daten-Folge wird im Englischen als **DEL***ete* (sprich: diliit) bezeichnet. Die Taste, die für diesen Vorgang zuständig ist, hat die Bezeichnung DEL; Sie finden Sie an der (von links gezählt) dritten Stelle in der zweiten Reihe des Tastenfeldes.

Technisch ist die Verwendung der DEL-Taste denkbar einfach. Das Bild H55.1 macht den Vorgang deutlich. Mit Hilfe der Taste ADDR wird die Adresse der Speicherzelle eingestellt, deren Inhalt aus dem Speicher entfernt werden soll. Gleich danach wird die Taste DEL betätigt (Teilbild a); in der Anzeige erscheint bei der Adresse des entfernten Bytes das Byte, das ursprünglich bei der nächstfolgenden Adresse stand. Damit ist der Vorgang bereits abgeschlossen.

Auch diesen Vorgang können Sie sich in einem Versuch ansehen:

Versuch H54.1

Das Entfernen eines Bytes

Tasten Sie ab der Adresse 1813 die in das Bild H55.1 eingetragenen Bytes ein, beginnend mit dem Byte ØØ bis zum Byte CC bei der

H54.1

```
                00100 ;
1800            00110           ORG     1800H
1800 3EFE       00120 QAL       LD      A,QAJ
1802 D302       00130           OUT     (QAC),A
1804 DB00       00140           IN      A,(QAA)
1806 2F         00150           CPL
1807 4F         00160           LD      C,A
1808 3EDF       00170           LD      A,QAI
180A D302       00180           OUT     (QAC),A
180C DB00       00190           IN      A,(QAA)
180E 2F         00200           CPL
180F B1         00210           OR      C
1810 210019     00220           LD      HL,QAR
1813 B6         00230           OR      (HL)
1814 4F         00240           LD      C,A
1815 CD1A18     00250           CALL    QAM
1818 18E6       00260           JR      QAL
181A 0606       00270 QAM       LD      B,QAD
181C 3E01       00280           LD      A,QAB
181E CB09       00290 QAN       RRC     C
1820 F5         00300           PUSH    AF
1821 D43918     00310           CALL    NC,QAP
1824 DC3E18     00320           CALL    C,QAQ
1827 F1         00330           POP     AF
1828 F6C0       00340           OR      QAH
182A D302       00350           OUT     (QAC),A
182C E63F       00360           AND     QAF
182E CB07       00370           RLC     A
1830 C5         00380           PUSH    BC
1831 06FF       00390           LD      B,QAK
1833 10FE       00400 QAO       DJNZ    QAO
1835 C1         00410           POP     BC
1836 10E6       00420           DJNZ    QAN
1838 C9         00430           RET
1839 3EBD       00440 QAP       LD      A,QAG
183B D301       00450           OUT     (QAB),A
183D C9         00460           RET
183E 3E30       00470 QAQ       LD      A,QAE
1840 D301       00480           OUT     (QAB),A
1842 C9         00490           RET
0000            01110 QAA       EQU     0000H
0001            01120 QAB       EQU     0001H
0002            01130 QAC       EQU     0002H
0006            01140 QAD       EQU     0006H
0030            01150 QAE       EQU     0030H
003F            01160 QAF       EQU     003FH
00BD            01170 QAG       EQU     00BDH
00C0            01180 QAH       EQU     00C0H
00DF            01190 QAI       EQU     00DFH
00FE            01200 QAJ       EQU     00FEH
00FF            01210 QAK       EQU     00FFH
1900            01220 QAR       EQU     1900H
402D            01230 QAS       EQU     402DH
402D            01240           END     QAS
00000 TOTAL ERRORS
24028  TEXT AREA BYTES LEFT
```

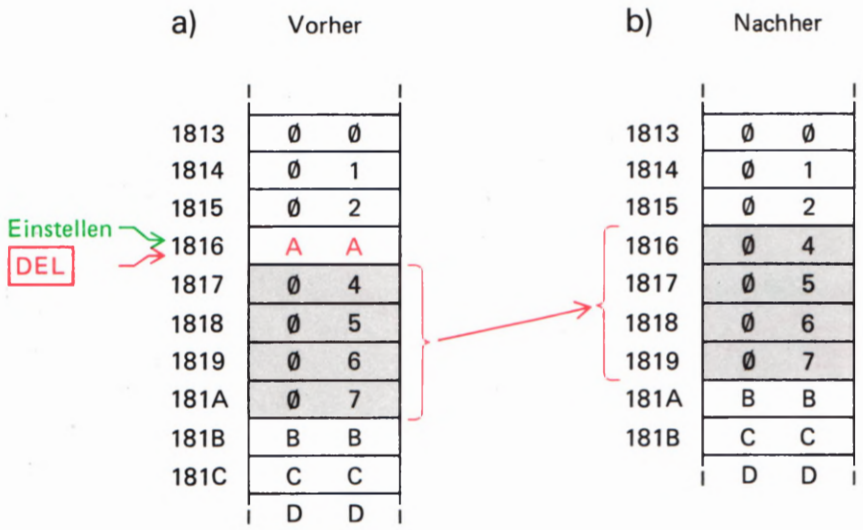


Bild H55.1
Nach dem Anwählen einer Adresse kann mit der DEL-Taste der Inhalt der adressierten Speicherzelle aus dem Speicher entfernt werden.

Adresse 181C. Bei der Adresse 1816 ist das Byte AA eingetragen, das im Versuch H52.1 in die Byte-Folge eingefügt wurde. Dieses Byte soll nun mit der DELETE-Routine wieder entfernt werden.

Betätigen Sie folgende Tasten:

Tasten	Anzeige	Bemerkungen
ADDR, 1, 8, 1, 6	1.8.1.6. AA	Einstellen der Adresse des Bytes, das aus der Byte-Folge entfernt werden soll.
DEL	1 8 1 6 0.4.	Daten-Eingabe-Modus bei der Adresse, bei der ein Byte entfernt worden ist.

Sie erkennen im Vergleich mit dem Bild H55.1a, daß jetzt das Byte Ø4, das vorher bei der Adresse 1817 stand, in der Speicherzelle steht, in der das jetzt entfernte Byte AA stand.

Kontrollieren Sie den Speicher-Inhalt ab der Adresse 1813 bis zur Adresse 181C und vergleichen Sie ihn mit der Darstellung im Teilbild b! – Die an sich nicht interessierenden Bytes BB und CC deuten mit ihren Adressen an, daß der gesamte Speicherinhalt oberhalb der Adresse 1817 um eine Speicherzelle nach unten transportiert worden ist. Das Betriebssystem nimmt diesen Transport bis zur Speicherzelle mit der Adresse 1DFF vor. In diese Speicherzelle wurde das Byte ØØ eingetragen.

Sie können sich im Versuch davon überzeugen, daß durch weitere Betätigungen der Taste DEL der Reihe nach jeweils die Bytes aus dem Speicher entfernt werden, die bei der anfangs angewählten Adresse stehen.

Verschieben eines Daten-Blocks

Der Vorgang des Verschiebens wird im Englischen mit **MOVE** (sprich: muuw) bezeichnet. Eine entsprechende Taste finden Sie im Tastenfeld an der (von links gezählt) zweiten Stelle der oberen Reihe.

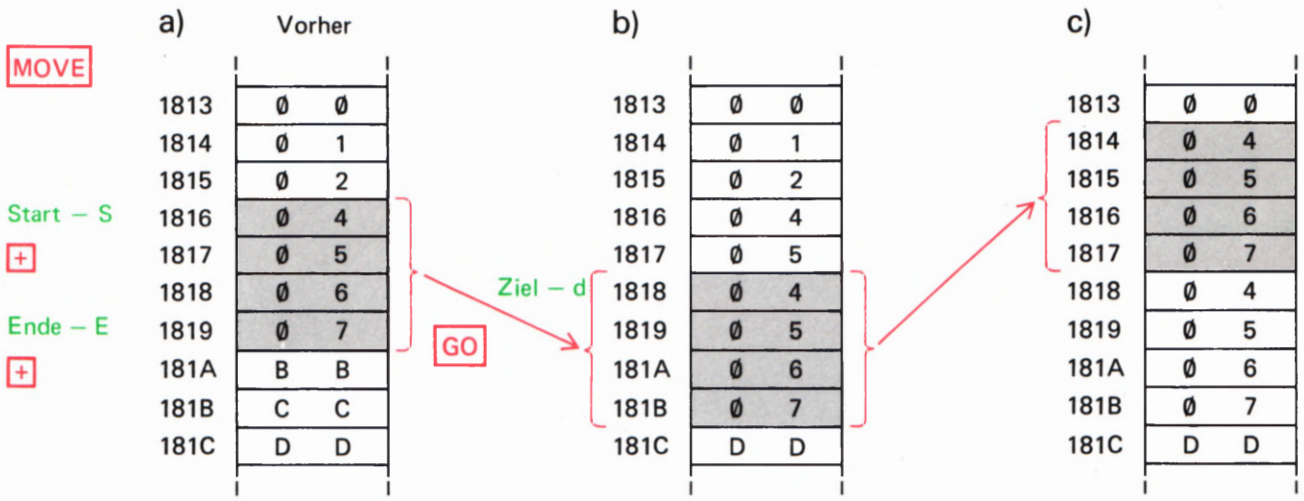


Bild H56.1
Nach Betätigung der MOVE-Taste läßt sich bei eingegebener Start-, End- und Ziel-Adresse durch Betätigung der Taste GO ein Daten-Block beliebig im Speicher verschieben.

Versuch H56.1

Verschieben eines Daten-Blocks

Der Versuch geht von dem Speicher-Inhalt aus, der sich aus dem vorhergehenden Versuch ergeben hat (Bild H56.1a). – Betätigen Sie bitte die folgenden Tasten:

Taste	Anzeige		Bemerkungen
MOVE	X.X.X.X.	– S	Das System erwartet die Eingabe der Anfangs- (Start-) Adresse des Daten-Blocks.
1, 8, 1, 6	1 .8 .1 .6 .	– S	Anfangs-Adresse
+	X.X.X.X.	– E	Das System erwartet die End-Adresse des Daten-Blocks
1, 8, 1, 9	1 .8 .1 .9 .	– E	End-Adresse
+	X.X.X.X.	– D	Das System erwartet die Adresse, bei der das erste Byte des Daten-Blocks stehen soll.
1, 8, 1, 8	1 .8 .1 .8 .	– D	Ziel-Adresse (Destination)
GO	1 8 1 8	0. 4.	Erstes Byte des verschobenen Blocks

Kontrollieren Sie den Speicher-Inhalt zwischen den Adressen 1813 und 181C und vergleichen Sie ihn mit der Darstellung im Teilbild b! Der markierte Daten-Block ist in den neuen Speicherbereich kopiert worden und hat dort die alten Daten überschrieben. Im ursprünglichen Bereich sind die Daten – soweit sie nicht überschrieben wurden – erhalten geblieben.

Aufgabe H56.1

In den Bildern H56.1b und c haben wir die Verschiebung des Daten-Blocks von der Adresse 1818 bis zur Adresse 181B in den Bereich zwischen den Adressen 1814 und 1817 angedeutet.

Bitte geben Sie die Tasten-Folge an, mit der diese Block-Verschiebung vorgenommen werden kann. – Machen Sie einen Versuch!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü11.

Ausgänge und Eingänge eines Mikroprozessor-Systems

Ein Mikroprozessor-Minimalsystem besteht aus einer CPU, einem Speicher und einigen, wenigen und einfachen anderen Bauelementen. Für ein solches System gibt es zwei grundsätzliche Anwendungen:

Entweder arbeitet das System als eine Art Rechner, dessen sich ein Mensch bedient und mit dem er kommunizieren möchte. In diese Kategorie gehört z. B. das im Versuch H9.1 vorgestellte Uhren-Programm.

Oder das System kommuniziert mit einer Maschine. Es übernimmt dann fast beliebig komplizierte Steuer- oder Regelfunktionen für diese Maschine und reagiert auch in außergewöhnlichen Fällen mit fast (!) menschlicher Intelligenz. Ein Beispiel für diese Anwendungsart ist die komfortable Steuerung von Aufzügen oder die Bedienung eines Walzwerks. (Dazu gehört dann allerdings etwas mehr als ein Minimal-System.)

Beide Anwendungsarten machen deutlich, daß ein Mikroprozessor-System niemals um seiner selbst willen existiert. Es hat grundsätzlich eine dienende Funktion, und das bedeutet: Es muß Informationen in Form von Daten aufnehmen können und anschließend, nach Verarbeitung der aufgenommenen Daten, seinerseits Informationen in Form von Daten an einen Empfänger (Mensch oder Maschine) abgeben können.

Der folgende Abschnitt befaßt sich mit den Schaltungsprinzipien, die eine Kommunikation des Mikroprozessors mit seiner **Peripherie** (= Umgebung) ermöglichen.

Serielle und parallele Ausgänge und Eingänge

Ein Acht-Bit-Mikroprozessor verarbeitet Daten in Form von Acht-Bit-Gruppen. Für den Transport dieser Daten bieten sich zwei Möglichkeiten an, die im Bild H57.1 dargestellt sind: Die Daten können **Bit-parallel, Byte-seriell** (Teilbild a) oder **Bit-seriell, Byte-seriell** (Teilbild b) transportiert werden. Das bedeutet, daß jedenfalls immer ein Byte nach dem anderen transportiert wird. Unterschiedlich werden lediglich die einzelnen Bits eines Bytes behandelt.

Das Bild macht deutlich, daß beim Bit-parallelen Transport für jedes Bit eine besondere Leitung zur Verfügung stehen muß, also insgesamt acht Datenleitungen. Damit ergibt sich zwar ein verhältnismäßig großer Leitungsaufwand, (der sich z. B. bei 16-Bit-Mikroprozessoren noch einmal verdoppelt), aber auch ein sehr schneller Datentransport, da immer ein komplettes Byte gleichzeitig auf die Reise geschickt werden kann. Diese Art des Datentransports wird auf dem Datenbus Ihres

Bild H57.1
a) Für den Bit-parallelen, Byte-seriellen Datentransport werden acht Leitungen benötigt.
b) Beim Bit-seriellen, Byte-seriellen Datentransport genügt eine Leitung. In beiden Fällen kommt eine Leitung für das Bezugspotential hinzu.

Systems zum Transport der Daten zwischen CPU und Speicher verwendet.

Der im Teilbild b dargestellte, Bit-serielle Datentransport ist vom Prinzip her wesentlich langsamer, da jedes einzelne Bit eines Bytes getrennt auf die Reise geschickt werden muß. Trotzdem hat die Bit-serielle Datenübertragung große Bedeutung, wenn der Datentransport über größere Entfernungen stattfinden soll. In solchen Fällen wäre der Aufwand acht paralleler Leitungen und einer zusätzlichen Leitung für das Bezugspotential einfach nicht mehr wirtschaftlich.

Wir werden uns hier nur mit der Bit-parallelen Datenübertragung beschäftigen.

Parallele Aus- und Eingänge mit Speicheradressen

Für den Bit-parallelen Datentransport steht innerhalb des Mikroprozessor-Systems der Datenbus zur Verfügung. Für den Datentransport über nicht zu große Entfernungen bietet es sich an, den Datenbus einfach zu verlängern und ihn auch für den Transport von Daten zwischen dem Mikroprozessor-System und seiner Umgebung zu benutzen.

Dabei ergibt sich allerdings ein Problem. Sehen Sie sich bitte noch einmal die Bilder S 45.1 und S 51.1 an, in denen schematisch das Zusammenwirken der CPU mit dem Speicher des Systems dargestellt ist. Sie erkennen, daß die CPU jeweils acht Daten-Bits mit dem Speicher austauscht. Dieser Daten-Austausch wird über den Datenbus vorgenommen. Außerdem muß die CPU dem Speicher jeweils mitteilen, mit welcher Speicherzelle ein Daten-Austausch vorgenommen werden soll. Diese Mitteilung erfolgt in Form einer 16 Bit breiten Adresse, die über einen Adreßbus übermittelt wird.

Beim Abarbeiten eines Programms findet ein ununterbrochener Daten-Austausch zwischen CPU und Speicher statt, denn die CPU muß in schneller Folge die einzelnen Befehls-Bytes aus dem Speicher abholen. Für das Abholen eines Bytes aus dem Speicher braucht die CPU eine Zeit, die in der Größenordnung von jeweils wenigen millionstel Sekunden liegt. Daten jedoch, die die CPU an ihre Peripherie liefert, sollen dort meist für wesentlich längere Zeit zur Verfügung stehen, und so lange kann der Datenbus unmöglich blockiert werden, wenn ein Programm laufen soll.

Wenn die Peripherie einfach an den Datenbus angeschlossen wird, dann ergibt sich als weiteres Problem, daß der Peripherie automatisch auch alle Programm-Daten übermittelt werden, die eigentlich den Datenaustausch zwischen CPU und Speicher betreffen und mit denen sie nichts anfangen kann. Wie soll die Peripherie entscheiden, welches nun Programm-Daten auf dem Datenbus sind, und welche Daten für sie bestimmt sind?

Dieses Problem ergibt sich grundsätzlich in gleicher Form auch für jede Speicherzelle. Es wird durch die Informationen auf dem Adreßbus gelöst, durch die Daten auf dem Datenbus immer nur mit der gerade adressierten Speicherzelle koordiniert werden. Es bietet sich also an, das gleiche Prinzip auch für die Peripherie anzuwenden: Jeder Peripherie-Einheit (wie immer sie auch aussehen mag) wird – wie einer Speicherzelle – eine bestimmte Adresse zugeordnet.

Es bleibt die Schwierigkeit, daß alle Daten immer nur für sehr kurze Zeit auf dem Datenbus erscheinen und dann durch andere Daten abgelöst werden. Bei Daten, die dem Speicher von der CPU übermittelt werden, spielt das keine Rolle: Es ist ja gerade die Eigenschaft des Speichers, kurzzeitig angebotene Daten für lange Zeit aufzubewahren. Auch die CPU legt die vom Speicher abgeholten Daten jeweils sofort in internen Registern ab (vgl. Bild S72.1).

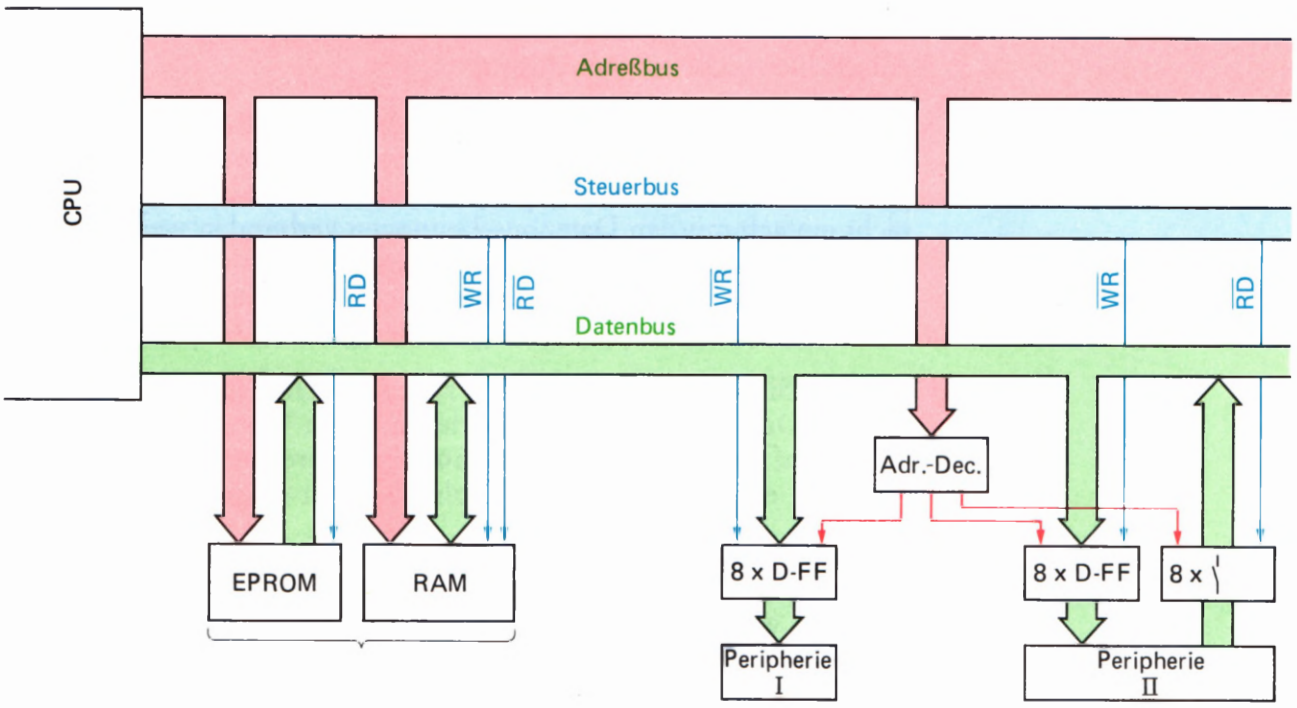
Wie im Einzelnen die Adressierung bestimmter Speicherzellen innerhalb des Speichers vorgenommen wird, braucht hier nicht zu interessieren. Wichtig ist nur die Feststellung, daß jedem Peripherie-Gerät eine (oder bei Bedarf auch mehrere) der 65536 möglichen Adressen zugeordnet wird. Das mit seiner Adresse angesprochene Peripherie-Gerät kann dann über den Datenbus – genau wie eine Speicherzelle – mit der CPU Daten in Form von 8-Bit-Worten austauschen.

Elegant ist bei diesem Verfahren, daß für den Daten-Transport von der CPU zu einem Peripherie-Gerät oder vom Peripherie-Gerät zur CPU genau die gleichen Befehle verfügbar sind, wie für den Daten-Austausch zwischen CPU und Speicher. Das Peripherie-Gerät wird von der Software behandelt wie eine Speicherzelle.

Das Bild H59.1 zeigt das Schema eines Mikroprozessor-Systems mit Festspeicher und Schreib-Lese-Speicher und zusätzlich den beiden, irgendwie gearteten Peripherie-Geräten I und II. Hier interessiert hauptsächlich der rechte Teil des Bildes mit den beiden Peripherie-Geräten.

Die beiden Peripherie-Geräte sind grundsätzlich in gleicher Weise mit dem Datenbus verbunden wie die Speicher. Die Pfeile geben die Richtung an, in der Daten fließen können. Das Peripherie-Gerät I kann also nur Daten vom Mikroprozessor empfangen. Es kann sich dabei z. B. um einen Drucker handeln. – Das Peripherie-Gerät II kann sowohl Daten vom Mikroprozessor empfangen als auch Daten an den Mikroprozessor senden. Ein solches Gerät ist z. B. eine Schreibmaschinen-Tastatur, kombiniert mit einem Bildschirm-Gerät.

Bild H59.1
Peripherie-Geräte können in gleicher Weise an das Bus-System angeschlossen werden wie Speicher. Jedes Peripherie-Gerät erhält eine oder mehrere Adressen.



Die Peripherie-Geräte selbst haben eigene Datenleitungs-Anschlüsse und gehen davon aus, daß die dort angelieferten Daten auch richtig und gültig sind. Oder aber sie liefern selbst Daten an ihre Anschlüsse und erwarten, daß sie der Mikroprozessor gefälligst im passenden Augenblick abholt und richtig verarbeitet.

Es muß nun dafür gesorgt werden, daß erstens die in irgendeinem bestimmten Augenblick vom Mikroprozessor gelieferten Daten (und nur diese!) der Peripherie auch dann noch zur Verfügung stehen, wenn diese Daten längst wieder vom Datenbus verschwunden sind, und daß zweitens die mit der Peripherie auszutauschenden Daten auch dem richtigen Peripherie-Gerät zugeführt bzw. vom richtigen Peripherie-Gerät abgeholt werden. Diese Aufgabe übernehmen die Bausteine mit den jeweils acht D-Flipflops und ein weiterer Baustein mit acht (elektronischen) Schaltern, die zwischen die Peripherie-Geräte und den Daten- und Adreßbus geschaltet sind.

Die D-Flipflops fangen die vom Mikroprozessor auf den Datenbus gelieferten und für die Peripherie bestimmten Daten in dem Augenblick auf, in dem die CPU mit einem *WR*-Signal meldet, daß diese Daten für das über den Adreßbus adressierte Peripherie-Gerät bestimmt sind. – Das *WR*-Signal hat seinen Namen vom englischen **WR***ite* (sprich: rait, Schreiben). Es meldet mit dem Wert Ø, daß die CPU Daten für eine adressierte Speicherzelle oder für ein adressiertes Peripherie-Gerät auf den Datenbus geschickt hat. Wenn die Kombination von Schreib-Signal und Adresse des Peripherie-Geräts vom Bus-System des Mikroprozessors wieder verschwindet, dann führt auch der Datenbus längst wieder andere Daten. Die D-Flipflops halten die angelieferten Daten jedoch auch weiterhin so lange für die Peripherie verfügbar, bis vom Mikroprozessor für dieses Peripherie-Gerät ein anderes Datenwort geliefert wird.

Das Signal für die D-Flipflops, Daten vom Datenbus aufzufangen, ergibt sich aus einer Verknüpfung des *WR*-Signals mit einem Ausgangs-Signal des Adreß-Decodierers, das immer nur dann aktiviert ist, wenn auf dem Adreßbus eine bestimmte Adreßbit-Kombination erscheint. Wir wollen uns hier nicht mit der internen Funktion des Adreß-Decodierers beschäftigen.

Das im Bild H59.1 dargestellte Peripherie-Gerät II kann Daten vom Mikroprozessor empfangen und außerdem auch Daten an den Mikroprozessor senden. Seine entsprechenden Anschlüsse dürfen natürlich nicht einfach mit den Datenbus-Leitungen verbunden werden: Seine Daten dürfen nur dann auf den Datenbus geschaltet werden, wenn die Daten-Quelle über ein Signal vom Adreß-Decodierer aktiviert wird und wenn der Mikroprozessor gleichzeitig ein *RD*-Signal sendet.

Das *RD*-Signal hat seinen Namen vom englischen **R***ea***D** (sprich: riid, Lesen). Der Mikroprozessor meldet mit dem Wert Ø dieses Signals, daß er auf dem Datenbus Daten aus einer adressierten Speicherzelle oder von einem adressierten Peripherie-Gerät erwartet.

Dann (und nur dann), wenn die Kombination des *RD*-Signals mit einem Signal vom Adreß-Decodierer meldet, daß die CPU die Peripherie II angewählt hat, werden die acht (elektronischen) Schalter geschlossen und die von der Peripherie gelieferten Signale auf den Datenbus geschaltet.

Die im Bild H 59.1 dargestellten beiden Peripherie-Geräte besitzen insgesamt drei Acht-Bit-Datenanschlüsse, die in der Fachsprache als **Port** bezeichnet werden. Zwei dieser Ports sind Ausgangs-Ports (oder kurz: Out-Ports) des Mikroprozessor-Systems; der dritte ist ein Eingangs-Port (kurz: In-Port). Jeder dieser Ports hat eine Adresse, die durch die Schaltung des Adreß-Decodierers festgelegt wird.

Die hier beschriebene Möglichkeit, Ports wie Speicherzellen zu behandeln und ihnen auch normale 16-Bit ≙ 2-Byte-Adressen zuzuweisen, wird in der Fachsprache als **Memory Mapped I/O** (sprich: memmori mäppt ai-ou) oder einfach als **I/O-Mapping** (sprich: ai-ou-mäpping) bezeichnet. Das englische Wort *map* bedeutet das Anordnen in einem Plan. Hier ist das Anordnen von In/Out- (= EIN/AUS-) Adressen, also von Port-Adressen, im Plan der Speicheradressen gemeint.

Parallele Aus- und Eingänge mit Port-Adressen

Der in Ihrem System verwendete Mikroprozessor Z 80 und auch der Mikroprozessor 8085 verfügen über eine besondere Einrichtung, die es erlaubt, die Adressierung von Ports für Peripherie-Geräte völlig von der Adressierung von Speicherzellen zu trennen. Diese Einrichtung geht davon aus, daß in einem Mikroprozessor-System immer weitaus mehr Speicherzellen als Ports notwendig sind.

Das Bild H 61.1 zeigt die in Ihrem System verfügbaren Speicher- und Port-Adressen. Für den Speicher stehen sämtliche 65 536 Adressen uneingeschränkt zur Verfügung; es braucht keine dieser Adressen für einen Port reserviert zu werden. – Für die Ports sind 256 eigene, sedezimal jeweils zweistellig darstellbare Adressen verfügbar.

Adressen kann der Mikroprozessor grundsätzlich nur über den Adreßbus liefern. Die Einrichtung, von der hier die Rede ist, besteht zunächst nur aus einem einzigen CPU-Anschluß, an dem ein Signal geliefert wird, das man bei oberflächlicher Betrachtung einfach als zusätzliches Adreßbit bezeichnen könnte. Mit einem solchen zusätzlichen Adreßbit ließe sich die von der CPU ansprechbare Anzahl der Adressen genau verdoppeln. Das erscheint vernünftig, denn dann könnte man die eine Hälfte der Adressen den Speicherzellen zuordnen und käme wieder zu 65 536 adressierbaren Speicherzellen. Die andere Hälfte der Adressen könnte man den Port zuordnen, und das wäre ganz unsinnig, denn soviele Ports wird man in einem System nie und nimmer adressieren wollen.

Tatsächlich verfügt die CPU unseres Systems über solch ein zusätzliches Signal-Bit. Es hat aber nicht einfach die Funktion eines Adreßbits und wird deshalb auch nicht dem Adreßbus zugeordnet, sondern dem Steuerbus, zu dem u.a. auch die Signale *RD* und *WR* gehören. Im Bild H 63.1 haben wir die Steuerbus-Leitung, die dieses Signal führt, durch eine größere Strichstärke hervorgehoben. Das Signal mit der Bezeichnung $\overline{IORQ}$ fordert mit dem Wert Ø eine Operation über einen In-Port oder über einen Out-Port an. IORQ ist die Abkürzung des englischen *Input Output ReQuest*, und das bedeutet gerade eine solche Anforderung. Solange dieses Bit den Wert 1 hat, wird mit den 16 Adreßbits eine Speicherzelle adressiert. Ist aber $\overline{IORQ}$ = Ø, dann wird über den Adreßbus einer der 256 möglichen Ports angesprochen.

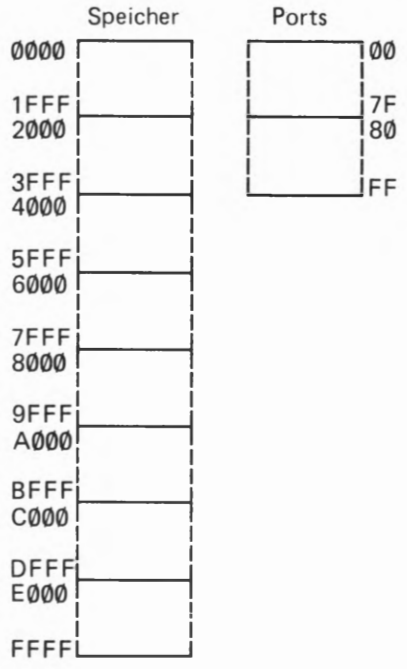


Bild H 61.1
Die Mikroprozessoren 8085 und Z 80 verfügen über eine Einrichtung, mit der bis zu 256 Ports unabhängig von der Speicheradresse adressiert werden können.

Hier liegt scheinbar ein Widerspruch vor, denn zur Adressierung eines von 256 Ports braucht man keine sechzehn Adreßbits, sondern deren nur acht. Unsere CPU verwendet bei Input- und Output-Operationen über Ports beim Wert Ø des *IORQ*-Bits tatsächlich nur die acht niederwertigen Adreßbits und macht dadurch die Adressierung eines Ports mit einem Zwei-Byte-Befehl möglich: Der dem Operationscode eines solchen Befehls folgende, einzige Operand kann die Port-Adresse darstellen. – Wir kommen darauf noch zurück.

Das Bild H62.1 zeigt die unterschiedliche Arbeitsweise der CPU bei der Adressierung einer Speicherzelle und eines Ports. Im Teilbild a ist die Ausführung des direkt adressierenden Ladebefehls LD A,(19DC) dargestellt (vgl. Seite S43). Der Mikroprozessor schaltet die im dritten und zweiten Befehlsbyte enthaltene Adresse der anzusprechenden Speicherzelle auf den Adreßbus und kopiert dann das Byte aus dieser Speicherzelle über den Datenbus in den Akkumulator.

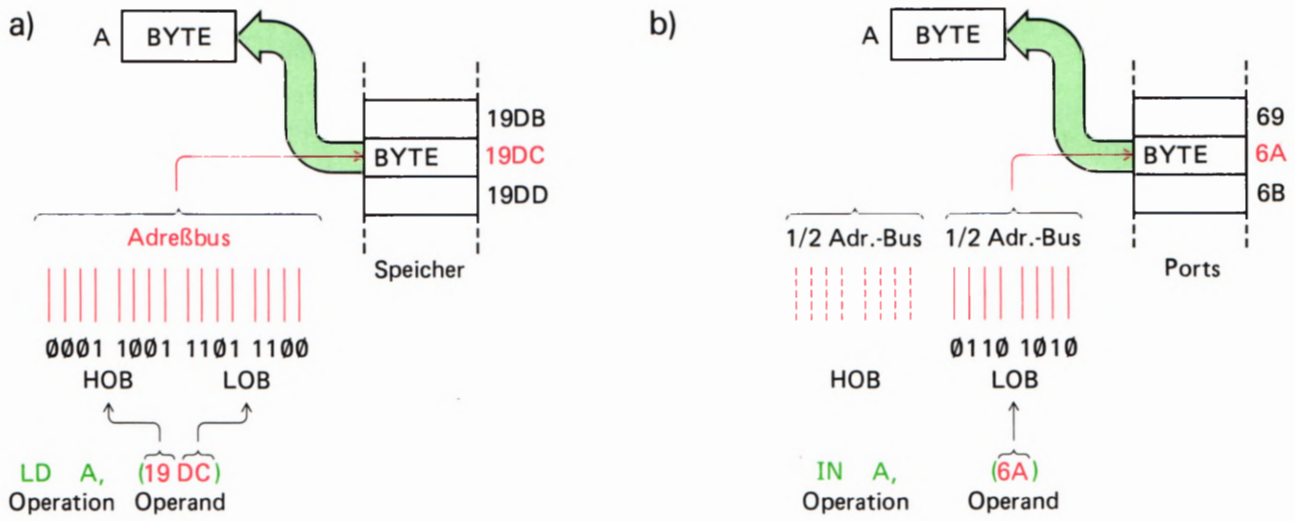
Das Teilbild b zeigt die Funktion des entsprechenden Ladebefehls IN A,(6A) mit direkter Port-Adressierung. Es handelt sich um einen **Input-Befehl** zum Abholen von Peripherie-Eingangsdaten. (Das englische Wort *input* bedeutet wörtlich: Eingeben.)

Der Ladebefehl LD A,(adr) und der Input-Befehl IN A,(n) entsprechen sich in ihrer Funktion genau. Der – wesentliche! – Unterschied besteht darin, daß beim Input-Befehl der Operand n aus einem einzigen Byte besteht, das die Port-Adresse darstellt und dessen Bits bei der Ausführung des Befehls auf die niederwertigen acht Leitungen des Adreßbus' geschaltet werden. Beim Ladebefehl stellt adr einen Zwei-Byte-Operanden dar. Die 16 Bits dieser zwei Bytes erscheinen bei der Befehls-Ausführung auf allen sechzehn Leitungen des Adreßbus'.

Die hier angestellten Überlegungen gelten übrigens in gleicher Weise auch für die entsprechenden Befehle LD (adr),A bzw. OUT (n),A, allerdings mit umgekehrter Daten-Richtung. Wir stellen Ihnen die IN- und OUT-Befehle noch vor.

Woher weiß das Mikroprozessor-System, daß z.B. beim IN A,(6A)-Befehl ein Byte vom Port mit der Adresse 6A und nicht etwa aus einer

Bild H62.1
a) Bei einem Ladebefehl mit direkter Speicheradressierung enthalten das dritte und zweite Befehlsbyte die 16-Bit-Speicheradresse.
b) Bei einem Portbefehl erscheint die 8-Bit-Port-Adresse auf den 8 niederwertigen Leitungen des Adreßbus'.



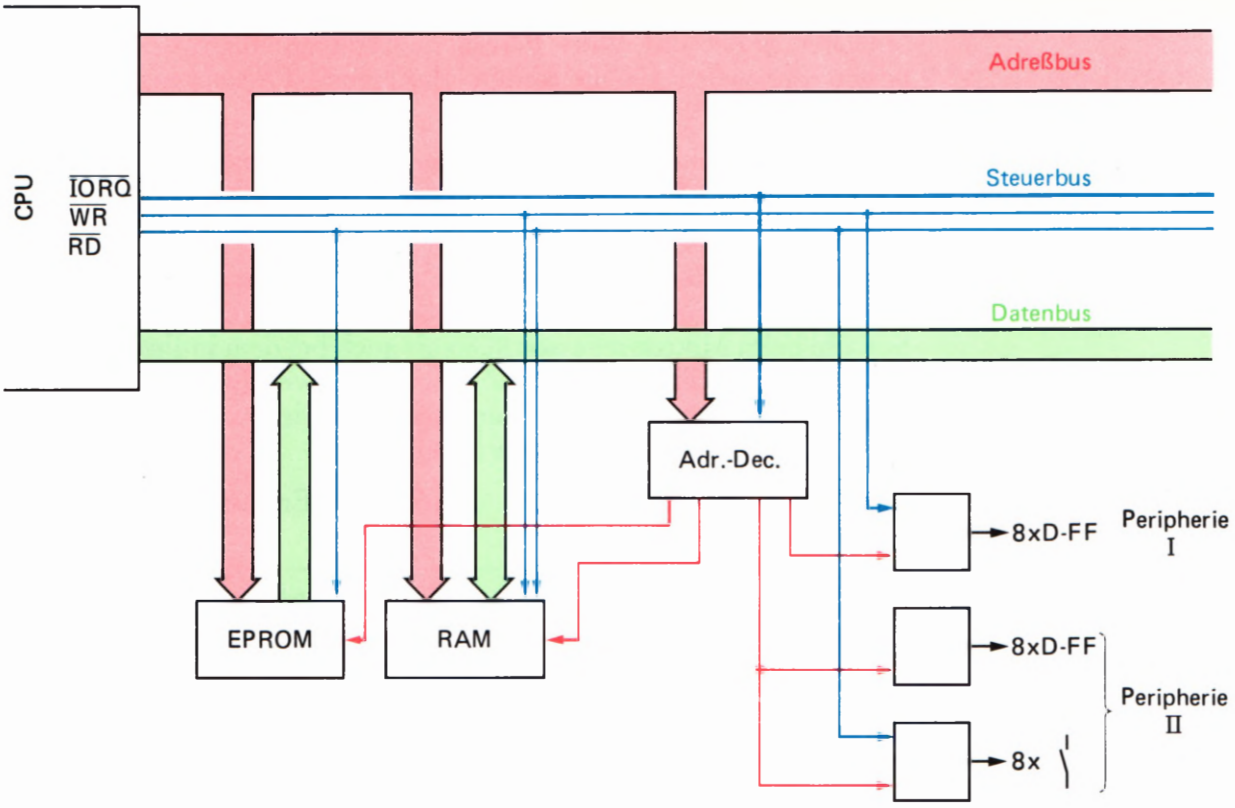


Bild H63.1
Das hier dargestellte Isolated-I/O-System unterscheidet sich vom Memory-Mapped-System im Bild H59.1 durch die Benutzung des Signals *IORQ*.

Speicherzelle abgeholt werden soll? – Bei den Port-bezogenen Befehlen IN und OUT wird von der CPU automatisch das $\overline{IORQ}$-Signal auf den Wert Ø gesetzt. Im Bild H63.1 haben wir das sehr vereinfachte Prinzip eines Systems dargestellt, das mit den gleichen Peripherie-Einheiten wie das System entsprechend dem Bild H59.1 arbeitet. Jetzt wird jedoch das dem Adreß-Decodierer zugeführte Signal $\overline{IORQ}$ benutzt, um den Peripherie-Einheiten eigene Port-Adressen zuzuordnen. Beim Ansprechen einer Port-Adresse schaltet der Adreß-Decodierer die Speicher-Bausteine ab; umgekehrt werden bei Speicher-Operationen die Ports gesperrt.

Das im Bild H63.1 dargestellte System wird in der Fachsprache als System mit **Isolated I/O** (sprich: aissoleeted ai-ou) bezeichnet.

Peripherie-Befehle

Der Transport von Daten zwischen der CPU und einem Peripherie-Gerät wird beim I/O-Mapping und beim Isolated I/O unterschiedlich gehandhabt.

Wir haben bereits auf der Seite H59 darauf hingewiesen, daß Peripherie-Geräte beim **I/O Mapping** wie Speicherzellen behandelt werden können. Entsprechend der Aufstellung auf der Seite S50 können Daten direkt adressiert oder indirekt über den Inhalt eines der Registerpaare adressiert zwischen der CPU und einem Peripherie-Gerät transportiert werden. Wenn Sie in der Aufstellung für den Fall einer

Kommunikation zwischen CPU und Peripherie jeweils das Wort Speicher durch das Wort Peripherie ersetzen, dann erkennen Sie bereits die vielen Möglichkeiten, die beim I/O-Mapping für einen solchen Daten-Transport bestehen.

Nachteilig ist beim I/O-Mapping jedenfalls, daß unter Umständen ein ganzer Adreß-Bereich, in dem man gerne Speicher anordnen möchte, durch einige wenige Peripherie-Adressen blockiert werden kann.

Die Möglichkeiten der Software, in einem **Isolated-I/O-System** mit der Peripherie zu korrespondieren, sind wesentlich bescheidener. – Sowohl beim Mikroprozessor 8085 als auch bei dem in Ihrem System verwendeten Mikroprozessor Z 80 gibt es zwei Grund-Befehle, die Daten vom Akkumulator zur Peripherie befördern und Daten von der Peripherie in den Akkumulator holen:

Mnemonischer Code	Operations-code	Operation	Erläuterung
IN A,(n)	DB n	A ← Port	Akkumulator mit dem Bitmuster laden, das am Port mit der Adresse n anliegt.
OUT (n),A	D3 n	Port ← A	Bitmuster aus dem Akkumulator an den Port mit der Adresse n liefern.

Wie bereits im Bild H 62.1 dargestellt, entsprechen diese Befehle den direkt adressierenden Speicherbefehlen LD A,(adr) beziehungsweise LD (adr),A, haben aber den Vorteil, daß es sich nur um Zwei-Byte-Befehle handelt.

Speziell der Mikroprozessor Z 80 verfügt noch über eine Reihe weiterer Befehle zum Datentransport in einem Isolated-I/O-System zwischen CPU und Peripherie, von denen wir Ihnen hier nur zwei vorstellen können. Bei diesen Befehlen muß die Ein-Byte-Port-Adresse vor der Ausführung des Befehls im C-Register abgelegt sein. Dann jedoch kann der Datentransport zwischen der Peripherie und einem beliebigen der Register A bis L programmiert werden:

Mnemonischer Code	Operation	Erläuterung
IN r,(C)	r ← Port	Das Register r wird mit dem Bitmuster geladen, das am Port mit der Adresse anliegt, die im C-Register abgelegt ist.
OUT (C),r	Port ← r	Bitmuster aus dem Register r an den Port mit der Adresse liefern, die im C-Register abgelegt ist.

Die Operationscodes dieser Befehle zeigt die folgende Tabelle:

r	A	B	C	D	E	H	L
IN r,(C)	ED 78	ED 40	ED 48	ED 50	ED 58	ED 60	ED 68
OUT (C),r	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

Der Stack zur Daten-Ablage

Die wichtigsten Register des in Ihrem System verwendeten Mikroprozessors sind Ihnen durch die Versuche in unserem Lehrgang inzwischen recht vertraut geworden. Zwar verfügt der Z 80-Mikroprozessor noch über einige Spezial-Register und außerdem über einen kompletten zweiten Registersatz (Seite T 2); mit diesen Registern können wir uns jedoch im Rahmen dieses Grundlehrgangs nicht beschäftigen. Ein sehr wichtiges Register müssen wir Ihnen aber noch vorstellen: Den Stack-Pointer. Auf der Seite T 2 erkennen Sie, daß dieses Register 16 Bit breit ist, und das ist ein Hinweis darauf, daß dieses Register zur Aufnahme von Adressen prädestiniert ist.

Das Retten von Register-Inhalten

Bei den in unserem Lehrgang beschriebenen Programmen sind wir nie in die Verlegenheit gekommen, daß die in der CPU verfügbaren Register beim Verarbeiten von Daten nicht ausgereicht haben. Bei komplexeren Programmen kann aber sehr bald der Fall eintreten, daß sämtliche Register mit irgendwelchen aktuellen Daten belegt sind und unbedingt noch weitere Daten untergebracht und bearbeitet werden müssen. Man reserviert deshalb für solche kurzzeitig aufzubewahrenden Daten einen bestimmten Speicherbereich, der als **Scratchpad** (vgl. Seite H 44) bezeichnet wird. In diesem Bereich kann man auch zwischenzeitlich die Inhalte von CPU-Registern ablegen, wenn diese Register vorübergehend anderweitig gebraucht werden. Nach Erledigung dieser Aufgabe kann man dann die CPU-Register wieder mit ihren ursprünglichen Werten aus dem Scratchpad laden.

Eine solche Daten-Ablage hat aber den gewichtigen Nachteil, daß dazu entweder die Drei-Byte-Befehle LD (adr),A bzw. LD A,(adr) mit direkter Adressierung der Speicherzelle oder aber die über Registerpaare indirekt adressierenden Zwei-Byte-Befehle verwendet werden müssen (vgl. Seite S 50). Das setzt jedoch wiederum voraus, daß eins der Registerpaare zur Aufnahme der Ablage-Adresse verfügbar ist.

Die Notwendigkeit des „Rettens" von Register-Inhalten ergibt sich am häufigsten, wenn in einem Programm Unterprogramme aufgerufen werden, die vielleicht – wie manche Unterprogramme des Betriebsprogramms Ihres Systems – ursprünglich für eine ganz andere Verwendung geschrieben wurden. Beim Entwurf eigener Programme erweisen sich diese Routinen dann jedoch ebenfalls als zweckmäßig. Es ergibt sich lediglich die Schwierigkeit, daß diese Routinen ganz ungeniert eine Reihe von CPU-Registern verwenden, die vielleicht in Ihrem Hauptprogramm vor dem Aufruf des Unterprogramms mit wichtigen Daten belegt sind. Es bleibt Ihnen nichts anderes übrig, als Ihre eigenen Register-Inhalte in den Speicher zu retten und sie nach Erledigung des Unterprogramms wieder aus dem Speicher in die CPU zurückzuholen, so daß Ihr Programm ordnungsgemäß weiter laufen kann.

Dieses „Retten" von Register-Inhalten könnte unter Umständen so aufwendig sein, daß man gar auf die Verwendung von nützlichen Unterprogrammen verzichten möchte – wenn nicht der Hersteller

Ihres Mikroprozessors diese Schwierigkeit vorhergesehen und entsprechend komfortable Möglichkeiten zur Verfügung gestellt hätte.

Ähnlich wie die Ladebefehle für die 16 Bit breiten Registerpaare BC, DE und HL (Seite H36) gibt es im Befehlssatz unseres Mikroprozessors einen Befehl zum Laden des ebenfalls 16 Bit breiten Stack-Pointers:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD SP,Konst	31	SP←Konst	Der Stack-Pointer wird mit einer sedezimal vierstellig dargestellten Zahl Konst geladen.

Der Mikroprozessor interpretiert den Inhalt des Stack-Pointers als Adresse, die nach der Programmierung dieses Ladebefehls um eins höher ist als die oberste Adresse eines in seinem Umfang nicht definierten Speicherbereichs. Dieser Speicherbereich unterhalb der im Stack-Pointer programmierten Adresse wird als **Stack** bezeichnet und ist speziell zur mehr oder weniger kurzzeitigen Ablage von Register-Inhalten vorgesehen. (Bild H66.1.)

Das englische Wort *stack* (sprich: stäck) bedeutet wörtlich Stapel. Entsprechend wird der Stack-Pointer oft auch mit dem deutschen Wort **Stapel-Zeiger** bezeichnet. Der Name ist ein wenig irreführend, denn unter einem Stapel stellt man sich etwas vor, das bei Ablegen von Dingen in die Höhe wächst. Der Stack des Mikroprozessors dagegen wächst bei der Ablage von Register-Inhalten in Richtung kleinerer Adressen, also „in den Keller". Statt von einem Stack spricht man deshalb auch vom **Keller-Speicher**. Obwohl diese Bezeichnung der Eigenschaft des Ablage-Speicher-Bereichs näher kommt, bleiben wir hier bei der Bezeichnung Stack.

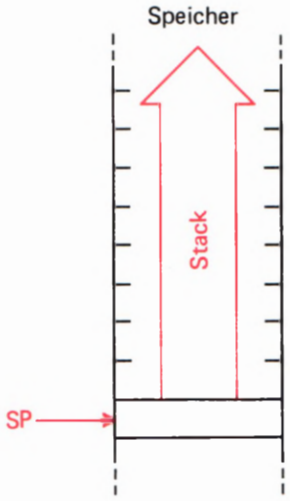


Bild H66.1
Der Stack wird als Ablage-Bereich für Daten im Speicher ab einer Adresse angelegt, die durch den anfänglichen Inhalt des Stackpointers festgelegt wird.

Für das Ablegen von Register-Inhalten stellt der Befehlssatz Ihres Mikroprozessors folgende Befehle zur Verfügung:

Mnemonischer Code	Operations-code	Operation	Erläuterung
PUSH AF	F5	Stack←(AF)	Der Inhalt eines Registerpaares wird in den Speicherzellen abgelegt, deren Adressen um 1 und um 2 kleiner sind als der augenblickliche Inhalt des Stackpointers. Der Inhalt des Stackpointers wird zweimal dekrementiert.
PUSH BC	C5	Stack←(BC)	
PUSH DE	D5	Stack←(DE)	
PUSH HL	E5	Stack←(HL)	

Der mnemonische Code PUSH (sprich: pusch) kommt vom englischen Wort *push*, was wörtlich schieben bedeutet: Die Bytes werden auf den Stack „geschoben".

Diese Aufstellung zeigt, daß mit einem einzigen Ein-Byte-Befehl die Inhalte von gleich zwei je acht Bit breiten Registern auf den Stack gerettet werden können. Es handelt sich dabei jeweils um die zwei Register eines Registerpaares. Interessant ist, daß in diesem Zusammenhang auch der Akkumulator und das Flag-Register sozusagen als Registerpaar betrachtet werden, obwohl diese beiden Register viel weniger verwandt sind als z. B. das B- und das C-Register.

Vor allem zeigt die Erläuterung der Befehle, daß der Stackpointer im Zusammenhang mit den PUSH-Befehlen eine ganz besondere Eigenschaft hat, die Sie sich am besten in einem Versuch ansehen:

Versuch H67.1

Die PUSH-Befehle

Tasten Sie bitte in Ihr System die im Bild H67.1 aufgelistete Befehlsfolge ein. Setzen Sie einen Breakpoint bei der Adresse 18Ø8:

ADDR, 1, 8, 0, 8, SBR

Starten Sie die Befehlsfolge bei der Adresse 18ØØ und überzeugen Sie sich anschließend mit Hilfe der Taste REG, daß die ersten vier Befehle die CPU-Register so geladen haben, wie es bei den Bemerkungen zu den Befehlen in der Liste angegeben ist. – Betätigen Sie anschließend die Taste PC.

Betätigen Sie die Taste STEP zur Ausführung der Anweisung Nr. 5 in einem Einzelschritt! – Der Inhalt des Akkumulators und der hier nicht weiter interessierende Inhalt des Flag-Registers sind jetzt „auf dem Stack", also in den beiden Speicherzellen abgelegt, die unmittelbar unter der Speicherzelle liegen, auf die der Stack-Pointer anfangs zeigte (1F9E, 1F9D).

Sehen Sie sich die aktuellen Inhalte des Akkumulators und den des Flag-Registers an:

REG, AF Anzeige: 0 1 X X A F

Wählen Sie anschließend die Adresse 1F9E an! Sie finden in dieser Speicherzelle den Wert Ø1 aus dem Akkumulator. Betätigen Sie die Taste – ! Bei der Adresse 1F9D wird der Inhalt XX des Flag-Registers angezeigt. – Betätigen Sie die Taste PC.

Lassen Sie mit der Taste STEP die Anweisung Nr. 6 als Einzelschritt ausführen, und vergleichen Sie die Inhalte der Speicherzellen mit den

Bild H67.1
Mit dieser Befehlsfolge werden im Versuch H67.1 die Inhalte der CPU-Register auf dem Stack abgelegt.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	LADEN	LD	A, Ø1	18ØØ	3E	Ø1		(A) = Ø1
2		LD	BC,2345	18Ø2	Ø1	45	23	(B) = 23; (C) = 45
3		LD	DE,6789	18Ø5	11	89	67	(D) = 67; (E) = 89
4		LD	HL,ABCD	18Ø8	21	CD	AB	(H) = AB; (L) = CD
5	RETTEN	PUSH	AF	18ØB	F5			Register-Inhalte retten
6		PUSH	BC	18ØC	C5			
7		PUSH	DE	18ØD	D5			
8		PUSH	HL	18ØE	E5			
9		RST	3Ø	18ØF	F7			Breakpoint

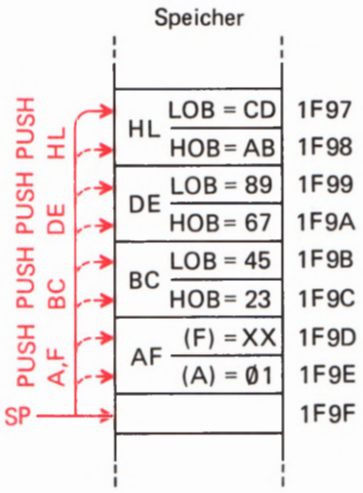


Bild H68.1
Inhalt des Stacks nach Ausführung der PUSH-Befehle im Versuch H67.1.

Adressen 1F9C und 1F9B mit dem aktuellen Inhalt des BC-Registerpaares! Auch dieser Inhalt ist in zwei Speicherzellen auf dem Stack abgelegt.

Wiederholen Sie die hier beschriebenen Schritte jeweils auch nach der Ausführung der Anweisungen Nr. 7 und Nr. 8, und vergessen Sie nicht, nach der Kontrolle des Stack-Inhalts die Taste PC zu betätigen. Im Bild H68.1 haben wir den Inhalt des Speichers nach der Ausführung der Anweisung Nr. 8 eingetragen.

Löschen Sie den Inhalt des Stacks zwischen den Adressen 1F97 und 1F9F (Eintragen der Bytes ØØ bei diesen Adressen) und betätigen Sie anschließend die Taste RS. Löschen Sie den Breakpoint (Taste CBR) und starten Sie nun das Programm bei der Adresse 18ØØ mit der Taste GO. – Der Befehl RST 3Ø sorgt dafür, daß sich Ihr System nach dem Abarbeiten der Befehlsfolge mit der Adresse 181Ø meldet. – Kontrollieren Sie jetzt den Inhalt des Stack-Pointers, in den Ihr System nach dem Betätigen der Taste RS den Wert 1F9F geladen hatte. Der Inhalt des Stack-Pointers wird angezeigt, wenn Sie nacheinander die Tasten REG und SP (von rechts gezählt zweite Taste in der zweiten Reihe des Tastenfeldes) betätigen. Sie finden (SP) = 1F97.

Wenn wir Sie nicht bereits bei der Erläuterung der PUSH-Befehle (Seite H66) darauf aufmerksam gemacht hätten, dann wäre diese Anzeige im höchsten Maße verblüffend: Die Befehlsfolge im Bild H67.1 enthält keinen speziellen Befehl zum Ändern des Stack-Pointer-Inhalts, und trotzdem zeigt der Stack-Pointer auf die Speicherzelle im Stack, in die der letzte PUSH-Befehl ein Byte gerettet hat. Dies ist die besondere Eigenschaft der PUSH-Befehle: Sie retten den Inhalt eines Registerpaares auf den Stack und aktualisieren gleichzeitig den Inhalt des Stack-Pointers. Dadurch ist der Stack-Pointer nach jedem PUSH-Befehl automatisch so eingestellt, daß der nächste PUSH-Befehl seine Bytes richtig „oben" auf den Stapel legt. (Daß „oben" hier in Wirklichkeit „unten" ist, soll Sie nicht stören; der Stack ist ja ein Keller-Speicher.)

Sie können sich diesen Vorgang genau ansehen, wenn Sie noch einmal bei der Adresse 18Ø8 einen Breakpoint setzen, das Programm bei der Adresse 18ØØ starten und anschließend die Anweisungen Nr. 5 bis Nr. 8 in Einzelschritten ausführen lassen. Jetzt brauchen Sie nicht mehr nach jedem Einzelschritt den Inhalt des Stacks zu kontrollieren. Sehen Sie sich dafür jeweils den Inhalt des Stack-Pointers an! Sie erkennen, daß sein Inhalt nach jedem PUSH-Befehl zweimal dekrementiert wird. (Vergessen Sie nicht, den Breakpoint zu löschen!)

Das Wieder-Herstellen von Register-Inhalten

Das Retten von Register-Inhalten auf den Stack ist nur dann sinnvoll, wenn es den PUSH-Befehlen entsprechende Befehle gibt, mit denen zum passenden Zeitpunkt die Register-Inhalte vom Stack wieder abgeholt und in die richtigen Register gebracht werden können.

Der Befehlssatz unseres Mikroprozessors stellt zu jedem PUSH-Befehl den passenden POP-Befehl zur Verfügung, mit dem zwei Byte vom Stack abgeholt und in ein bestimmtes Registerpaar eingetragen werden:

Mnemonischer Code	Operations-code	Operation	Erläuterung
POP AF	F1	AF ← (Stack)	Der Inhalt der Speicherzelle, deren Adresse augenblicklich im Stack-Pointer steht, und der Inhalt der nächsthöheren Speicherzelle werden in einem Registerpaar abgelegt. Gleichzeitig wird der Inhalt des Stack-Pointers zweimal inkrementiert.
POP BC	C1	BC ← (Stack)	
POP DE	D1	DE ← (Stack)	
POP HL	E1	HL ← (Stack)	

Der mnemonische Code POP entspricht dem englischen Wort *pop* (sprich: popp), mit dem ein schnelles Abholen beschrieben wird: Die Inhalte von zwei Speicherzellen des Stacks werden mit einem Befehl von nur ein Byte Länge in ein Registerpaar gebracht.

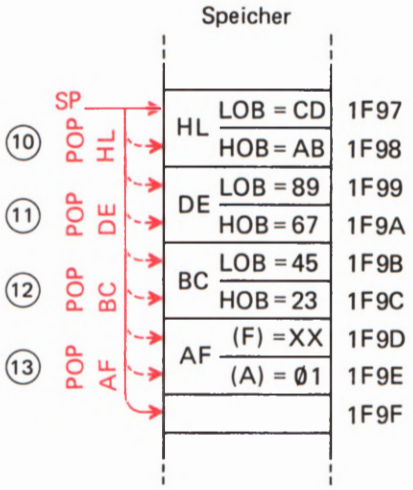


Bild H69.1
Im Versuch H69.1 sorgen die POP-Befehle dafür, daß die CPU-Register mit den auf dem Stack abgelegten Daten geladen werden.

Versuch H69.1

Die POP-Befehle

Wir gehen davon aus, daß Ihr System noch mit der Befehlsfolge aus dem Bild H67.1 geladen ist. – Tasten Sie bitte zusätzlich zu diesen Befehlen ab der Adresse 18ØF die im Bild H69.2 aufgelisteten Befehle ein. Dabei wird der vorher programmierte Befehl RST 3Ø bei der Adresse 18ØF überschrieben. – Tasten Sie bitte ab der Adresse 1829 das im Bild S99.2 als Hex-Dump angegebene Unterprogramm ALARM ein.

Starten Sie das Programm bei der Adresse 18ØØ! – Es passiert nichts sensationelles: Das Alarm-Signal ertönt, und anschließend bleibt das Programm bei der Adresse 1817 stehen.

Die Wirkungsweise des Programms ist ebenfalls einfach: Zunächst werden die CPU-Register mit definierten Werten geladen, und dann wird das Unterprogramm ALARM aufgerufen. Innerhalb dieses Unterprogramms werden (außer dem C-Register) sämtliche CPU-Register gebraucht und ihre Inhalte geändert. (Sie dürfen uns das glauben, auch ohne daß wir den Ablauf dieses Unterprogramms im einzelnen beschreiben.)

Starten Sie das Programm noch einmal bei der Adresse 18ØØ und sehen Sie sich nach dem Anhalten die Inhalte der Register an! Sie finden die Register-Inhalte von Ø1 bis CD, die von den ersten Befehlen geladen

Bild H69.2
Ergänzen Sie bitte für den Versuch H69.1 die Befehlsfolge im Bild H67.1 um die hier aufgelisteten Befehle.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
9		CALL	ALARM	18ØF	CD	29	18	Unterprogramm
10		POP	HL	1812	E1			Register-Inhalte
11		POP	DE	1813	D1			wieder herstellen
12		POP	BC	1814	C1			
13		POP	AF	1815	F1			
14		RST	3Ø	1816	F7			Breakpoint

worden sind. – Der Vorgang ist durchsichtig: Vor dem Aufruf des Unterprogramms wurden diese Inhalte mit PUSH-Befehlen auf den Stack gebracht; nach dem Ablauf des Unterprogramms wurden die Inhalte mit POP-Befehlen wieder vom Stack abgeholt und in den richtigen CPU-Registern abgelegt.

Die Betonung liegt dabei auf „in den richtigen". – Sehen Sie sich bitte noch einmal das Bild H68.1 an! Nach dem letzten PUSH-Befehl zeigt der Stack-Pointer auf die Speicherzelle, in welcher der Inhalt des L-Registers abgelegt wurde. Dorthin zeigt er auch nach dem Ablauf des Unterprogramms (Bild H69.1).

> Damit das Laden der Register richtig erfolgt, muß als erster der POP-Befehl für das Registerpaar programmiert werden, das als letztes mit einem PUSH-Befehl auf den Stack gebracht wurde.

Das Bild H69.1 zeigt, wie sich der Stack-Pointer bei den POP-Befehlen verhält. Sie können sich den Inhalt des Stack-Pointers nach dem Ablauf des Programms ansehen und sich davon überzeugen, daß er wieder auf die Speicherzelle mit der Adresse 1F9F zeigt und damit bereit ist, bei neuen PUSH-Befehlen den Stack von Grund auf neu aufzubauen. Der alte – und nun nicht mehr benötigte – Stack-Inhalt wird dann einfach überschrieben.

Der Programmzähler-Inhalt auf dem Stack

Bei der Beschreibung des Versuchs H69.1 und der dabei vorgenommenen Untersuchung des Stack-Pointers haben wir Ihnen eine wichtige Tatsache unterschlagen, die Sie sich im folgenden Versuch ansehen sollen:

Versuch H70.1

Retten des Programmzähler-Inhalts

Für diesen Versuch muß Ihr System mit den Befehlsfolgen in den Bildern H67.1 und H69.2 und ab der Adresse 1829 mit den Bytes für das Unterprogramm ALARM aus dem Bild S99.2 geladen sein.

Setzen Sie einen Breakpoint bei der Anfangsadresse 1829 des Unterprogramms ALARM und starten Sie das Programm bei der Adresse 18ØØ!

Ehe der Alarmton kommen kann, wird das Programm angehalten. Dabei wird der erste Befehl des Unterprogramms gerade noch ausgeführt und dann erscheint in der Anzeige die Adresse 182B des nächstfolgenden Befehls im Unterprogramm, der uns hier jedoch nicht interessiert. Wichtig ist nur, daß das Unterprogramm richtig angesprungen worden ist.

Sehen Sie sich jetzt bitte den Inhalt des Stack-Pointers an:

REG, SP Anzeige: 1 F 9 5 S P

Entsprechend unserer vorhergehenden Betrachtung müßte der Stack-Pointer nach der Ausführung der vier PUSH-Befehle auf die Adresse 1F97 zeigen (vgl. Bild H68.1). Daß er auf eine Speicherzelle zeigt, die um zwei unter Speicherzelle mit der Adresse 1F97 liegt, läßt vermuten, daß das Programm noch zwei weitere Bytes auf den Stack gebracht hat, anscheinend auch ohne daß wir einen entsprechenden Befehl programmiert haben.

Sehen Sie sich doch einfach einmal an, was da auf dem Stack los ist:

ADDR, 1, F, 9, E, —, —, —, ...

Im Bild H71.1 haben wir den Inhalt des Stacks dargestellt. Er entspricht bis zur Adresse 1F97 genau dem Inhalt, den Sie bereits aus den Bildern H68.1 und H69.1 kennen. Darüber hinaus finden Sie bei der Adresse 1F96 das Byte 18 und bei der Adresse 1F95 das Byte 12. Wie auch bei den anderen abgelegten Byte-Paaren ist sicher bei der höheren Adresse ein HOB abgelegt und bei der niedrigeren Adresse ein LOB. Daraus ergibt sich offenbar der Inhalt eines 16 Bit breiten Registers, und wenn das so ist, dann handelt es sich um die Adresse 1812. — Was hat es mit dieser Adresse auf sich?

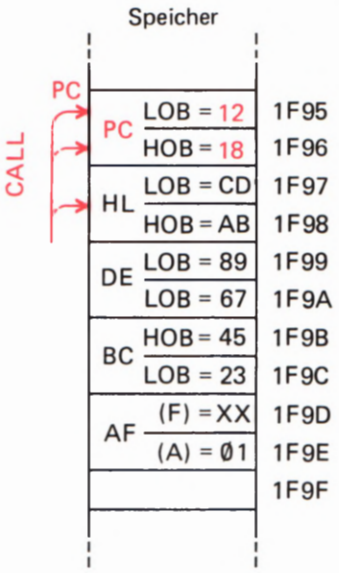


Bild H71.1
In den Versuchen H69.1 und H70.1 wird vom CALL-Befehl der Inhalt des Programmzählers auf dem Stack abgelegt.

Das Bild H69.2 zeigt, daß nach der Ausführung des Befehls CALL ALARM (der sozusagen stellvertretend für die Befehle im Unterprogramm ALARM steht) das Hauptprogramm mit dem Befehl POP HL fortgesetzt wird. Und dieser Befehl steht gerade bei der Adresse 1812, die nach dem Aufruf des Unterprogramms ganz „oben" auf dem Stack abgelegt ist.

Das ist es: Beim Ansprung eines Unterprogramms merkt sich der Mikroprozessor, bei welcher Adresse er die Ausführung des Hauptprogramms nach dem Abarbeiten des Unterprogramms fortsetzen muß, indem er einfach die auf den CALL-Befehl im Hauptprogramm folgende Adresse auf dem Stack ablegt. Da der Programmzähler nach dem Einholen (Fetch) des letzten Bytes eines Befehls automatisch inkrementiert wird und damit auf die Adresse des ersten Bytes des folgenden Befehls zeigt (Bild S72.1), ergibt sich folgende Aufgabe:

> Bei der Ausführung des CALL-Befehls zum Aufruf eines Unterprogramms legt die CPU zunächst den aktuellen Inhalt des Programmzählers auf dem Stack ab und lädt den Programmzähler dann mit der Anfangsadresse des Unterprogramms.

Der CALL-Befehl ist demnach die Kombination des Befehls für einen absolut adressierenden Sprung JP mit einem push pc-Befehl zum Ablegen des Programmzähler-Inhalts auf dem Stack, der einzeln im Befehlssatz des Mikroprozessors nicht vorgesehen ist.

Jetzt wird klar, wie der Mikroprozessor es anstellen kann, daß er auch nach mehrmaligem Aufruf des gleichen Unterprogramms an verschiedenen Stellen des Hauptprogramms anschließend das Hauptprogramm jeweils an der richtigen Stelle fortsetzt (vgl. Seite S97). Es muß dazu nur am Ende eines jeden Unterprogramms eine Art pop pc-Befehl pro-

grammiert werden, der dafür sorgt, daß der Programmzähler zur Fortsetzung des Programms mit der Adresse geladen wird, die vorher beim Aufruf des Unterprogramms auf dem Stack abgelegt worden ist.

Ein solcher pop pc-Befehl ist in dieser Form im Befehlssatz des Mikroprozessors nicht enthalten. Er existiert, aber er hat einen anderen mnemonischen Code, der seine Funktion deutlicher beschreibt:

Mnemonischer Code	Operationscode	Operation	Erläuterung
RET	C9	PC←(Stack)	Das Programm wird bei der Adresse fortgesetzt, bei welcher der auf den Unterprogramm-Aufruf folgende Befehl programmiert ist.
RET c	Tabelle	PC←(Stack) wenn c erfüllt	Wenn die Bedingung c erfüllt ist, wird das Programm bei Adresse fortgesetzt, bei welcher der auf den Unterprogramm-Aufruf folgende Befehl programmiert ist.

Der mnemonische Code RET kommt vom englischen **RET***urn* (sprich: retörn) und bedeutet wörtlich: Kehre (vom Unterprogramm zum Hauptprogramm) zurück. – Die folgende Tabelle zeigt die Operationscodes der bedingten RET-Befehle:

Flag	Z		CY		P/V		S	
c	Z	NZ	C	NC	PE	PO	M	P
RET	C8	CØ	D8	DØ	E8	EØ	F8	FØ

Weil die RET-Befehle (nicht eigens im Befehlssatz enthaltenen) pop pc-Befehlen entsprechen, ist mit ihrer Ausführung immer ein zweimaliges Inkrementieren des Stack-Pointer-Inhalts verbunden. So kommt es, daß Sie beim Versuch H 69.1 das Ablegen und Zurückholen des Programmzähler-Inhalts überhaupt nicht bemerkt haben. Der CALL-Befehl hat den Stack-Pointer-Inhalt zweimal dekrementiert, und danach wurde am Ende des Unterprogramms ALARM mit einem RET-Befehl der Stack-Pointer-Inhalt wieder zweimal inkrementiert.

Interessant ist, daß die RET-Befehle – wie die CALL-Befehle – nicht nur zu unbedingter, sondern auch zu bedingter Ausführung programmiert werden können. Ein Unterprogramm kann also nicht nur mit einem unbedingten RETurn-Befehl zur Rückkehr zum Hauptprogramm abgeschlossen werden; es kann auch programmiert werden, daß diese Rückkehr nur unter bestimmten Bedingungen stattfindet.

Lehrgang MIKROPROZESSORTECHNIK · Herausgeber: R. Christiani

Software

Software

Mag eine elektronische Schaltung noch so kompliziert sein, mag sich ihr Schaltplan über mehrere Seiten erstrecken und eine Unzahl unterschiedlicher elektronischer Bauelemente enthalten: Erschrecken wird sie den versierten Elektroniker kaum. Soll er ein solches Gebilde in eine funktionierende Schaltung verwandeln, dann kann er – zumindest grob – den Zeitaufwand und vor allem den Preis dieser Hardware abschätzen. Er hat es mit einer ihm wohl vertrauten Materie zu tun. Er weiß, wo er die Bauelemente kaufen kann und kennt ihren Preis. Ist die Schaltung dann aufgebaut, dann braucht er einen Signalgenerator, verschiedene Meßinstrumente, ein Oszilloskop, etwas Zeit und Geduld – und er wird das Ding schon „zum Laufen" bringen.

Geradezu wie Hexerei muß ihm dagegen eine Anlage vorkommen, die vielleicht nur aus recht wenigen unterschiedlichen Bauelementen besteht, die aber trotzdem eine komplexe Aufgabe lösen kann und nach Betätigung einiger Tasten durch Änderung des Programms auf einmal ein ganz anderes Verhalten an den Tag legt.

Was ist das – ein Programm? Es ist fast buchstäblich die Seele der ganzen Anlage. Nicht etwa das Herz. Das ist die CPU, ein elektronisches Bauelement gewohnter Form, das man ansehen, anfassen und notfalls in der Rocktasche verschwinden lassen kann.

Das Programm kann man nicht anfassen, es steht geschrieben auf Papier, besteht vielleicht aus Löchern in einem Papierstreifen, also aus Nichts mit was drumrum. Eigentlich ein Witz.

Ein Programm ist keine Hardware, klirrt und klappert nicht, wenn es auf den Boden fällt. Und doch ist es die Seele der Anlage, ohne die sie nicht läuft. Ein unfaßbares, „weiches" Gebilde. So haben es die Amerikaner genannt, und so wollen wir es auch nennen: Software (sprich: ssoftwär), weiche Ware.

Der Programmablaufplan

Technik ist eine präzise Sache, zu deren Beschreibung auch präzise Ausdrücke gehören. Programmablaufplan ist ein präziser Ausdruck. Leider ist Präzision nicht zwangsläufig mit Schönheit verbunden, aber das soll nicht unsere Sache sein. Im Englischen gibt es für das, womit wir uns in diesem Kapitel beschäftigen wollen, den schlichten Ausdruck *Flowchart* (sprich: floutschart), das wörtlich etwa „Flußtabelle" heißt. In Anlehnung daran heißt das Ding im Umgangs-Fachjargon auch oft **Flußdiagramm** – nur, damit Sie Bescheid wissen.

Programmablaufpläne enthalten Operationen, Unterprogramme, Verzweigungen, Übergangsstellen, Ein- und Ausgaben usw. Wir wollen in diesem Abschnitt untersuchen, was es damit auf sich hat.

S

2

Anweisungen

Ein Programm gibt es im Zirkus, im Theater und auch im Radio und Fernsehen. Wahrscheinlich liegt neben Ihrem Fernsehgerät auch ein „Programm", aus dem Sie ablesen, wann die Übertragung des Fußballspiels stattfindet. Wir wollen präzise sein und müssen deshalb feststellen, daß es sich bei dieser Auflistung der zeitlichen Folge bestimmter Sendungen keineswegs um das Programm selbst handelt. Das Programm ist die Folge der einzelnen Sendungen, die Ihnen die Sendeanstalt übermittelt. Bei der Auflistung der einzelnen Sendungen handelt es sich bereits um eine einfache Form eines Programmablaufplans.

Grundsätzlich ist Ihnen ein Programm nichts neues oder fremdes. Von diesem allgemein bekannten Begriff eines Programms wollen wir zunächst ausgehen. Zwar haben wir auf der Seite H 2 bereits angedeutet, welche Bewandtnis es mit einem Programm im Zusammenhang mit Mikroprozessoren hat: Es handelt sich um eine **Anweisung** in Form von Befehlen, die zur Lösung einer Aufgabe mit dem Mikroprozessor notwendig ist. Da wir uns mit diesen Befehlen noch nicht beschäftigt haben, lassen wir diese spezielle Form eines Programmablaufplans zunächst außer Betracht.

Abgesehen von der Urlaubszeit, in der man gern systematisch faulenzt, läuft unser tägliches Leben nach einem ganz bestimmten Programm ab. Sogar die von Berufsarbeit freie Zeit ist – mit entsprechenden Variationsmöglichkeiten – in dieses Programm eingeschlossen.

Ganz losgelöst von der Zeit, zu der man seinen Wecker gestellt haben mag, soll (für Sie ganz unverbindlich!) ein Teil dieses Programms im folgenden dargestellt und untersucht werden. Eventuelle Abweichungen von dem von Ihnen als normal empfundenen Programm bitten wir dabei großzügig zu entschuldigen. Die einzelnen Schritte dieses Programms wollen wir zur besseren Übersicht numerieren.

1. Wecker rappelt	6. Rasieren	11. Nach Kaffee rufen
2. Gähnen	7. Waschen	12. Frühstücken
3. Aufstehen	8. Kämmen	13. Verabschieden
4. Ins Bad gehen	9. Ankleiden	14. Mantel anziehen
5. Zähne putzen	10. Ins Eßzimmer gehen	15. Spiegel gucken usw.

Diese Auflistung enthält die einzelnen Programmabschnitte in Form von Stichworten. Sie können diese Stichworte aber auch als Anweisungen betrachten, die von einer inneren Stimme gegeben werden. Lediglich der erste Punkt ist keine Anweisung; er stellt den Beginn des täglichen Programms dar.

Der hier aufgestellte Programmablaufplan ist nicht sehr fein unterteilt; einige dieser Anweisungen können in weitere Anweisungen unterteilt werden. Zum Beispiel:

6a Elektrischen Rasierer von der Wand nehmen
 b Anschalten
 c Rasierer zum Kinn heben usw.

Am grundsätzlichen Aufbau des Programmablaufplans ändert das jedoch nichts. Wir haben den Plan bei der Nummer 15 vorzeitig abge-

brochen; bis zum Schlafengehen ließen sich auf diese Weise mühelos viele Seiten füllen.

Ansonsten bedarf dieser Plan keiner näheren Erläuterung. Er ist auf einen ganz normalen Morgen zugeschnitten.

Verzweigungen und Schleifen

Wie funktioniert das Programm aber, wenn Sie den Punkt 3 mit dem linken Bein beginnen, wenn also nicht alles programmgemäß abläuft? Betrachten wir den Punkt 11. Sie sind fertig angekleidet, sitzen im Eßzimmer am Frühstückstisch und rufen nach dem Kaffee. Nur: Der Kaffee kommt nicht; er ist noch nicht fertig. Nun können Sie nicht zum Programmpunkt 12 übergehen und frühstücken. Das Programm funktioniert nicht in der angegebenen Form. Diese Störung im Programm muß berücksichtigt werden.

Wahrscheinlich wird die Sache so ablaufen:

11. Nach Kaffee rufen
    11a warten
    11b schimpfen
    11c warten
    11d nach Kaffee rufen
    11e warten
    11f Kaffee kommt
12. Frühstücken

Es soll hier vorsichtshalber nicht untersucht werden, welch unliebsame Weiterungen sich daraus ergeben, daß Sie mit dem linken Bein zuerst aufgestanden sind und auf den Kaffee warten mußten. Wir beschränken uns auf die reine Programmtechnik.

Schauen Sie sich den beim Punkt 11 erweiterten Programmablaufplan an! Das sieht zwar recht plausibel aus, funktioniert aber leider nicht ganz.

Die innere Programmstimme muß den Programmablauf Ihres frühen Morgens doch in jedem Fall berücksichtigen, egal, ob der Kaffee nun sofort kommt oder nicht. In der jetzt vorgesehenen Erweiterung dagegen tun wir so, als hätten Sie jedenfalls immer erst mal Grund zum Schimpfen. Was doch sicherlich nicht stimmt.

Wir wollen einmal pessimistisch sein: Sind Sie sicher, daß der Kaffee nach dem ersten Warten, nach einmaligem Schimpfen und neuerlichem Warten wirklich kommt?

Wir sprechen nun nicht mehr von Ihnen selbst, denn Sie sind ganz sicher höflich genug, nach dem Warten in die Küche zu gehen und zu helfen. (Wahrscheinlich hätten Sie nicht einmal geschimpft.) Wir tun so, als hätten wir es mit einem ganz ungehobelten Burschen zu tun. Wie wird er sich verhalten?

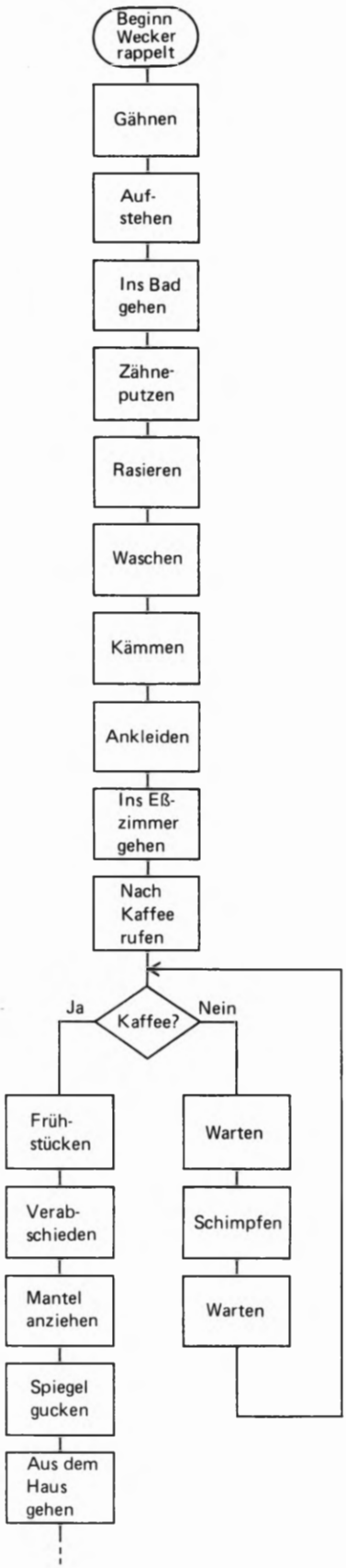


Bild S4.1
In diesem Programmablaufplan tritt in Abhängigkeit von äußeren Umständen eine Verzweigung auf.

Nach dem Warten beim Punkt 11c wird er zunächst wieder nach seinem Kaffee rufen (Punkt 11) und einen Augenblick warten. Aber dann schimpft er von neuem los, wartet, ruft, wartet, schimpft, wartet, ruft wieder, wartet, schimpft... und das geht so weiter, bis endlich der Kaffee kommt. Und dann? Dann tut er (hoffentlich!) das gleiche, was er auch getan hätte, wenn der Kaffee gleich nach dem ersten Rufen gekommen wäre.

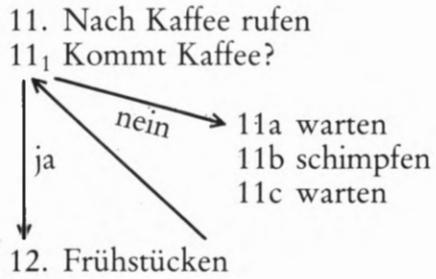
Sie erkennen: Nach Erledigung des Programmpunktes 11 (nach Kaffee rufen) muß das Programm – je nach den äußeren Umständen (Kaffee kommt oder Kaffee kommt nicht) – unterschiedliche Programmabläufe vorsehen.

Kommt der Kaffee sofort, dann kann auf den Programmschritt 11 sofort der Programmschritt 12 (Frühstücken) folgen.

Kommt der Kaffee jedoch nicht, dann läuft das Programm anders ab: Es wird gewartet, geschimpft, wieder gewartet und dann der Programmabschnitt 11 wiederholt. Nun ist der Programmablauf so weit wie vorher: Je nachdem, ob der Kaffee kommt oder nicht kommt, wird entweder der Programmabschnitt 12 (Frühstücken) erledigt oder erneut das Schimpfprogramm gewählt.

Abweichend vom bisherigen Programmablaufplan muß nun allerdings ein zusätzlicher Programmschritt eingefügt werden, der die Entscheidung für einen der beiden **Zweige** im Programmablaufplan ermöglicht: Es muß eine **Verzweigung** vorgesehen werden.

Bei der von uns gewählten Darstellung des Programmablaufplans sähe das so aus:



Diese Art der Darstellung ist nicht sehr übersichtlich. Üblicher und wesentlich übersichtlicher ist die im Bild S4.1 gewählte Darstellung. Wir wollen die einzelnen Schritte innerhalb des Programms als **Operationen** bezeichnen und dem Charakter nach unterschiedlichen Operationen entsprechende Sinnbilder zuordnen. Eine normale Operation (Gähnen, Waschen usw.) wird durch einen rechteckigen Kasten dargestellt. Für die Verzweigung wählen wir als Sinnbild eine Raute. Am Anfang des Programmablaufplans finden Sie eine **Grenzstelle**, die keine Operation versinnbildlicht. Am hier nicht betrachteten Ende des Programms stände nach der Operation „Einschlafen" eine entsprechende End-Grenzstelle.

Ein sozusagen problemlos ablaufendes Programm ohne die Möglichkeit einer Verzweigung ist von der Programmerstellung her eine recht langweilige Angelegenheit. Interessant wird die Sache immer erst dann, wenn der Programmablaufplan Besonderheiten, z. B. in Form von Verzweigungen, enthält. Im Bild S4.1 haben wir gleich zwei solcher Besonderheiten eingebaut: Eine **Verzweigung** und eine **Schleife**.

Eine Verzweigung im Programmablaufplan liegt dann vor, wenn im Programmablauf einer von zwei (oder mehreren) Zweigen eingeschlagen wird.

Eine solche Verzweigung tritt in unserem Beispiel bei der Entscheidung „kommt Kaffee?“ auf. Der eine mögliche Zweig führt direkt zur Operation „Frühstücken“, der andere führt in eine Schleife, die bei der Verzweigung beginnt und nach der zweiten Operation „Warten“ über die Zusammenführung der Programmwege wieder zur Verzweigung gelangt.

Eine Programm-Schleife entsteht aus einer Verzweigung und einer Zusammenführung, wenn in Richtung des Programmablaufs die Schleife bei der Verzweigung beginnt und über die Zusammenführung wieder zur Verzweigung führt.

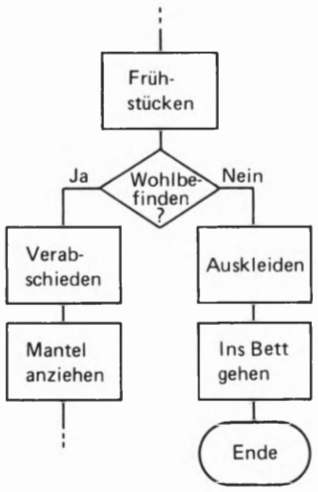


Bild S5.1
Bei einer Verzweigung kann der eine Programmzweig statt zu einer Schleife auch zum Ende des Programms führen.

Natürlich sind auch Verzweigungen ohne angeschlossene Schleife möglich. Wir wollen einen solchen Fall betrachten, indem wir unser Programm nach dem Frühstücken etwas erweitern. Im Bild S5.1 haben wir den entsprechenden Teil des Programmablaufplans dargestellt. Die Darstellung spricht für sich selbst. Von der Verzweigung an führt der eine Weg wie vorher letztlich zu „aus dem Haus gehen“, der andere wieder ins Bett. Hier landet der Programmablaufplan an einer Grenzstelle, an der das Tagesprogramm ein vorzeitiges Ende nimmt.

Unterprogramme

Es wurde bereits auf der Seite S3 darauf hingewiesen, daß unser Programmablaufplan recht grob ist. Jede einzelne Anweisung aus dem Bild S4.1 müßte bei einer feineren Unterteilung in mehrere oder viele Einzelanweisungen gegliedert sein. Sie können sich vorstellen, daß z. B. die Anweisung „Kämmen“ in einem ausführlichen Programmablaufplan aus einer großen Anzahl von Anweisungen bestehen muß, ähnlich wie wir es auf der Seite S3 für die Anweisung „Rasieren“ angedeutet haben.

Mit gutem Grund könnten wir nun unser hier als Beispiel gewähltes Programm um etliche Anweisungen „Kämmen“ erweitern. Die Anweisung „Kämmen“ könnte dann etwa zusätzlich wiederholt werden hinter der Anweisung „Ankleiden“, hinter den Anweisungen „Mantel anziehen“, „Spiegel gucken“ und – nicht zu vergessen! – hinter der Anweisung „Verabschieden“. (Da vielleicht ganz besonders.)

Den Programmablaufplan im Bild S4.1 an den genannten Stellen jeweils um die Operation „Kämmen“ zu erweitern, erweist sich nicht

S
6

als besonders umständlich. Wie sieht die Sache aber aus, wenn es sich um einen ausführlichen Programmablaufplan handelt? Dann müßte an jeder dieser Stellen jeweils die ganze Serie der Kämm-Anweisungen eingefügt werden. Abgesehen davon, daß das eine Menge Schreibarbeit bedeutet, müßte die innere Stimme, die sich die Anweisungen für den programmierten Menschen merken muß, schon ein recht umfangreiches Gedächtnis haben.

Die Anweisung „Kämmen" können wir in unserem Programm als **Programmbaustein** betrachten, der dann an verschiedenen Stellen des Programms in unveränderter Form eingesetzt wird. Es handelt sich um eine **Prozedur**, die bei wiederholter Verwendung innerhalb des Programms in ihren Einzelheiten nur einmal als Programmablaufplan beschrieben werden muß.

Immer dann, wenn wir in unser Programm die Anweisung „Kämmen" einbauen wollen, brauchen wir nur eine **Prozeduranweisung** zu geben, die die Prozedur „Kämmen" entsprechend der Darstellung im Bild S7.1 als **Unterprogramm** aufruft. Im Bild S6.1 haben wir den so erweiterten Teil des im Bild S4.1 dargestellten Programms aufgezeichnet. Die Unterprogramm-Aufrufe haben wir zur Verdeutlichung rot eingetragen. Sie erkennen, daß man das Unterprogramm als normale Anweisung mit zusätzlichen seitlichen Balken versinnbildlicht.

Unterprogramme werden oft mit dem aus der englischen Sprache entlehnten Wort **Subroutine** bezeichnet, oder auch einfach als **Routine.**

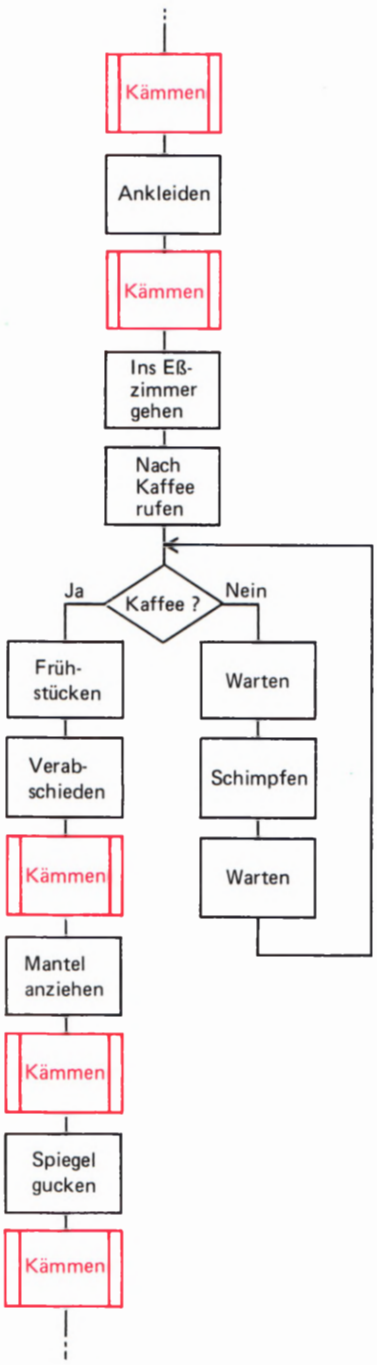


Bild S6.1
Ein Teil des im Bild S4.1 dargestellten Programms wird hier durch Einfügungen mehrerer Prozeduren „Kämmen" erweitert, die man als Unterprogramm entsprechend dem Bild S7.1 einfügen kann.

> Ein Unterprogramm ist ein Programmbaustein, der eine zur Lösung einer Aufgabe vollständige Anweisung enthält. Es wird in einem Programm wiederholt aufgerufen, braucht aber nur einmal als vollständige Anweisung aufgezeichnet zu werden.

Aufgabe S6.1

Bei dem im Bild S6.1 dargestellten Teil des Programmablaufplans haben wir es nicht nur mit einem ungehobelten Burschen zu tun, der so lange schimpft, bis endlich sein Kaffee auf dem Frühstückstisch steht. Maßlos eitel ist er zudem. Bei jeder nur möglichen Gelegenheit schwingt er den Kamm, gleichgültig, ob das notwendig ist oder nicht.

Normalerweise wird man sich ja nur dann kämmen, wenn die Frisur auch wirklich in Unordnung geraten ist. – Ändern Sie das Programm aus dem Bild S6.1 so ab, daß nach der Anweisung „Verabschieden" der Kamm nur dann benutzt wird, wenn das Haar tatsächlich „verwuschelt" ist.

Fügen Sie dazu eine Anweisung „Spiegel gucken" ein und danach eine Verzweigung, die den Aufruf des Unterprogramms „Kämmen" davon abhängig macht, ob die Frisur in Ordnung ist oder nicht.

Sie brauchen nur den geänderten Programmteil zu skizzieren.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü1.

Merker (Flags)

Die am häufigsten in einem Programmablaufplan auftretenden Sinnbilder haben Sie nun kennengelernt. Wenn wir später Programme für den Mikroprozessor entwerfen, führen wir bei Bedarf noch weitere Sinnbilder ein, die uns vorläufig aber noch nicht zu interessieren brauchen.

Wir wollen jedoch noch zwei Besonderheiten betrachten, deren Kenntnis sich beim Entwurf eines Programmablaufplans als sehr nützlich erweist.

Betrachten Sie bitte noch einmal den Programmablaufplan im Bild S4.1! Die als vorletzte stehende Anweisung „Spiegel gucken" hätte eigentlich auch weggelassen werden können; es sei denn, sie diente nur der Betrachtung der eigenen Schönheit, also der Eitelkeit. Richtig sinnvoll wäre sie doch nur, wenn der Blick in den Spiegel z. B. der Feststellung diente, ob man die richtige Kleidung trägt. Wenn ja, dann kann man beruhigt aus dem Hause gehen. Wenn man aber vor dem Griff zur Haustürklinke feststellt, daß die geblümte Krawatte wohl doch nicht recht zu den Jeans paßt, dann ist es sicher zweckmäßig, sich neu anzukleiden.

Nach der Anweisung „Spiegel gucken" fügt man also sinnvoll eine Verzweigung ein, die in Abhängigkeit von der Entscheidung „Kleidung in Ordnung?" entweder direkt durch die Haustür führt (Ja), oder aber noch einmal die Anweisung „Ankleiden" mit allen (hier nicht aufgeführten) Einzelheiten folgen läßt (Nein) und dann erst den Schritt auf die Straße gestattet.

Programmtechnisch ergeben sich jetzt zwei Möglichkeiten. Die eine kennen Sie bereits: Man kann ein Unterprogramm „Ankleiden" schreiben und dieses Unterprogramm sowohl nach „Waschen" und „Kämmen" als auch gegebenenfalls nach dem abschließenden Blick in den Spiegel aufrufen.

Manchmal erweist sich jedoch eine andere Möglichkeit als zweckmäßiger. Bei der Entscheidung „Kleidung in Ordnung? – Nein!" führt die Verzweigung im Programmablaufplan rückwärts zur bereits angeführten Anweisung „Ankleiden". Zwischen den Anweisungen „Kämmen" und „Ankleiden" erfolgt also eine Zusammenführung der Wege vom Beginn des Programms her und von der Verzweigung. Im Bild S7.2 haben wir den Programmablaufplan mit dieser Modifikation dargestellt. Die hier nicht interessierenden Anweisungen wurden weggelassen.

Fällt Ihnen auf, daß wir hier den ganz typischen Programmablauf bei einem zerstreuten Professor dargestellt haben? Nachdem der arme Mann festgestellt hat, daß er z. B. zwei verschiedene Schuhe angezogen hat, kleidet er sich neu an – ganz folgerichtig. Dann geschieht jedoch das Erstaunliche: Angestammter Gewohnheit folgend marschiert er nach dem Ankleiden nicht etwa, wie man erwarten sollte, zur Haustür, sondern ins Eßzimmer, um zu frühstücken. Ganz sicher wird er jetzt nach dem Rufen nach Kaffee in der Schimpf-Schleife landen.

Was ist hier verkehrt? Wir müssen unbedingt einen für den nicht zerstreuten Normalbürger ganz selbstverständlichen Vorgang im Programm berücksichtigen. In dem im Bild S7.2 dargestellten Programm-

Bild S7.1
Dieses Programm „Kämmen" kann in einem anderen Programm als Programmbaustein dienen. Es wird dann bei Bedarf als Unterprogramm aufgerufen.

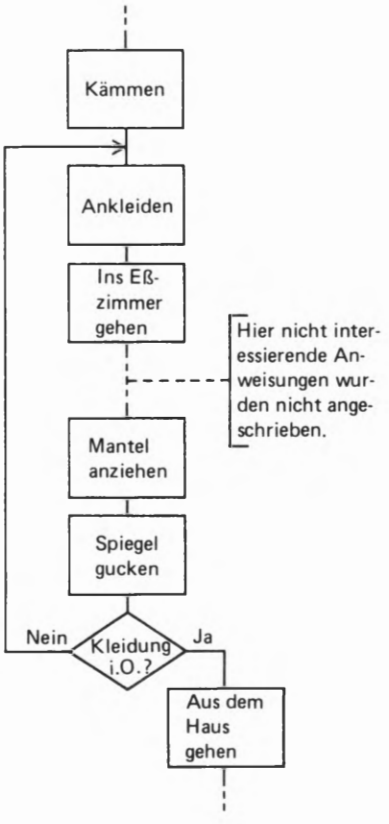


Bild S7.2
Programmablaufplan eines zerstreuten Professors. – Bei der Verzweigung können der „Ja"- und der „Nein"-Zweig beliebig angeordnet sein. Anmerkungen werden in der hier gezeigten Form angebracht.

ablaufplan muß der Programmablauf nach der Anweisung „Ankleiden" verzweigen. Nehmen wir an, unser programmierter Bürger besäße eine Komfortwohnung mit eigenem Ankleidezimmer. Je nach dem Weg, auf dem er sein Ankleidezimmer betreten hat, muß er es nach dem Ankleiden wieder verlassen: Kommt er aus dem Bad, dann führt der Weg nach dem Ankleiden ins Eßzimmer zum Frühstück. Kommt er jedoch vom Spiegel an der Haustür, dann muß der Weg nach dem neuerlichen Ankleiden wieder zur Haustür führen.

Beim nicht zerstreuten Normalbürger funktioniert dieser Mechanismus mit Sicherheit automatisch. Beim Entwurf eines Programmablaufplans müssen wir jedoch immer mit dem Schlimmsten rechnen, daß nämlich der Kopf des Programmierten voll von hoher Wissenschaft ist, und kein Platz für die Speicherung des bereits abgelaufenen Programms bleibt.

Wie helfen wir der gequälten Kreatur? Am besten legen wir an der Tür des Ankleidezimmers ein kleines Fähnchen bereit. Kommt der zerstreute Professor ganz normal aus dem Bad ins Ankleidezimmer, dann kümmert er sich nicht um das Fähnchen. Nach dem Ankleiden sieht er kein Fähnchen und marschiert wie gewohnt zum Frühstück.

Kommt er jedoch mit verschiedenen Schuhen vom Spiegel an der Haustür, dann setzt er das bereitliegende Fähnchen in eine dafür vorgesehene Öse. Das kann er nicht vergessen, denn dafür wird er eigens von uns programmiert. Verläßt er jetzt das Ankleidezimmer, dann ist das Fähnchen nicht zu übersehen und er nimmt den richtigen Weg zur Haustür. Jetzt müssen wir allerdings aufpassen. Wir dürfen unter keinen Umständen vergessen, ihn per Programm anzuweisen, das Fähnchen wieder aus der Öse herauszunehmen, ehe er zur Haustür geht.

Aufgabe S 8.1

Warum ist es so überaus wichtig, daß im zuletzt beschriebenen Fall das Fähnchen wieder entfernt wird?

Überlegen Sie, was passiert, wenn die Anweisung zum Entfernen des Fähnchens vergessen wird!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü1.

Fähnchen der hier beschriebenen Art spielen beim Entwurf von Programmabläufen oft eine wichtige Rolle. Man bezeichnet sie als **Merker**.

> Ein Merker ist ein binäres Speicherglied, das durch einen von zwei möglichen Zuständen den späteren Programmablauf an Verzweigungen zu beeinflussen ermöglicht.

In der englischen Fachsprache bezeichnet man einen solchen Merker ganz bildhaft als *Flag* (sprich: fläg), was wörtlich mit „Flagge" übersetzt werden kann. Dieser Ausdruck hat sich auch in der deutschen Fachsprache eingebürgert. Das Setzen eines Merkers bezeichnet man

als Flag Setzen, das Entfernen eines Merkers als Flag Löschen. Beim Umgang mit dem Mikroprozessor erweist es sich oft als zweckmäßig, den Ausdruck Flag zu verwenden.

Im Bild S 9.1 haben wir den Programmteil aus dem Bild S 7.2 so erweitert, daß der bisher falsch programmierte Professor nicht mehr zweimal frühstücken muß, wenn er irrtümlich zwei verschiedene Schuhe angezogen hat.

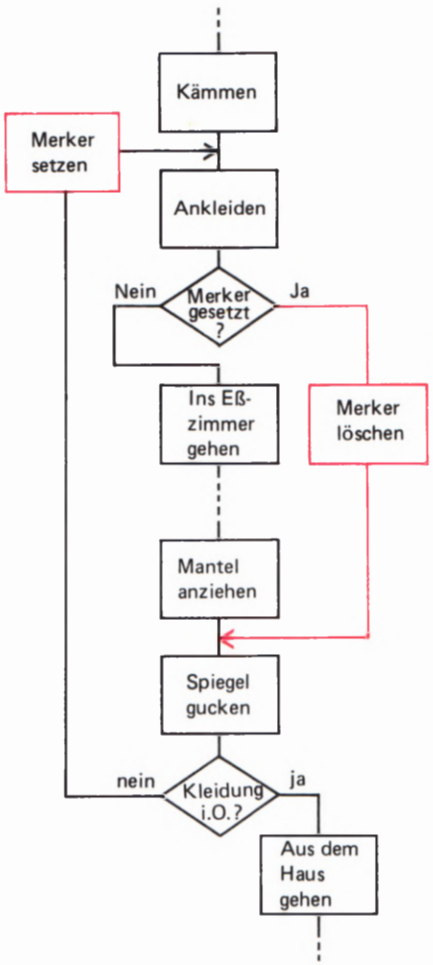


Bild 9.1
Die rot eingetragene Verzweigung und die Merker-Anweisungen verbessern den Programmablaufplan aus dem Bild S 7.2 so, daß die Gefahr des doppelten Frühstücks vermieden wird.

Aufgabe S 9.1

Entwerfen Sie – ausgehend von der Darstellung im Bild S 4.1 – einen Programmablaufplan, in dem eine Anweisung „Waschen" nach dem Frühstück bedarfsweise vorgesehen ist. Bei diesem Übungsbeispiel braucht Sie nicht zu stören, daß die weiter oben im Programm eingebaute Anweisung „Waschen" vermutlich eine umfangreichere Aktion vorsieht, als nach dem Frühstück notwendig ist. Sie können diese (im ausführlichen Programmablaufplan aus vielen Einzelanweisungen bestehende) Anweisung also benutzen.

Verwenden Sie für Ihre Programm-Erweiterung einen Merker: Machen Sie also nicht von der Unterprogramm-Technik Gebrauch.

Bedenken Sie bitte, daß die Reinigung im Bad stattfinden soll und daß – aus welchen Gründen auch immer – die Anweisung „Zähneputzen" nach dem Frühstück nicht benötigt wird. (Sie sind sicher mit uns anderer Meinung; da es sich hier jedoch nicht um einen Hygiene-Lehrgang handelt, sei uns diese Nachlässigkeit verziehen.)

Sie brauchen, ähnlich wie im Bild S 9.1, nur die Teile des Programmablaufplans zu skizzieren, die hier von Interesse sind.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 2.

Programmunterbrechungen (Interrupts)

Für die zweite der angekündigten Besonderheiten im Programmablauf können wir im Programmablaufplan kein Sinnbild anbringen. Es ist geradezu die charakteristische Eigenschaft dieser Besonderheit, daß sie keinen festen Platz im Programmablaufplan einnehmen kann.

Was werden Sie tun, wenn morgens gerade dann das Telephon klingelt, wenn Sie auf dem Wege vom Schlafzimmer zum Bad sind? Oder wenn das Telephon die Ausführung der vielleicht etwas in die Länge gezogenen Anweisung „Verabschieden" stört? Vermutlich gehört schon mehr als Charakterstärke dazu, zu sagen: Und wenn's der Kaiser von China ist – ich hör' nicht hin. Ganz ähnliche Wirkungen hat wahrscheinlich das Läuten der Haustürglocke. Vielleicht ist es der Geldbriefträger, der den Lottogewinn auszahlen möchte!

Kurz: Bei sehr vielen Programmabläufen gibt es Ereignisse, auf die sofort eine Reaktion ausgelöst werden muß, ganz unabhängig davon, welche Anweisung des Programmablaufplans gerade ausgeführt wird. Ein solches Ereignis muß eine **Programmunterbrechung** auslösen, für

S
10

die außerhalb des normalen Programmablaufs bestimmte Anweisungen als Unterprogramm vorgesehen sind.

Eine Programmunterbrechung läßt sich im zeitlichen Ablauf des Programms nicht unterbringen. Sie wissen ja nicht, wann genau das Telephon läuten wird. Sie haben auch dann keinen Einfluß auf diesen Zeitpunkt, wenn Sie den Anruf im Laufe des frühen Morgens erwarten.

Ganz ähnlich ist es mit der Haustürglocke. Auch der Geldbriefträger richtet sich normalerweise nicht nach Ihrem Programm. Trotzdem müssen Anweisungen als Unterprogramm vorgesehen werden, die beim Auftreten einer Programmunterbrechung aufgerufen werden.

Häufig wird eine Programmunterbrechung mit dem im Englischen genau „Unterbrechung" bedeutenden Ausdruck **Interrupt** bezeichnet (sprich: interapt).

Zur Ausführung einer Interrupt-Routine (vgl. Seite S6), also des Unterprogramms, das bei einer Programmunterbrechung ablaufen soll, muß im programmierten System selbst die Bereitschaft vorhanden sein. Wenn das programmierte System ein Mensch ist, dann muß dieser willens sein, eine gemeldete Programmunterbrechung anzunehmen. Handelt es sich um ein Mikroprozessor-System, dann muß diese Bereitschaft in der Hardware, meist innerhalb der CPU, eigens vorgesehen und hergestellt werden.

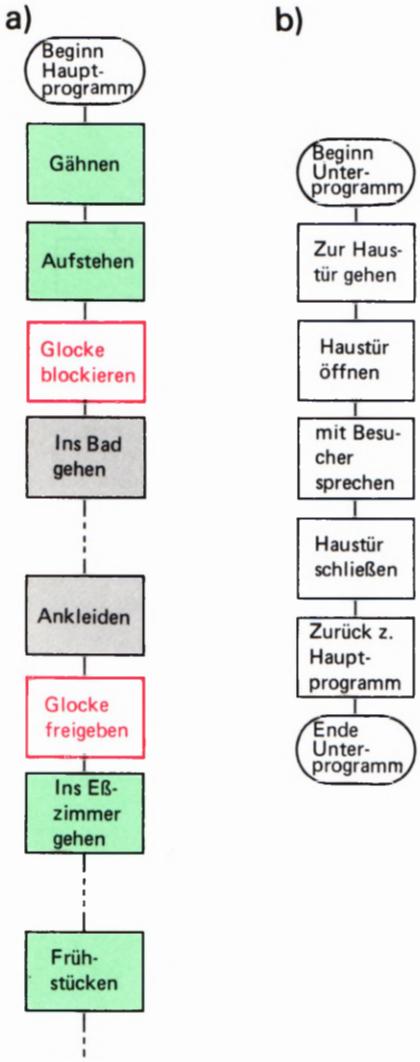


Bild S10.1
a) Eine Programmunterbrechung durch die Haustürglocke ist nur während der grün unterlegten Anweisungen möglich.
b) Dieses Unterprogramm wird von einer Programmunterbrechung aufgerufen.

Sehr nützlich ist es, wenn man die Bereitschaft, auf eine Programmunterbrechung mit dem Ablauf eines Unterprogramms zu reagieren, durch das Programm steuern kann.

In unserem hier untersuchten Beispiel ist das bei der Haustürglocke leicht möglich: Man fügt einfach in den Programmablaufplan Anweisungen zum Blockieren oder Freigeben der Haustürglocke ein. Solange die Glocke blockiert ist, kann die Nachbarin auf den Klingelknopf drücken, so lange sie will: Sie hören es nicht und bleiben zwischen Aufstehen und Frühstück ungestört. Erst nach Freigabe der Glocke kurz vor dem Frühstück sind Sie wieder bereit, einen Besucher zu empfangen.

Im Bild S10.1a haben wir die Anweisungen rot dargestellt, welche die Bereitschaft für eine Programmunterbrechung steuern. Während der grün unterlegten Anweisungen ist eine Programmunterbrechung möglich, nicht aber während der grau unterlegten Anweisungen.

Das von einer Programmunterbrechung aufgerufene Unterprogramm zeigt das Teilbild b. Dort steht als letzte Anweisung „Zurück zum Hauptprogramm", die bei derartigen Interrupt-Routinen besonders wichtig ist. Es sei hier nur angemerkt, daß zu Beginn der Interrupt-Routine ein irgendwie gearteter Mechanismus eingebaut sein muß, der es ermöglicht, an ihrem Ende wieder an genau die Stelle des Hauptprogramms zurückzukehren, an der die Programmunterbrechung zum nicht vorhersagbaren Zeitpunkt erfolgte.

Programmunterbrechungen können natürlich in der Praxis durch mehr als einen Vorgang ausgelöst werden. Eine zweite Möglichkeit der Programmunterbrechung haben wir bereits angedeutet: Auch beim Läuten des Telephons kann das Programm unterbrochen werden. (Leider läßt die Post das Blockieren der Glocke nicht ohne weiteres

zu.) Allerdings muß jetzt ein anderes Unterprogramm als beim Läuten der Haustürglocke vorgesehen werden.

Die Möglichkeiten für mehrere Programmunterbrechungen müssen im programmierten System selbst vorgesehen werden, beim Mikroprozessor also in der Hardware. Bei aufwendigen Systemen kann man eine ganze **Interrupt-Hierarchie** aufbauen. Unter einer Hierarchie versteht man eine Rangordnung. Den verschiedenen Programmunterbrechungen können also unterschiedliche Wichtigkeiten zugeordnet werden. Daraus folgt beispielsweise, daß beim gleichzeitigen Auftreten von zwei Programmunterbrechungen die wichtigere, also die mit der höheren Rangordnung, zuerst erledigt wird. So werden Sie etwa, wenn gleichzeitig mit der Haustürglocke das Telephon läutet, zuerst ans Telephon gehen, den Anruf erledigen und dann erst die Haustür öffnen.

Außerdem ist es möglich, das nach einer Programmunterbrechung ablaufende Unterprogramm seinerseits zu unterbrechen und zwischendurch diese wichtigere Programmunterbrechung „abzuarbeiten". Beispiel: Sie sind gerade bei der Anweisung „Mit Besucher sprechen" (Bild S10.1b). Wenn jetzt das Telephon klingelt, dann lassen Sie den Besucher einen Augenblick warten, erledigen den Anruf, widmen sich danach wieder Ihrem Besucher und kehren schließlich zum Frühstück im Hauptprogramm zurück.

Je nachdem, wie aufwendig das programmierte System ist, kann man eine mehr oder weniger komplizierte **Interrupt-Schachtelung** vorsehen, bei der eine Programmunterbrechung, die ihrerseits eine Interrupt-Routine unterbrochen hat, selbst wieder unterbrochen werden kann.

Der Programmablaufplan beim Mikroprozessor

Wir haben uns in diesem Abschnitt recht eingehend mit einem richtig oder falsch programmierten, wohl- oder übellaunigen Frühaufsteher beschäftigt. Sie haben die Verhaltensweise eines zerstreuten Professors kennengelernt, aber was in aller Welt hat das, so werden Sie fragen, mit Mikroprozessoren zu tun?

Die Gedankengänge, die dem Programmieren eines Mikroprozessors zugrunde liegen, sind viel menschlicher, viel mehr unserem gewohnten Denken angepaßt, als die Denkschemata beim Entwurf einer digitalen Hardware.

Sie werden später beim Programmieren eines Mikroprozessor-Systems ganz genau das gleiche Schema eines Programmablaufplans verwenden, das wir hier an einem alltäglichen Beispiel studiert haben. Der wirklich einzige Unterschied besteht in der Art der Anweisungen, die Sie dem Mikroprozessor geben müssen. Einen frühstückenden Mikroprozessor gibt es glücklicherweise nicht, zu waschen braucht er sich auch nicht, und verabschieden wird er sich schlimmstenfalls mit der dem Elektroniker wohlbekannten Rauchwolke. Aber das wird man nur selten als Anweisung vorsehen.

Fassen wir zusammen:

> Ein Programm ist die zur Lösung einer Aufgabe vollständige Anweisung.

> Der Programmablaufplan ist die Darstellung aller beim Programmablauf möglichen Wege.

Der Programmablaufplan kann Verzweigungen enthalten, die in Abhängigkeit von einer Entscheidung das Programm auf unterschiedlichen Wegen ablaufen lassen. Nach einer Verzweigung kann das Programm auch in eine Schleife führen, die beliebig oft durchlaufen werden kann.

Von Unterprogrammen macht man Gebrauch, wenn ein Programmbaustein innerhalb eines Programms mehrfach benutzt wird.

Merker (Flags) können den Programmablauf an Verzweigungen beeinflussen.

Durch eine Programmunterbrechung (Interrupt) kann ein bestimmtes Unterprogramm an nicht vorhersehbaren Stellen im Programmablauf aufgerufen werden.

Die Darstellung von Zahlen durch Zeichen

Eine Zahl scheint ein recht einfaches Ding zu sein – und ist doch ein sehr kompliziertes Etwas. Wenn man Sie fragt, was „5" sei, dann werden Sie sagen, es handle sich doch ganz offensichtlich um die Zahl Fünf – was denn sonst? Stimmt ja gar nicht! „5" ist ein Zeichen, dargestellt durch Druckerschwärze auf Papier. Es ist eine Ziffer – sonst nichts!

Was ist denn nun eine Zahl? Man könnte sagen: Eine Zahl ist eine Menge von Dingen. Aber damit begeben wir uns in ein gefährliches Fahrwasser. Wir wollen diese Frage den Mathematikern und Philosophen überlassen und uns hier darauf einigen, zwischen Zahlen (die wir als Wort schreiben wollen) und Ziffern zur Darstellung dieser Zahlen zu unterscheiden.

Zählen ist wieder eine andere Sache. Wenn ein Kind fünf Äpfel hat, dann tippt es beim Zählen nacheinander auf die Äpfel und sagt „eins" beim ersten Apfel, „zwei" beim zweiten, „drei" beim dritten usw. Stimmt ja wieder nicht! Der dritte Apfel ist ein Apfel wie der erste und keineswegs „drei". Sie sehen, wie kompliziert das ist.

Der Mikroprozessor ist ein zahlenverarbeitendes System. Seine Befehle erhält er in Form von Zahlen – etwas anderes als Zahlen kann er einfach nicht verstehen und verarbeiten. Selbstverständlich versteht der Mikroprozessor keine gesprochenen Wörter; die Zahlen müssen ihm in codierter Form mitgeteilt werden.

Wir wollen uns in diesem Abschnitt mit verschiedenen Codierungen von Zahlen befassen. Dabei wollen wir bedenken, daß die Darstellung von Zahlen durch Ziffern auch nichts anderes als eine Codierung ist.

Geläufig ist Ihnen das Dezimalsystem zur ziffernmäßigen Darstellung von Zahlen. Diesen Code haben Sie in der Schule gelernt und handhaben ihn tagtäglich. Das Dezimalsystem ist aber durchaus nicht das einzig mögliche System zur Darstellung von Zahlen. Für den Mikroprozessor ist es sogar recht ungeeignet. Von den vielen möglichen Codierungsarten wird uns vorzugsweise das Dualsystem und das Sedezimalsystem beschäftigen.

Das Sedezimalsystem

Sie haben im vorangegangenen Abschnitt gesehen, daß zu jeder Anweisung für den Mikroprozessor vierstellige Adressen und zweistellige Ziffernpaare gehören. Diese zweistelligen Ziffernpaare enthalten die eigentliche Anweisung; die vierstellige Adresse stellt offenbar eine Art Numerierung der Anweisungen dar. Warum wir bei unseren Programmen statt bei der Adresse null oder eins bei z. B. der Adresse 18ØØ beginnen, soll im Augenblick nicht interessieren. Jedenfalls kommt, wie die Liste im Bild H 9.1 zeigt, nach der Adresse 18ØØ für das erste Ziffernpaar die Adresse 18Ø1 für das zweite Ziffernpaar, dann 18Ø2 usw. bis 18Ø9. Aber dann wird es seltsam: Es folgt keineswegs, wie man doch erwarten sollte, die Adresse 181Ø! Diese Adresse kommt

erst ein Stück später. Zunächst folgt auf die Adresse 18Ø9 die Adresse 18ØA (die Sie im Bild H 9.1 nicht ausgedruckt finden), dann die Adresse 18ØB, dann 18ØC usw. Was hat es mit den Buchstaben in einer Ziffernfolge auf sich, die doch in der Numerierung eine Zahl darstellen soll?

Um es vorwegzunehmen: Beim Umgang mit dem Mikroprozessor sollte man eigentlich das gewohnte und vertraute Dezimalsystem vergessen. Wir sagten es schon: Das Dezimalsystem ist für den Mikroprozessor ungeeignet. Auch mit dem System, in dem neben den gewohnten Ziffern Ø bis 9 die Buchstaben A bis F vorkommen, und das wir zur Befehlsnumerierung benutzen, kann der Mikroprozessor nichts anfangen. Aber bei diesem System besteht wenigstens ein direkter Zusammenhang zu dem System, mit dem der Mikroprozessor tatsächlich arbeitet. Dieser Zusammenhang wird uns später noch beschäftigen.

Zunächst wollen wir festhalten: Die Ziffernfolge 18ØØ steht im Zusammenhang mit dem Mikroprozessor nicht etwa für die Zahl Eintausendachthundert, sondern für die Zahl Sechstausendeinhundertvierundvierzig. Sonderbar, nicht wahr?

Nun, so sonderbar ist das eigentlich gar nicht, in einem anderen als dem Dezimalsystem zu denken und zu rechnen. Und um ein solch anderes System handelt es sich ja wohl, wenn die Ziffernfolge 18ØØ nicht Eintausendachthundert bedeutet.

Das **Dezimalsystem** hat seinen Namen nicht hauptsächlich daher, daß alle Rechnungen irgendwie mit der Zahl Zehn zu tun haben. (Das lateinische Wort *decem* bedeutet zehn.) Es hat seinen Namen daher, daß man sämtliche Zahlen durch geschickte Kombination von Zeichen aus einem Vorrat mit zehn unterschiedlichen Zeichen darstellen kann: Den Ziffern Ø, 1, 2, 3, 4, 5, 6, 7, 8 und 9.

Im Zusammenhang mit dem Mikroprozessor arbeiten wir mit dem **Sedezimalsystem**. Das lateinische Wort *sedecim* bedeutet sechzehn. Wir haben es also mit einem System zur Darstellung von Zahlen zu tun, bei dem man jede beliebige Zahl durch die Kombination von Zeichen aus einem Vorrat mit sechzehn unterschiedlichen Zeichen darstellen kann. – In der Umgangsfachsprache wird dieses System meist als **Hexadezimalsystem** bezeichnet. „Hexadezimal" ist allerdings eine ungeschickte Kombination eines griechischen mit einem lateinischen Wort. Wir bleiben in diesem Lehrgang bei der Bezeichnung sedezimal.

Bei der Darstellung von Zahlen im Sedezimalsystem ist man auf die sachlich nicht korrekte Idee gekommen, ebenfalls die zehn Zeichen des Dezimalsystems zu benutzen und für die zusätzlichen sechs Zeichen Buchstaben zu wählen. Es ergibt sich dann für das Sedezimalsystem ein Zeichenvorrat mit den sechzehn unterschiedlichen Zeichen Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E und F.

Diese Wahl der Zeichen hat zwar den Vorteil, daß man – abgesehen von den Zeichen A bis F – keine neuen Bedeutungen für Zahlzeichen zu lernen braucht; nachteilig ist, daß Zeichenfolgen wie 18ØØ nicht mehr eindeutig einer bestimmten Zahl zugeordnet werden können. Werden die Zeichen aus dem Zeichenvorrat des Dezimalsystems genommen, dann bedeutet 18ØØ die Zahl Eintausendachthundert. Stammen die Zeichen jedoch aus dem Zeichenvorrat des Sedezimalsystems, dann bedeutet 18ØØ die Zahl Sechstausendeinhundertvierundvierzig.

Solange man es mit Sicherheit nur mit dem Dezimalsystem zu tun hat, stört diese Doppeldeutigkeit nicht. Auch bei der Anzeige unseres Mikroprozessor-Systems ist ein Irrtum kaum möglich, denn wir wissen einfach, daß es sich um eine Sedezimal-Anzeige handelt. Wenn allerdings Zweifel möglich sind, dann kennzeichnet man eine Zeichenfolge durch den nachgestellten Buchstaben **H** (von **H**exadezimal), wenn es sich um eine Zahlendarstellung im Sedezimalsystem handelt:

1800**H** = Sechstausendeinhundertvierundvierzig.

Entstammen die Zeichen jedoch dem Zeichenvorrat des Dezimalsystems, dann verzichtet man auf eine besondere Kennzeichnung:

1800 = Eintausendachthundert.

Nun ist es natürlich interessant zu wissen, warum 1800**H** gerade die Zahl Sechstausendeinhundertvierundvierzig darstellt, oder wie Dreihundertsiebenunddreißig im Sedezimalsystem geschrieben wird.

Ohne Begründung läßt sich bereits jetzt vermuten, daß die Zahl Sechzehn im Sedezimalsystem eine ähnliche Rolle spielt wie die Zahl Zehn im Dezimalsystem. – Im Dezimalsystem lassen sich die Zahlen Null bis Neun durch jeweils ein Zeichen darstellen; die Zahl Zehn ist die erste, zu deren Darstellung man mehr als ein Zeichen braucht. Es ergibt sich folgende Zuordnung:

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

Das Prinzip ist durchsichtig: Bei der auf die Zahl Neun folgenden Zahl Zehn beginnt man einfach wieder mit dem Zeichen 0 aus dem Zeichenvorrat und kennzeichnet durch ein vorgestelltes Zeichen 1, daß beim Zählen sämtliche zehn unterschiedlichen Zeichen des Zeichenvorrats bereits vorher einmal verwendet wurden.

Die Zeichenfolge 43 bedeutet: Man ist beim Zählen, bei eins beginnend, bis drei gekommen. Das vorgestellte Zeichen 4 sagt jedoch, daß man vorher bereits viermal sämtliche zehn unterschiedlichen Zeichen 1, 2, 3 ... 9, 0 aus dem Zeichenvorrat verwendet hat.

43 bedeutet demnach: Viermal zehn plus dreimal eins. Und das ist gleich dreiundvierzig.

Die Zeichen haben unterschiedliche Wertigkeit, je nachdem, an welcher Stelle sie in einer Zeichenfolge stehen. Man nennt ein solches System deshalb **Stellenwertsystem**. Für die Bewertung gilt dabei – das haben Sie bereits in der Schule gelernt! – folgendes Schema:

4. Stelle	3. Stelle	2. Stelle	1. Stelle
Tausender	Hunderter	Zehner	Einer
mal 10^3	mal 10^2	mal 10^1	mal 10^0

Die Zeichenkombination 3 9 8 2 ist wie folgt zu lesen:

drei mal tausend + neun mal hundert + acht mal zehn + zwei mal eins

dreimal 10^3 + neunmal 10^2 + achtmal 10 + zweimal 10^0

Auch das Sedezimalsystem ist ein Stellenwertsystem. Es ist also im Prinzip genau so aufgebaut wie das Dezimalsystem. Der Unterschied liegt lediglich in der Anzahl der unterschiedlichen Zeichen im Zeichenvorrat. Für diese Zeichen ergibt sich folgende Zuordnung:

Zahl	Null	Eins	Zwei	Drei	Vier	Fünf	Sechs	Sieben	Acht
Zeichen	Ø	1	2	3	4	5	6	7	8

Zahl	Neun	Zehn	Elf	Zwölf	Dreizehn	Vierzehn	Fünfzehn	Sechzehn
Zeichen	9	A	B	C	D	E	F	1Ø

Bei diesem System ist die Zahl Sechzehn die erste, zu deren Darstellung man mehr als ein Zeichen braucht. Man beginnt nach der Zahl Fünfzehn auch hier einfach wieder mit dem Zeichen Ø und kennzeichnet durch ein vorgestelltes Zeichen 1, daß beim Zählen sämtliche sechzehn unterschiedlichen Zeichen des Zeichenvorrats bereits vorher einmal verwendet wurden.

Die Zeichenfolge 43 **H** bedeutet jetzt: Man ist beim Zählen, bei eins beginnend, bis drei gekommen. Das vorgestellte Zeichen 4 sagt jedoch, daß man vorher bereits viermal sämtliche sechzehn unterschiedlichen Zeichen 1, 2, 3 ... 9, A, B, C, D, E, F und Ø aus dem Zeichenvorrat verwendet hat.

43 **H** bedeutet demnach: Viermal sechzehn plus dreimal eins. Und das ist gleich siebenundsechzig.

Wir können folgende Gleichung anschreiben:

43 **H** = 67

Die Zeichen haben also wieder eine unterschiedliche Wertigkeit, je nachdem, an welcher Stelle sie in einer Zeichenfolge stehen. Wegen der beim Sedezimalsystem verwendeten sechzehn unterschiedlichen Zeichen ergibt sich diese Wertigkeit allerdings nicht aus Potenzen von zehn, sondern aus Potenzen von sechzehn. Das folgende Schema macht das deutlich:

4. Stelle	3. Stelle	2. Stelle	1. Stelle
Viertausendsechs-undneunziger	Zweihundertsechs-undfünfziger	Sechzehner	Einer
mal 16^3	mal 16^2	mal 16^1	mal 16^0

Dementsprechend ist die Zeichenkombination 3 9 8 2 **H** wie folgt zu lesen:

Dreimal viertausendsechsundneunzig + neunmal zweihundertsechsundfünfzig + achtmal sechzehn + zweimal eins oder

dreimal 16^3 + neunmal 16^2 + achtmal 16^1 + zweimal 16^0

Es ergibt sich vierzehntausendsiebenhundertzweiundzwanzig und es gilt:

3982 **H** = 14722

Das ist im Grunde nicht kompliziert. Umständlich erscheint nur die Umrechnung in die Darstellung im Dezimalsystem. Aber stellen Sie sich einmal vor, Sie hätten in der Schule niemals etwas vom Dezimalsystem gehört und immer nur im Sedezimalsystem gedacht! Sie kämen dann wohl kaum auf die Idee, eine im Ihnen gewohnten Sedezimalsystem dargestellte Zahl in die Dezimaldarstellung umzurechnen.

Wir haben bisher nur Zahlen betrachtet, bei deren Darstellung im Sedezimalsystem die gewohnten Zeichen Ø, 1, 2 ... 8, 9 vorkommen. Welche Zahl entspricht aber der Darstellung A7C**H**?

Hier hätten wir uns den Hinweis auf das Sedezimalsystem mit dem nachgestellten Buchstaben **H** eigentlich sparen können, denn die Zeichen A und C geben den deutlichen Hinweis, daß es sich nicht um eine Darstellung im Dezimalsystem handelt. Das ist richtig. Aber bedenken Sie bitte, daß außer dem Dezimalsystem mit zehn unterschiedlichen Zeichen und dem Sedezimalsystem mit sechzehn unterschiedlichen Zeichen auch beliebig andere Systeme mit beliebig vielen unterschiedlichen Zeichen möglich sind! Denkbar wäre etwa ein Quindezimalsystem (*quindecim* = fünfzehn) mit fünfzehn unterschiedlichen Zeichen Ø, 1, 2 ... 9, A, B, C, D und E, bei dem die Stellenwerte der Zeichen durch Potenzen von fünfzehn bestimmt sind. Dabei ergibt die Zeichenkombination A7C eine ganz andere Zahl als A7C**H**!

Für den Mathematiker sind solche Systeme zwar recht interessant; bei der Handhabung unseres Mikroprozessor-Systems arbeiten wir jedoch mit dem Sedezimalsystem und dürfen – falls Verwechslungen mit dem Dezimalsystem ausgeschlossen sind – die Kennzeichnung mit dem Buchstaben **H** auch weglassen.

Welche Zahl entspricht also der Darstellung A7C? – Es ist ganz einfach: Aus dem Schema der Stellenwerte für das Sedezimalsystem ergibt sich:

A mal zweihundertsechsundfünfzig + 7 mal sechzehn + C mal eins;
A**H** x 16^2 + 7**H** x 16^1 + C**H** x 16^0

Mit der Tabelle im Bild S17.1 finden wir den Übergang ins Dezimalsystem:

10 x 256 + 7 x 16 + 12 x 1 =
2684

Ergebnis: A7C**H** = zweitausendsechshundertvierundachtzig

Aufgabe S17.1

Wandeln Sie die folgenden Zeichenfolgen in die Darstellung im Dezimalsystem um und geben Sie jeweils die entsprechende Zahl an:

a) 420**H** b) 5AB**H** c) F4**H** d) Ø9D**H** e) FABC**H**

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü3.

Nun müssen wir noch überlegen, wie die Sache andersherum geht, wie man also eine vorgegebene Zahl im Sedezimalsystem darstellt oder wie man eine Zeichenfolge aus dem Dezimalsystem in eine entsprechende Zeichenfolge im Sedezimalsystem umwandelt.

Für diese Umwandlungen gibt es verschiedene, mehr oder weniger einfache Möglichkeiten. Rechnen muß man in jedem Fall. Wir wollen Ihnen hier eine Methode zeigen, die vielleicht nicht die einfachste ist, die aber den Vorteil hat, daß man sie nicht auswendig zu lernen braucht und sie folglich auch nicht vergessen kann.

Bei der Darstellung einer Zahl im Dezimalsystem geht man bekanntlich so vor, daß man die Zahl „zerlegt": Man fragt, wieviele Tausender in der Zahl enthalten sind, wieviele Hunderter dann noch übrig bleiben, wieviele Zehner der verbleibende Rest enthält und schreibt das, was dann noch übrig bleibt, als Einer an.

Zahl	Dezimal	Sedezimal
Null	Ø	Ø
Eins	1	1
Zwei	2	2
Drei	3	3
Vier	4	4
Fünf	5	5
Sechs	6	6
Sieben	7	7
Acht	8	8
Neun	9	9
Zehn	10	A
Elf	11	B
Zwölf	12	C
Dreizehn	13	D
Vierzehn	14	E
Fünfzehn	15	F

Bild S17.1
Tabelle der Zeichen für Zahlen im Dezimalsystem und im Sedezimalsystem.

Vielleicht ist Ihnen niemals richtig bewußt geworden, daß Sie es dauernd so machen. Die Art des Aussprechens einer Zahl in unserer Sprache ist dabei ein entscheidendes Hilfsmittel.

Bei der Darstellung einer Zahl im Sedezimalsystem geht man genauso vor. Dabei besteht lediglich die Schwierigkeit, daß uns die offenbar dezimal orientierte Sprache keine Hilfe leisten kann. Das braucht Sie aber nicht zu erschrecken, denn Rechnen ist kein Zauberkunststück.

Wir betrachten willkürlich die Zahl Dreihundertachtunddreißig und zeigen zunächst noch einmal, wie diese Zahl im Dezimalsystem dargestellt wird:

Anzahl der **Hunderter**: Dreihundertachtunddreißig geteilt durch **hundert** ist drei.
Das Dezimalzeichen für drei wird in der Hunderterstelle angeschrieben: **3..**
Der verbleibende Rest von achtunddreißig wird weiterverarbeitet.

Anzahl der **Zehner**: Achtunddreißig geteilt durch **zehn** ist drei.
Das Dezimalzeichen für drei wird in der Zehnerstelle angeschrieben: **33.**
Der verbleibende Rest von acht wird weiterverarbeitet.

Anzahl der **Einer**: Acht geteilt durch **eins** ist acht.
Das Dezimalzeichen für acht wird in der Einerstelle angeschrieben: **338**

Weil Sie es so zu machen gewohnt sind, erscheint Ihnen diese Überlegung trivial. Aber bitte: Gerade so trivial ist die Darstellung einer Zahl im Sedezimalsystem. Sie müssen nun allerdings wegen der anderen Stellenwertigkeit nicht durch hundert (10^2), zehn (10^1) und eins (10^0) dividieren, sondern durch zweihundertsechsundfünfzig (16^2), sechzehn (16^1) und eins (16^0). Für die Zahl Dreihundertachtunddreißig geht das so:

Anzahl der **Zweihundertsechsundfünfziger**: Dreihundertachtunddreißig geteilt durch **zweihundertsechsundfünfzig** ist eins.

Das Sedezimalzeichen für eins wird in der Zweihundertsechsundfünfzigerstelle angeschrieben: **1..**
Der verbleibende Rest von zweiundachtzig wird weiterverarbeitet.

Anzahl der **Sechzehner**: Zweiundachtzig geteilt durch **sechzehn** ist fünf.
Das Sedezimalzeichen für fünf wird in der Sechzehnerstelle angeschrieben: **15.**
Der verbleibende Rest von zwei wird weiterverarbeitet.

Anzahl der **Einer**: Zwei geteilt durch **eins** ist zwei.
Das Sedezimalzeichen für zwei wird in der Einerstelle angeschrieben: **152**
Nach der Teilung durch eins bleibt natürlich kein Rest.

Ergebnis: Dreihundertachtunddreißig = 338 = 152**H**

Eigentlich waren wir hier leichtsinnig und haben ohne Überlegung vorausgesetzt, daß die Zahl Dreihundertachtunddreißig durch eine dreistellige Folge von Sedezimalzeichen dargestellt wird. Was wäre geschehen, wenn wir nach diesem Schema die Zahl Viertausendsechshundertundsechzig sedezimal darstellen wollten?

Wir würden gleich beim ersten Schritt aufmerksam, daß da etwas nicht stimmt:

Anzahl der **Zweihundertsechsundfünfziger**: Viertausendsechshundertundsechzig geteilt durch **zweihundertsechsundfünfzig** ist achtzehn.
(Der Rest interessiert noch nicht.)

Für achtzehn gibt es kein Sedezimalzeichen. Also ist in Viertausendsechshundertundsechzig noch mindestens ein **Viertausendsechsundneunziger** (16^3) enthalten, der an der (von rechts gezählt) vierten Stelle eingetragen werden muß (Seite S16).

Zweckmäßig schaut man sich die darzustellende Zahl daraufhin an, welche höchste Potenz von sechzehn in ihr sicher gerade nicht mehr enthalten ist und teilt dann durch eine um eins kleinere Potenz von sechzehn.

Bei der Zahl Viertausendsechshundertundsechzig wissen wir nun schon Bescheid und fangen gleich mit der Rechnung an. Das Schema haben Sie durchschaut; wir können uns also kürzer fassen:

$466Ø : 16^3 = 466Ø : 4Ø96 = 1$ (Rest 564)	**1...**
$564 : 16^2 = 564 : 256 = 2$ (Rest 52)	**12..**
$52 : 16^1 = 52 : 16 = 3$ (Rest 4)	**123.**
$4 : 16^0 = 4 : 1 = 4$	**1234**

Ergebnis: Viertausendsechshundertundsechzig = 1234**H**

Wir wollen noch ein Beispiel zusammen rechnen und die Zeichenfolge 2572 in die sedezimale Darstellung umwandeln.

In 2572 ist $16^3 = 4Ø96$ nicht enthalten.

$2572 : 16^2 = 2672 : 256 = 1Ø$ (Rest 12)

Diese zehn Zweihundertsechsundfünfziger müssen wir unter Verwendung des Sedezimalzeichens für zehn (Bild S17.1) an der (von rechts gezählt) dritten Stelle eintragen: **A..**

Anschließend wird geprüft, wieviele Sechzehner im verbleibenden Rest von zwölf enthalten sind:

$12 : 16^1 = 12 : 16 = Ø$ (Rest 12)

Es ist also nach Abzug der zehn Zweihundertsechsundfünfziger in 2572 kein Sechzehner vorhanden: **AØ.**

Jetzt werden die im Rest verbleibenden Einer angeschrieben:

$12 : 16^0 = 12 : 1 = 12$

Es sind zwölf Einer vorhanden, die unter Verwendung des Sedezimalzeichens für zwölf (Bild S17.1) an der von rechts gezählt ersten Stelle angeschrieben werden: **AØC**

Ergebnis: 2572 = AØC**H**

Aufgabe S 20.1

Stellen Sie die folgenden Zahlen sedezimal dar bzw. wandeln Sie die dezimal dargestellten Zahlen in eine sedezimale Darstellung um:

a) Einhundertsechsundvierzig

b) Eintausendeinhundertsiebenundsechzig

c) 3Ø21

d) 385Ø

e) 51783

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 3.

Ehe wir diesen Abschnitt abschließen, soll noch erwähnt werden, daß bei Mikrocomputern manchmal statt der sedezimalen Darstellung von Zahlen die oktale Darstellung gewählt wird, also das **Oktalsystem**. Wir beschäftigen uns hier mit diesem System nicht weiter. Sollte es Ihnen je begegnen, dann fällt Ihnen der Umgang damit nicht schwer. Die Überlegungen, die der Umwandlung vom und zum Dezimalsystem zugrunde liegen, entsprechen denen, die wir hier für das Sedezimalsystem angestellt haben. Mit einem Unterschied: Im Oktalsystem wird mit einem Vorrat von acht unterschiedlichen Zeichen (Ø, 1, 2 ... 5, 6, 7) gearbeitet. Dementsprechend werden die Stellenwerte jeweils durch Potenzen von acht bestimmt.

Wir fassen zusammen:
Beim Umgang mit Mikroprozessoren wird meist statt mit der dezimalen Zahlendarstellung mit dem **Sedezimalsystem** (und manchmal mit dem Oktalsystem) gerechnet.

Das Sedezimalsystem und das Oktalsystem sind wie das Dezimalsystem Stellenwertsysteme. Entsprechend der Anzahl der im Zeichenvorrat vorhandenen unterschiedlichen Zeichen sind die Stellen, an denen die Zeichen in einer Zeichenkombination erscheinen, mit Potenzen der Anzahl der verfügbaren unterschiedlichen Zeichen bewertet.

Zum Schluß noch ein Hinweis:
Haben Sie auch Schwierigkeiten, sich die Bedeutung der Zeichen A bis F zu merken? Es gibt da eine Eselsbrücke, die Sie sicher nicht vergessen:

C = **C**wölf (Entschuldigung! Aber es ist ja eine Eselsbrücke!)
D = **D**reizehn
F = **F**ünfzehn

Wenn Sie diese drei Zeichen als Gerüst verwenden, dann ist es leicht, auch die Bedeutung der Zeichen A, B und E (Bild S 17.1) zu behalten.

Die Verknüpfung binärer Signale mit dem Mikroprozessor

Im Abschnitt „Was kann ein Mikroprozessor“ haben Sie gelesen, daß der Mikroprozessor ein digitales Bauelement ist und daß seine Leistungsfähigkeit darin liegt, komplexe digitale Aufgaben zu lösen. Das Schwergewicht der Entwicklung liegt dabei bei der Software, also beim Erstellen des Programms. Die Denkweise beim Umgang mit dem Mikroprozessor ist jedoch immer digital. Besonders deutlich wird das daran, daß Mikroprozessoren eigens dafür eingerichtet sind, digitale Grundverknüpfungen vorzunehmen. Sie sollen deshalb die ersten praktischen Erfahrungen mit dem Mikroprozessor an solchen Grundverknüpfungen machen. Dabei können Sie sich zugleich weiter mit der Handhabung Ihres Systems vertraut machen.

Was sind binäre Signale?

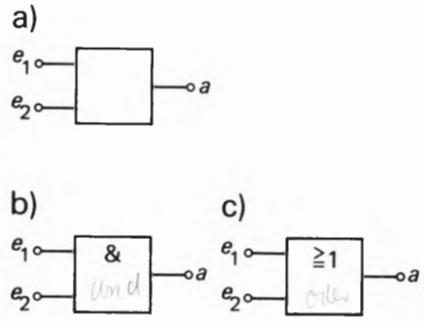
Unter Signalen verstehen wir physikalische Größen, durch deren Wert eine Information dargestellt werden kann. Die Digitaltechnik beschäftigt sich mit der Verarbeitung solcher Signale, die **nur zwei unterschiedliche Werte** annehmen können, also mit **binären** Signalen. Diese Werte bezeichnen wir mit den Ziffern Ø und 1. Als physikalische Größe, die den Wert einer Information darstellt, also als Signalparameter, ist für uns in diesem Zusammenhang nur die elektrische Spannung interessant.

Wenn wir uns auf den Wert des Signalparameters Spannung beziehen, dann können wir statt der Ziffer 1 auch den Buchstaben H für den mehr positiven Wert der Spannung benutzen und statt der Ziffer Ø den Buchstaben L für den weniger positiven Wert der Spannung. Dabei ist H die Abkürzung des englischen Wortes *HIGH* (sprich: hei), das „hoch“ bedeutet, und L die Abkürzung des englischen Wortes *LOW* (sprich: lou), das „niedrig“ bedeutet.

Von den möglichen Arten der Verknüpfung solcher binärer Signale interessieren uns vor allem die UND- und die ODER-Verknüpfung und die sogenannte Exclusiv-ODER-Verknüpfung, die wir Ihnen im Laufe dieses Abschnitts noch vorstellen und die in der Rechnertechnik eine wichtige Rolle spielt.

Auf der Seite H2 haben wir angedeutet, daß man mit einem Mikroprozessor-System bei entsprechender Programmierung auch die einfache Verknüpfung von zwei binären Signalen vornehmen kann. Praktisch ist das natürlich ein Unding: Mit einem einzigen kleinen TTL-Schaltglied erreicht man das gleiche viel billiger. Trotzdem sollen Sie sich die Grundverknüpfungen mit Hilfe Ihres Mikroprozessor-Systems ansehen.

Im Bild S22.1a haben wir die Grundform des Schaltzeichens für ein Verknüpfungsglied dargestellt. Diese Grundform sagt aus, daß die beiden digitalen Eingangssignale e_1 und e_2 zu einem Ausgangssignal a verknüpft werden. Der Wert des Ausgangssignals ist eine Funktion der Werte der beiden Eingangssignale und außerdem natürlich abhängig von der Art der Verknüpfung, die in dem Schaltbild vorgenommen wird, welches das Schaltzeichen darstellt.



S 22

Bild S 22.1
a) Grundform des Schaltzeichens für ein Verknüpfungsglied.
b) Schaltzeichen für die UND-Verknüpfung von zwei Eingangssignalen.
c) Schaltzeichen für die ODER-Verknüpfung von zwei Eingangssignalen.

Über die Art der vorgenommenen Verknüpfung sagt das Schaltzeichen im Bild S 22.1a zunächst nichts aus. Es deutet lediglich an, daß zwischen den Werten der variablen Eingangssignale e_1 und e_2 und dem Wert des ebenfalls variablen Ausgangssignals a ein irgendwie gearteter, funktionaler Zusammenhang besteht. Solche variablen Eingangs- und Ausgangssignale werden einfach als Variable bezeichnet, die bei digitalen Schaltgliedern entweder die Werte 0 oder 1 annehmen können. Den zunächst noch unbestimmten, funktionalen Zusammenhang zwischen den Eingangsvariablen und der Ausgangsvariablen kann man in der Sprache der Mathematik so ausdrücken:

$$a = f(e_1, e_2)$$

(Lies: a gleich ef von e eins und e zwei.)

Die Variable a ist eine Funktion der Variablen e_1 und e_2.

In den Teilbildern b und c sind Schaltzeichen dargestellt, die den funktionalen Zusammenhang zwischen der Variablen a und den Variablen e_1 und e_2 genau definieren. Das Teilbild b stellt eine **UND-Verknüpfung** der Eingangssignale dar, das Teilbild c eine **ODER-Verknüpfung**.

Der Mikroprozessor verknüpft zwei binäre Signale

Sie wissen bereits, daß ein Mikroprozessor-System erst dann arbeiten kann, wenn es entsprechende Anweisungen in Form eines Programms erhält (Seite H2). Ein solches Programm besteht aus Befehlen, die das Mikroprozessor-System verstehen kann.

Sie können einem solchen System nun zwar die Anweisung in Form von Befehlen geben, es solle bellen wie ein Hund. Diese Anweisung wird das System jedoch vermutlich nicht ausführen, da die Ausführung der zugehörigen Befehle vom Konstrukteur des Systems nicht vorgesehen wurde. Solche Befehle gehören nicht zum **Befehlssatz** des Mikroprozessors. Ausgeführt werden kann nur die beschränkte Anzahl von Befehlen, die im Befehlssatz enthalten ist. Andererseits ist man bei der Konstruktion eines Mikroprozessors natürlich bestrebt, den Befehlssatz so auszulegen, daß durch die geschickte Kombination der möglichen Befehle auch komplexe Anweisungen ausgeführt werden können. So betrachtet ist die Anweisung für das Bellen vielleicht doch nicht ganz absurd.

Der in unserem System verwendete Mikroprozessor hat einen Befehlssatz, der es uns recht einfach macht, die Verknüpfung binärer Signale vorzunehmen. Wie das im einzelnen geschieht, werden wir Ihnen im Laufe des Lehrgangs noch erläutern. Zunächst begnügen wir uns damit, die Grundzüge des Ablaufplans für ein entsprechendes Programm zu entwerfen.

Beim Entwurf eines solchen Ablaufplans muß zunächst genau festgelegt werden, was eigentlich erreicht werden soll. Die Absichtserklärung: 'Wir wollen die Funktion eines digitalen Verknüpfungsglieds nachbilden' ist bei weitem nicht genau genug. Es muß schon genau festgelegt werden, wie das geschehen soll.

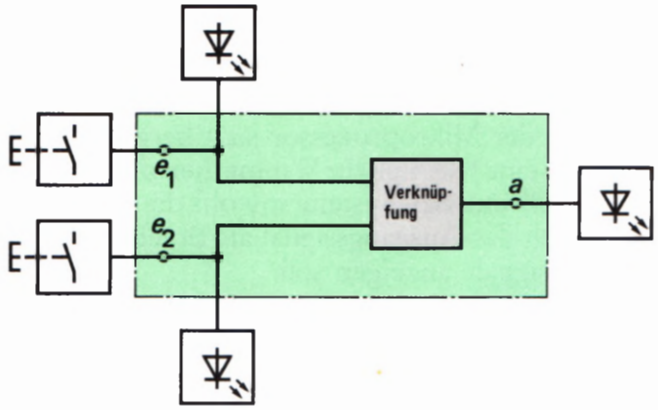


Bild S23.1
Die Verknüpfung von zwei über Tastschalter eingegebenen Eingangssignalen zu einem Ausgangssignal soll über das hier grün angedeutete Mikroprozessor-System vorgenommen werden. Die Signalwerte 1 sollen optisch angezeigt werden.

Betrachten Sie das Bild S23.1! Die grün unterlegte Fläche stellt das eigentliche Mikroprozessor-System dar, dem über zwei Tastschalter die Eingangssignale e_1 und e_2 zugeführt werden. Das System soll die eingegebenen Signale mit Leuchtanzeigen quittieren.

Innerhalb des Systems werden die beiden Eingangssignale miteinander verknüpft (grau unterlegtes Feld). Den Wert des resultierenden Ausgangssignals soll das System ebenfalls mit einer Leuchtanzeige melden. Die Leuchtanzeigen sollen ein 1-Signal durch Leuchten darstellen; bei Ø-Signalen sollen die Anzeigen dunkel bleiben.

Überlegen wir, welche Tätigkeiten das Mikroprozessor-System – von einem Programm gesteuert – zur Lösung der gestellten Aufgabe nacheinander ausüben muß.

1. Zunächst wird festgestellt, ob überhaupt eine Taste für ein Eingangssignal betätigt ist, ob also ein vom Wert Ø abweichendes Eingangssignal vorliegt. Solange das nicht der Fall ist, soll der Mikroprozessor nichts unternehmen.

   (Wenn Sie sich in der Digitaltechnik auskennen, dann werden Sie feststellen, daß wir damit die Wirksamkeit unseres Programms entscheidend einschränken, weil wir die Kombination von zwei Ø-Eingangssignalen nicht berücksichtigen. Das wollen wir aber bei diesem einfachen Programm bewußt in Kauf nehmen.)

   Die Eingangssignale werden abgefragt.

2. Wenn der Mikroprozessor festgestellt hat, daß mindestens eines der Eingangssignale vom Wert Ø abweicht, dann muß sich das System die festgestellte Kombination der Werte der Eingangssignale e_1 und e_2 merken.

   Diese Tätigkeit entspricht genau der, die Sie selbst in diesem Fall auch ausüben würden: Sie schreiben die Kombination der nachher miteinander zu verknüpfenden Eingangssignale zunächst einmal auf, z. B. $e_1 = Ø$, $e_2 = 1$. – Beim Mikroprozessor sagt man fachmännisch:

   Die Werte der Eingangssignale werden gespeichert.

3. Die folgende Anweisung ist die entscheidende in unserem Programm:

   Die gespeicherten Eingangssignale werden verknüpft.

S

24

4. Das Ergebnis der Verknüpfung wird als anschließend auszugebendes Ausgangssignal – ebenso wie vorher die beiden Eingangssignale – gespeichert.
5. Eigentlich hat der Mikroprozessor jetzt bereits seine Schuldigkeit getan – bis auf eine Kleinigkeit: Wir merken nichts davon. Es wurde ja verlangt, daß uns das System sowohl die Werte der Eingangssignale als auch das Ausgangssignal als Ergebnis der Verknüpfung durch Leuchtsignale anzeigen soll.

   Die Werte der Eingangssignale und des Ausgangssignals werden angezeigt.
6. Der bis hierher beschriebene Vorgang erfüllt zwar die von uns gestellte Aufgabe; er tut das aber nur ein einziges Mal. Wenn die Eingangssignale ihre Werte wechseln, dann kann das System dies zunächst nicht feststellen. Es muß also den Punkt 1 immer wieder ausführen und dann damit rechnen, andere Werte der Eingangssignale und damit auch einen anderen Wert des Ausgangssignals anzeigen zu müssen. Es muß die Möglichkeit zur Anzeige anderer Signalwerte geschaffen werden. Dazu muß die Anzeige gelöscht werden. Damit man aber überhaupt etwas sehen kann, soll die einmal sichtbar gemachte Anzeige wenigstens einen Augenblick lang unverändert stehen bleiben.

   Der Mikroprozessor wartet einen Augenblick lang.
7. Die Anzeige wird gelöscht.
8. Damit der Mikroprozessor auch andere Signal-Kombinationen erfassen und anzeigen kann, muß das jetzt beschriebene Programm immer von neuem ablaufen.

   Das Programm muß wieder vom Punkt 1 an ablaufen.

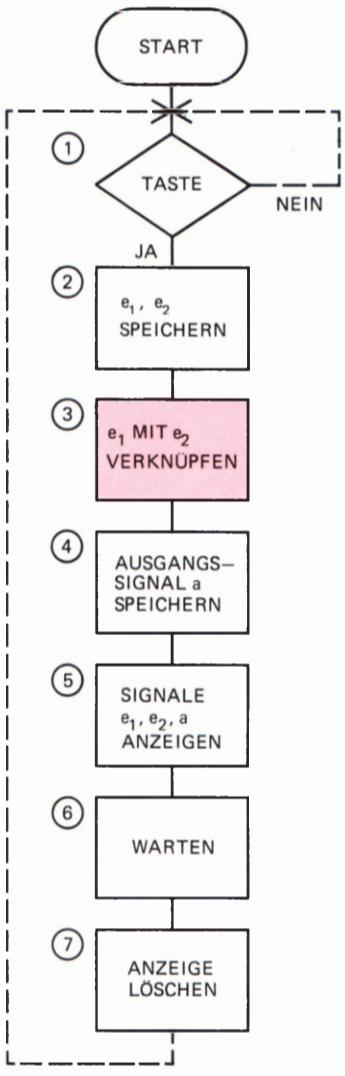


Bild S24.1
Programmablaufplan für die Verknüpfung von zwei Eingangssignalen mit dem Mikroprozessor.

Diese Auflistung der vom Mikroprozessor verlangten Tätigkeiten mag Ihnen geradezu pedantisch vorkommen. Recht haben Sie! Aber wir müssen uns damit abfinden: Der Mikroprozessor ist ein geistloser Sklave, der mit unglaublicher Präzision und Geschwindigkeit das, und genau nur das tut, was ihm durch Anweisungen vorgeschrieben wird. Wenn irgend eine Anweisung vergessen oder falsch gegeben wird, dann tut er wieder genau nur das (und nichts anderes), was ihm seine Anweisungen sagen – und „läuft in die Wüste". Der Programmierer guckt ihn böse an und hat doch an der Misere selbst die Schuld: Er hat eben mit falschen oder unvollständigen Anweisungen falsch programmiert.

Unsere Auflistung ist jedoch sicher vollständig. Wenn wir's so machen, dann sagen wir dem Mikroprozessor unmißverständlich und genau, was er zu tun hat. Ist Ihnen aufgefallen, daß Sie selbst – abgesehen vielleicht von den Anweisungen „Anzeigen" – bei der Analyse einer Digitalschaltung ähnlich vorgehen würden?

Die Liste unserer Anweisungen ist natürlich recht unpraktisch und unübersichtlich. Sie entspricht etwa der angefangenen Tagesablauf-Liste auf der Seite S2. Im Bild S24.1 sehen Sie die gleichen Anweisungen deshalb noch einmal in Form eines fachgerechten Programmablaufplans, bei dem der gestrichelt eingetragene Rückweg vom Ende zum Start der oben angegebenen achten Anweisung entspricht.

Als Tasten für die Eingabe der Eingangssignale werden in den folgenden Versuchen zwei Tasten des Tastenfeldes Ihres Systems benutzt. Aus technischen Gründen haben wir dazu die beiden unteren Tasten der rechten Tasten-Spalte mit den Bezeichnungen 3 und 7 ausgewählt. Dabei soll die Taste 7 das Eingangssignal e_1 beeinflussen und die Taste 3 das Eingangssignal e_2. Wie die Eingangssignale und das Ausgangssignal angezeigt werden, macht nachher der Versuch sehr deutlich. Wir verwenden dafür einzelne Segmente der Sieben-Segment-Anzeigen.

Aus unseren Überlegungen auf der Seite H 8 geht bereits hervor, daß Mikroprozessoren nichts anderes als einfache Zahlen verarbeiten können. Der soeben erstellte Programmablaufplan ist zwar zur Festlegung der Programmier-Aufgabe äußerst nützlich (bei komplexen Programmier-Aufgaben ist er schier unerläßlich); der Mikroprozessor selbst kann mit einem solchen Plan jedoch überhaupt nichts anfangen. Die in diesen Plänen verwendete menschliche Sprache ist ihm einfach unverständlich. Wir müssen dem Mikroprozessor seine Anweisungen in einer ihm verständlichen Sprache geben. Diese Sprache nennt man **Maschinensprache**, und die besteht – wie gesagt – aus einfachen Zahlen.

Das Ziel unseres Lehrgangs ist es unter anderem, Ihnen diese Maschinensprache bekannt zu machen. Das geht natürlich nicht in der kurzen Zeit, in der Sie das Verknüpfungsprogramm „laufen" sehen möchten. Wir geben Ihnen deshalb im Bild S 25.1 die fertige Übersetzung des Programmablaufplans in die Maschinensprache an. Verraten können wir Ihnen jedoch bereits jetzt, daß jeder Anweisung im Programmablaufplan ganz bestimmte Zahlenkombinationen in der Auflistung des Programms in der Maschinensprache entsprechen. Die eingekreisten Nummern in dieser Auflistung machen das deutlich. Sie gehören jeweils zu den gleichen Nummern beim Programmablaufplan.

Vergleichen Sie das Bild S 25.1 mit dem Bild H 9.1! Sie sehen, daß die Anordnung in beiden Listen die gleiche ist. Wenn Sie Ihrem System also das in Maschinensprache abgefaßte Verknüpfungsprogramm mitteilen wollen, dann können Sie genau nach dem Schema vorgehen, daß wir beim Versuch H 9.1 beschrieben haben.

	Adresse	1. Eingabe	2. Eingabe
①	18ØØ	3E	FE
	18Ø2	D3	Ø2
	18Ø4	DB	ØØ
	18Ø6	2F	
	18Ø7	E6	Ø3
	18Ø9	28	F5
②	18ØB	47	
	18ØC	CB	2Ø
③	18ØE	AØ	
④	18ØF	E6	Ø2
	1811	CB	2Ø
	1813	BØ	
	1814	F6	CØ
⑤	1816	D3	Ø2
	1818	3E	Ø2
	181A	D3	Ø1
⑥	181C	Ø6	C9
	181E	1Ø	FE
⑦	182Ø	AF	
	1821	D3	Ø1
⑧	1823	18	DB

Bild S 25.1
Auflistung der in die Maschinensprache übersetzten Anweisungen aus dem vorhergehenden Bild.

Versuch S 25.1

Ein Verknüpfungsprogramm für den Mikroprozessor

Betätigen Sie auf Ihrem System die Taste RS (RESET), um die System-Meldung in die Anzeige zu bringen. (Wenn Sie Ihr System für diesen Versuch neu einschalten, dann erübrigt sich die Betätigung der RS-Taste; nach dem Verbinden des Systems mit der Stromversorgung erhalten Sie automatisch die System-Meldung.)

Nach der System-Meldung wird zunächst die Anfangsadresse 18ØØ des Programms (links oben in der Liste) eingetastet. Wir geben Ihnen hier noch einmal die Reihenfolge der zu betätigenden Tasten an:

ADDR, 1, 8, 0, 0

Anschließend können Sie die Daten für die einzelnen Befehle des Programms in Form von Ziffernpaaren eingeben und vor jedem folgenden

Ziffernpaar die Adresse mit der Taste + um eins erhöhen. Machen Sie sich dabei bitte klar, daß in der Liste jede Zeile einem Befehl entspricht. Sie erkennen, daß zu den meisten Befehlen unseres Programms zwei Ziffernpaare gehören. Einzelne Befehle werden von nur einem Ziffernpaar gebildet. (Grundsätzlich werden die Befehle des in unserem System verwendeten Mikroprozessors aus einem bis zu vier Ziffernpaaren gebildet.)

Betätigen Sie also der Reihe nach folgende Tasten und überzeugen Sie sich jeweils nach dem Betätigen der Taste + davon, ob an den linken vier Anzeigestellen die richtige Adresse für das nachfolgende Ziffernpaar erscheint:

DATA	(Mitteilung an das System, daß jetzt Daten folgen)
3, E	(Adresse 18ØØ, erstes Ziffernpaar)
+, F, E	(Adresse auf 18Ø1 erhöhen, zweites Ziffernpaar)
+, D, 3	(Adresse auf 18Ø2 erhöhen, drittes Ziffernpaar)
u.s.w. bis	
+, 1, 8	(Adresse auf 1823 erhöhen, Ziffernpaar)
+, D, B	(Adresse auf 1824 erhöhen, letztes Ziffernpaar)
(RS)	(kann auch entfallen)

Lesen Sie bitte auf den Seiten H10 und H11 noch einmal nach, wie Sie irrtümliche Eingaben (auch nach dem Betätigen der Taste +) korrigieren können!

Nach der Eingabe des vollständigen Programms können Sie die Ziffernpaare noch einmal kontrollieren. Wählen Sie dazu die Adresse des ersten Ziffernpaares des ersten Befehls an:

ADDR	(Mitteilung an das System, daß eine Adresse folgt)
1, 8, 0, 0	(Adresse des ersten Befehls-Ziffernpaares)

Neben der Adresse 18ØØ sehen Sie an den rechten beiden Anzeigestellen (wenn Sie vorher richtig eingegeben haben) das Ziffernpaar 3E. Erhöhen Sie jetzt die Adressen durch Betätigen der Taste + jeweils um eins und schauen Sie nach, ob bei den in der Anzeige erscheinenden Adressen die richtigen Ziffernpaare stehen. Wenn Sie ein falsches Ziffernpaar entdecken, dann teilen Sie dem System mit der Taste DATA mit, daß Sie die Daten ändern wollen, und korrigieren Sie das Ziffernpaar. (Die Dezimalpunkte in der Anzeige melden, ob eine Eingabe im Adressenfeld oder im Datenfeld wirksam wird.) – Nach einer solchen Korrektur können Sie mit der Taste + weiter kontrollieren; Sie brauchen also vorher nicht wieder die Taste ADDR zu betätigen.

Ehe Sie das nun „geladene" Programm starten, soll für einen Vorversuch noch eine kleine, vorläufige Änderung des Programms vorgenommen werden. Betätigen Sie dazu folgende Tasten:

ADDR, 1, 8, 2, 3, SBR, RS

Die Taste SBR finden Sie als (von links gezählt) vierte in der oberen Reihe des Tastenfeldes. SBR ist die Abkürzung für **S***et* **BR***eakpoint*, und das bedeutet: Setze einen Haltpunkt. Die Anzeige reagiert auf die Betätigung dieser Taste mit Dezimalpunkten an allen sechs Stellen. Sie haben damit bewirkt, daß das Programm nach einmaligem Ablauf abgebrochen wird. Es wird also nicht mit einem neuen Ablauf fortgesetzt.

Jetzt können Sie das Programm starten (vgl. Seiten H8 und H12):

ADDR, 1, 8, 0, 0, GO

Die Anzeige verlöscht, und ihr programmiertes System wartet auf die Eingabe des ersten Eingangssignals mit einer der beiden Tasten 3 oder 7 (Anweisung 1 im Bild S24).

Betätigen Sie die Taste 7 auf dem Tastenfeld! Sie liefern damit an das Verknüpfungsglied „Mikroprozessor-System" die Eingangssignal-Kombination $e_1 = 1$; $e_2 = \emptyset$. Was geschieht?

Die Reaktion Ihres Systems ist sehr enttäuschend: Sofort nach der Eingabe des Signals $e_1 = 1$ erscheint die Anzeige 1800 3E, also das erste Ziffernpaar des Programms und die zugehörige Adresse. Scheinbar ist also überhaupt nichts geschehen.

Sehen Sie sich bitte noch einmal den Programmablaufplan im Bild S24.1 an! Mit der vorhergehenden Betätigung der Taste SBR haben Sie Ihrem System die Anweisung gegeben, nach einmaligem Abarbeiten des Verknüpfungsprogramms seine Aktivität abzubrechen. Und genau das hat Ihr System auch getan.

Zunächst hat Ihr System nach dem Start des Programms in ununterbrochener Folge immer wieder die Anweisung 1 ausgeführt, solange keine der Tasten 7 oder 3 betätigt wurde. Nach dem Betätigen der Taste 7 hat das System die Anweisung 1 verlassen und die Eingangssignal-Kombination $e_1 = 1$; $e_2 = \emptyset$ gespeichert und anschließend diese beiden Eingangssignale zu einem Ausgangssignal a verknüpft. Auch dieses Signal wurde gespeichert und dann wurden die Werte der drei Signale e_1, e_2 und a in die Anzeige gebracht. Die anschließende Anweisung „WARTEN" hat diese Signal-Kombination zwar einen Augenblick lang in der Anzeige stehen lassen, aber dieser Augenblick war viel zu kurz, als daß Sie ihn hätten wahrnehmen können. Danach wurde die Anzeige wieder gelöscht – und dann war der Ablauf des Programms beendet: Das System hat sich mit 1800 3E in der Anzeige gemeldet.

In Ihrem System ist also sehr wohl etwas passiert. Aber dieses Geschehen ist in einem winzigen Bruchteil einer Sekunde abgelaufen – viel zu schnell, als daß Sie es hätten wahrnehmen können.

Sie sehen, daß unser Programm bei einem einmaligen Ablauf zu nichts nütze ist. Das ist der Grund, aus dem wir als letzte Anweisung in unserem Programm einen Rückweg zum Anfang des Programms programmiert haben. Das Programm wird dann einige tausendmal in jeder Sekunde abgearbeitet, und wenn das geschieht, dann werden Sie die erwartete Reaktion Ihres Systems erkennen können. Vor allem werden Sie feststellen, daß Ihr Programm jeden Wechsel der Werte der Eingangssignale registriert und zusammen mit dem Wert des aus der Verknüpfung der Eingangssignale gebildeten Ausgangssignals richtig zur Anzeige bringt. Der Ablauf des Programms erfolgt jedesmal so schnell, daß es Ihnen sicher nicht gelingt, dem System eine Tastenbetätigung zu unterschlagen.

Betätigen Sie folgende Tasten:

RS, CBR, MONI, RS

Die Taste CBR finden Sie auf Ihrem Tastenfeld in der zweiten Reihe an der (von links gezählt) vierten Stelle unter der Taste SBR. CBR ist die

Abkürzung des englischen **C***lear* **BR***eakpoint:* Lösche den (vorher gesetzten) Haltpunkt. – Was es mit der Taste MONI (erste Taste in der zweiten Reihe des Tastenfeldes; Abkürzung für **MONI**tor) auf sich hat, braucht uns an dieser Stelle noch nicht zu interessieren.

Starten Sie das Programm von neuem bei der Adresse 18ØØ:

ADDR, 1, 8, 0, 0, GO

Die Anzeige verlöscht und das Programm wartet auf die Eingabe von Eingangssignalen mit den Tasten 7 und 3.

Betätigen Sie versuchsweise diese beiden Tasten einzeln und gemeinsam und beobachten Sie die Anzeige! – Das Bild S28.1 zeigt, welche Anzeigen die Werte der Eingangssignale und den Wert des Ausgangssignals melden.

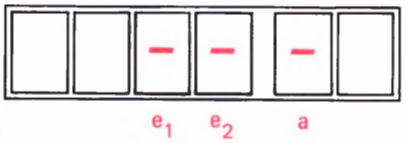


Bild S28.1
Zuordnung der Signalanzeigen zu den Werten der Eingangssignale und des Ausgangssignals beim Verknüpfungsprogramm.

Auch das planlose Spielen mit den Eingangssignal-Tasten macht bereits deutlich, daß $a = f(e_1, e_2)$ ist, daß also der Wert des Ausgangssignals eine Funktion der Kombination der Eingangssignale ist. Wenn Sie sich in der Digitaltechnik ein wenig auskennen, dann sehen Sie sofort, welche Art der Verknüpfung wir programmiert haben. Trotzdem sollen Sie sich in den folgenden Versuchen kurz mit den Verknüpfungsmöglichkeiten des Mikroprozessors vertraut machen.

Die UND-Verknüpfung

Die UND-Verknüpfung haben Sie im Prinzip beim Spielen mit dem vorangegangenen Versuch bereits kennengelernt. Wir wollen hier noch etwas systematischer untersuchen, was es mit dieser Verknüpfung auf sich hat.

a)

e_1	e_2	a
Ø	Ø	0
Ø	1	0
1	Ø	0
1	1	1

b)

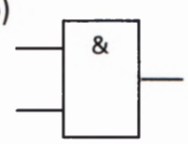


Bild S28.2
a) Zu Aufgabe S28.1: Ergänzen Sie bitte diese Funktionstabelle entsprechend dem Ergebnis des Versuchs S25.1.
b) Schaltzeichen für ein UND-Verknüpfungsglied mit zwei Eingängen.

Aufgabe S28.1

Ergänzen Sie die Funktionstabelle im Bild S28.2a durch einen Versuch mit dem entsprechend dem Bild S25.1 programmierten Mikroprozessor! Die Taste 7 liefert das Signal e_1 , die Taste 3 liefert das Signal e_2. (Betätigte Taste bzw. leuchtende Anzeige entspricht einem 1-Signal.)

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü4.

Die Tabelle zeigt, daß es bei zwei binären Eingangssignalen vier mögliche Eingangssignal-Kombinationen gibt. Man kann auch drei, vier oder noch mehr Eingangssignale miteinander verknüpfen. Die Anzahl der Kombinationen steigt dann schnell an. Zum Aufzeigen des Prinzips genügen zwei Eingangssignale.

Verbal läßt sich der ermittelte Zusammenhang einfach formulieren:

> Die Variable am Ausgang eines UND-Glieds nimmt dann und nur dann den Wert 1 an, wenn die Variablen an allen Eingängen die Werte 1 haben.

Das Bild S28.2b zeigt das Schaltzeichen für eine UND-Verknüpfung von zwei Eingangssignalen.

Die ODER-Verknüpfung

In der Programmliste im Bild S25.1 haben wir die Anweisung Nr. 3 mit der Adresse 18ØE rot markiert. Das Bild S24.1 zeigt, daß mit dieser Anweisung (Nr. 3) die eigentliche Verknüpfung der Eingangssignale verursacht wird. Diese Anweisung wird von einem Befehl gebildet, der aus nur einem einzigen Ziffernpaar besteht. Wir können daraus bereits jetzt schließen, daß offenbar in der Maschinensprache die UND-Verknüpfungsanweisung durch das Ziffernpaar AØ ausgedrückt wird.

Versuch S 29.1

Die ODER-Verknüpfung mit dem Mikroprozessor

Wir wollen das Ziffernpaar AØ bei der Adresse 18ØE durch eine andere Maschinensprachen-Anweisung ersetzen. – Sorgen Sie dafür, daß Ihr System mit dem Programm aus dem Bild S25.1 geladen ist.

Ersetzen Sie das Ziffernpaar AØ bei der Adresse 18ØE durch das Ziffernpaar BØ:

```
RS, ADDR, 1, 8, 0, E
    DATA,  B, 0        (RS)
```

RS kann weggelassen werden.

Wir brauchen die Bedeutung dieser Tasten-Funktionen jetzt nicht mehr zu erläutern und brauchen auch nicht mehr zu beschreiben, wie Sie unsere folgende Anweisung ausführen können:

Starten Sie das Programm bei der Adresse 18ØØ!

Die Anzeige verlöscht. Betätigen Sie wieder die Tasten 7 und 3 versuchsweise und beobachten Sie die Anzeige!

Der funktionale Zusammenhang zwischen den Eingangssignalen und dem Ausgangssignal ist offenbar ein ganz anderer als der bei der UND-Verknüpfung.

Aufgabe S29.1

Ergänzen Sie die Funktionstabelle im Bild S29.1a für die ODER-Verknüpfung entsprechend dem Ergebnis des vorangegangenen Versuchs!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü4.

a)

e_1	e_2	a
Ø	Ø	0
Ø	1	1
1	Ø	1
1	1	1

b)

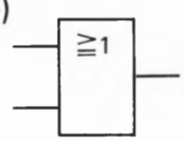


Bild S29.1
a) Zu Aufgabe S29.1: Ergänzen Sie bitte diese Funktionstabelle entsprechend dem Ergebnis des Versuchs S29.1.
b) Schaltzeichen für ein ODER-Verknüpfungsglied mit zwei Eingängen.

> Die Variable am Ausgang eines ODER-Glieds nimmt immer dann den Wert 1 an, wenn an mindestens einem Eingang eine Variable den Wert 1 hat.

Das Bild S29.1b zeigt das Schaltzeichen für eine ODER-Verknüpfung von zwei Eingangssignalen.

S

30

Die Exklusiv-ODER-Verknüpfung

Neben den UND- oder ODER-Verknüpfungen spielt in der Mikroprozessor-Technik die Exklusiv-ODER-Verknüpfung eine wichtige Rolle. Sie wird in der digitalen Hardware verhältnismäßig selten angewendet; in der Mikroprozessor-Technik ist sie dagegen fast wichtiger als die UND- und ODER-Verknüpfungen.

Der Begriff des exklusiven (ausschließenden) ODER, der offenbar mit dem normalen ODER nahe verwandt ist, ist weitgehend unbekannt. Ganz primitiv bedeutet das normale ODER:

Das eine ODER das andere ODER beide.

Ein Kind, das ein Stück Schokolade ODER ein Eis haben möchte, hat gewiß nichts dagegen, wenn es beides bekommt.

Ganz anders das exklusive ODER, das bedeutet:

Das eine ODER das andere, aber nicht beide.

Wenn der Arzt Herrn Meier ODER Frau Müller in sein Sprechzimmer bittet, dann wird er sicher nicht beide zugleich herein lassen.

Versuch S 30.1

Die Exklusiv-ODER-Verknüpfung mit dem Mikroprozessor

Ersetzen Sie – ähnlich wie im vorhergehenden Versuch beschrieben – das Ziffernpaar bei der Adresse 18ØE in dem Bild S 25.1 aufgelisteten Programm durch das Ziffernpaar A8 und starten Sie das Programm anschließend bei der Adresse 18ØØ!

Betätigen Sie versuchsweise die Tasten 7 und 3 einzeln und gemeinsam und sehen Sie sich jeweils den Wert des Ausgangssignals an!

a)

e_1	e_2	a
Ø	Ø	0
Ø	1	1
1	Ø	1
1	1	0

b)

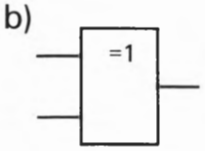


Bild S 30.1
a) Zu Aufgabe S 30.1: Ergänzen Sie bitte diese Funktionstabelle entsprechend dem Ergebnis des Versuchs S 30.1.
b) Schaltzeichen für ein Exklusiv-ODER-Verknüpfungsglied mit zwei Eingängen.

Aufgabe S 30.1

Ergänzen Sie die Funktionstabelle im Bild S 30.1a für die Exklusiv-ODER-Verknüpfung entsprechend dem Ergebnis des vorangegangenen Versuchs!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 4.

Die Funktion der Exklusiv-ODER-Verknüpfung, deren Schaltzeichen das Bild S 30.1b zeigt, läßt sich so beschreiben:

> Die Variable am Ausgang eines Exklusiv-ODER-Glieds nimmt nur dann den Wert 1 an, wenn an einem, und nur an einem Eingang die Variable den Wert 1 hat.

Die Tatsache, daß die Exklusiv-ODER-Funktion immer dann ein 1-Signal liefert, wenn die Eingangsvariablen untereinander verschieden sind, wird in Mikroprozessor-Programmen recht häufig beim Vergleich von Daten ausgenutzt.

Rechnen mit sedezimal dargestellten Zahlen

Bei der Art des Programmierens, die wir in diesem Lehrgang verwenden, also auf der Ebene der Maschinen- bzw. Assembler-Sprache, spielt sich die Zwiesprache mit Mikroprozessor-Systemen fast ausschließlich mit der sedezimalen Darstellung von Zahlen ab. Von großer Wichtigkeit ist es dabei, daß man Zahlen in sedezimaler Schreibweise addieren und vor allem subtrahieren kann. Warum das notwendig ist, werden Sie in den nächsten Abschnitten sehr schnell erkennen. Wir wollen uns deshalb mit diesem Thema auf den folgenden Seiten beschäftigen.

Addieren im Sedezimalsystem

Sie müssen beim Lesen dieses Abschnitts keineswegs einen Schrecken bekommen: Auch beim Rechnen mit sedezimal dargestellten Zahlen bleibt drei plus drei gleich sechs und zwölf minus sieben gleich fünf. Auch die Regeln für das schriftliche Addieren und Subtrahieren mit Übertrag und „Borgen" sind die gleichen, die Sie in der Schule für das Dezimalsystem gelernt haben. Man muß nur ein bißchen mehr aufpassen und daran denken, daß man es mit einem Sechzehner- und nicht mit einem Zehner-System zu tun hat. Wie einfach die Sache im Prinzip ist, können wir an ein paar Beispielen klar machen.

Was ergibt fünf plus sieben? Wir haben es schon angedeutet: An den Grundlagen der Mathematik ist nicht zu deuteln. Fünf plus sieben kann niemals etwas anderes als zwölf ergeben.

Schreiben Sie diese primitive Aufgabe nun in Ziffern an:

5 + 7 = ... Halt!

Jetzt müssen Sie sich entscheiden, ob Sie mit dem Dezimal- oder mit dem Sedezimalsystem arbeiten wollen. – Die Tabelle im Bild S 31.1, die Sie bereits aus dem Bild S17.1 kennen, zeigt, daß sowohl für die Zahl Fünf als auch für die Zahl Sieben in beiden Systemen das gleiche Zeichen verwendet wird. Noch stimmt das, was wir soeben angeschrieben haben, also auf jeden Fall. Die Entscheidung fällt beim Anschreiben des Ergebnisses „zwölf":

Dezimalsystem 5 + 7 = 12
Sedezimalsystem: 5 + 7 = C

Wenn wir unterstellen, daß bei unseren Überlegungen keine anderen als das Dezimal- und das Sedezimalsystem in Betracht kommen (und das wollen wir im folgenden tun), dann können wir hier die Kennzeichnung sedezimal dargestellter Zahlen durch das nachgestellte **H** getrost weglassen. Das angeschriebene Ergebnis der Addition zeigt deutlich, welches System wir gewählt haben. – Worauf es uns hier ankommt, ist dies: Bei Verwendung des Dezimalsystems brauchen wir zum Anschreiben des Ergebnisses zwei Zeichen; beim Sedezimalsystem kommen wir mit einem Zeichen aus.

Zahl	Dezimal	Sedezimal
Null	Ø	Ø
Eins	1	1
Zwei	2	2
Drei	3	3
Vier	4	4
Fünf	5	5
Sechs	6	6
Sieben	7	7
Acht	8	8
Neun	9	9
Zehn	10	A
Elf	11	B
Zwölf	12	C
Dreizehn	13	D
Vierzehn	14	E
Fünfzehn	15	F

Bild S31.1
Wiederholung von Bild S17.1. Tabelle der Zeichen für Zahlen im Dezimalsystem und im Sedezimalsystem.

Das ist so einfach, daß sich eigentlich jede weitere Erläuterung erübrigt. Und doch zeigt sich hier bereits das ganze Problem. Wenn es überhaupt ein Problem ist. Bei Verwendung des Dezimalsystems sind wir wegen der kleineren Anzahl verfügbarer unterschiedlicher Zeichen bereits beim Überschreiten der Zahl Neun gezwungen, die nächstfolgende Stelle in Anspruch zu nehmen, also einen Übertrag auf die nächsthöhere Stelle zu machen; beim Sedezimalsystem brauchen wir das erst beim Überschreiten der Zahl Fünfzehn zu tun.

Wir wollen die Gleichung „neun plus acht gleich siebzehn" in Ziffern anschreiben:

$9 + 8 = 17$

Hier haben wir das Dezimalsystem verwendet. Was Sie beim Anschreiben des Ergebnisses tun, ist so selbstverständlich, daß Sie darüber kaum Rechenschaft geben: Das Ergebnis überschreitet die Zahl neun um acht. Sie zählen also unbewußt ab neun – bei null beginnend – um acht weiter und merken die erstmalige Überschreitung der neun durch das vorgestellte Zeichen 1 an.

Wie sieht die Sache beim Anschreiben der Gleichung im Sedezimalsystem aus? – Hier überschreitet das Ergebnis die Zahl Fünfzehn, die gerade noch mit einem einzigen Zeichen darstellbar ist, um zwei. Wir müssen in Gedanken also ab fünfzehn – wieder bei null beginnend – um zwei weiterzählen, landen bei eins und merken die erstmalige Überschreitung der fünfzehn durch das vorgestellte Zeichen 1 an:

$9\mathbf{H} + 8\mathbf{H} = 11\mathbf{H}$

Hier sind wir gezwungen, mit **H** die Verwendung des Sedezimalsystems anzumerken, denn sonst lesen Sie: „Neun plus acht gleich elf" – und das ist natürlich Unsinn. Eine mit **H** gekennzeichnete Zeichenfolge dürfen Sie auf keinen Fall so lesen, als handle es sich um eine Ziffernfolge in Dezimaldarstellung! 11**H** ist also **nicht** gleich elf! Es ist ganz eindeutig siebzehn und nichts anderes. Allenfalls dürfen Sie lesen „eins eins sedezimal" oder – im Fachjargon – „eins eins hex".

Haben Sie einmal überlegt, warum man eigentlich beim Addieren von Zahlen vom Kopfrechnen zum schriftlichen Addieren übergeht? Offenbar ist man immer dann geneigt, zum Bleistift zu greifen, wenn man Zahlen addieren will, die sich dezimal nicht mehr mit einem einzigen Zeichen darstellen lassen. Sechs plus acht mit Hilfe des Bleistifts auf dem Papier auszurechnen, wäre lächerlich.

Die Aufgabe dreizehn plus achtzehn ist aber bereits ein Grenzfall. Wie würden Sie im Kopf rechnen? Sie vollziehen im Kopf mit Hilfe Ihres Gedächtnisses genau das, was Sie auch auf dem Papier tun würden: Sie addieren die Einer, merken sich die Überschreitung der neun durch einen Übertrag, addieren die Zehner und zählen dann den Übertrag hinzu. Wesentlich ist: Ohne gedankliche oder schriftliche Hilfsmittel können Sie im allgemeinen nur einstellig darzustellende Zahlen addieren.

Genau diese Möglichkeit bietet das schriftliche Addieren: Durch Untereinanderschreiben jeweils der Einer, der Zehner, der Hunderter usw. Bei Verwendung des Dezimalsystems ergibt sich ein übersichtliches Rechenschema zur Addition mehrstellig darstellbarer Zahlen:

```
    4  2  9
+   3  7  8
+   1  1
-------------
    8  0  7
```

Übertrag

Das gleiche Schema ist auch dann brauchbar, wenn zwei Zahlen mit mehreren Stellen im Sedezimalsystem dargestellt sind und addiert werden sollen. Jetzt werden jeweils die Einer, Sechzehner, Zweihundertsechsundfünfziger usw. untereinander geschrieben und dann wird stellenweise addiert. Der Übertrag wird jeweils an der nächsthöheren Stelle gutgeschrieben. Dabei muß man allerdings aufpassen: Ein Übertrag entsteht immer dann, wenn bei der Addition die Zahl Fünfzehn überschritten wird. – Wir zeigen das wieder an einem Beispiel:

```
      D  A  9  H
+     B  5  8  H
+  1  1  1
-------------------
   1  9  0  1  H
```

Übertrag

Aufgabe S 33.1

Kontrollieren Sie, ob die angegebene Addition stimmt! Wandeln Sie dazu die beiden dargestellten Summanden in dezimal dargestellte Zahlen um, addieren Sie und wandeln Sie die jetzt dezimal dargestellte Summe in die sedezimale Darstellung um!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 6.

Wir wollen das oben angegebene Additionsschema für die Zahlen DA9**H** und B58**H** etwas näher betrachten.

Einer:
Die Addition 9**H** + 8**H** = 11**H** haben wir auf der Seite S 32 bereits untersucht.

Sechzehner:
Hier muß die Aufgabe A**H** + 5**H** + 1**H** gelöst werden.

Die Addition des Übertrags 1 können wir zunächst außer Betracht lassen. – Vielleicht ist es Ihnen nie recht bewußt geworden, aber haben Sie nicht die Ergebnisse der Addition von jeweils zwei beliebigen, im Dezimalsystem einstellig darstellbaren Zahlen genauso auswendig im Kopf wie das kleine Einmaleins? Bei der schriftlichen Addition brauchen Sie das Additionsergebnis einer Dezimalstelle nur noch in dezimale Zeichen umzuformen und einschließlich des Übertrags anzuschreiben.

Es wäre zu überlegen, ob man dieses Auswendigwissen auf den Zahlenbereich erweitern sollte, der sich im Sedezimalsystem einstellig darstellen läßt. Es ist nichts dagegen einzuwenden. Wir machen es anders. Wahrscheinlich umständlicher, aber mit weniger Gedächtnisaufwand. Bei der Sechzehnerstelle z. B. lesen wir: A = zehn, 5 = fünf und rechnen nun: Zehn plus fünf gleich fünfzehn. – Jetzt addieren wir den Übertrag: Fünfzehn plus eins gleich sechzehn. – Das ist mehr als einfach.

Wenn überhaupt etwas kompliziert dabei ist, dann ist es die Darstellung des Ergebnisses im Sedezimalsystem, die wir ja bereits eingehend erläutert haben. Aber auch das ist jetzt besonders einfach. Entweder ist das Ergebnis kleiner als sechzehn. In diesem Fall liefert die Tabelle im Bild S 31.1 sofort das Ergebnis.

Oder das Ergebnis ist gleich oder größer als sechzehn. Dann entsteht ein Übertrag von eins. Das Additionsergebnis, vermindert um den übertragenen Sechzehner, wird entsprechend der Tabelle im Bild S 31.1 angeschrieben. Für das hier betrachtete Additionsergebnis der Sechzehnerstelle bedeutet das: Es ist gleich sechzehn. Es entsteht also ein Übertrag von eins. – Das Ergebnis sechzehn wird um den übertragenen Sechzehner vermindert. Was bleibt übrig? Null. Also schreiben wir das Zeichen Ø in der Sechzehnerstelle an.

Zweihundertsechsundfünfziger:
Es ist die Aufgabe D**H** + B**H** + 1H zu lösen.

Wir addieren zunächst: Dreizehn plus elf gleich vierundzwanzig. Dann wird der Übertrag addiert: Vierundzwanzig plus eins gleich fünfundzwanzig.

Die Zahl Fünfundzwanzig wird sedezimal dargestellt. Sie ist größer als sechzehn, es entsteht also ein Übertrag von eins. Nun wird das Ergebnis (fünfundzwanzig) um den zu übertragenden Sechzehner vermindert. Übrig bleiben neun, die sedezimal in der Zweihundertsechsundfünfzigerstelle angeschrieben werden.

Viertausendsechsundneunziger:
Hier ergibt sich als einziger Summand der Übertrag aus der vorhergehenden Stelle.

Aufgabe S 34.1

Führen Sie schriftlich, ohne Umwandlung in die Dezimaldarstellung, die folgenden Additionen durch:

a) 7C**H** + 3B**H**
b) 135**H** + 427**H**
c) ABC**H** + DEF**H**
d) 1E4A**H** + 2AC1**H**

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 6.

Subtrahieren im Sedezimalsystem

Es gibt Tricks, das Subtrahieren sedezimal dargestellter Zahlen zu vereinfachen. Einer davon ist besonders elegant und wir werden ihn später noch vorstellen: Sie können nämlich Ihr System rechnen lassen. Aber obwohl ein Auto heute fast zur Normalausstattung des mündigen Bürgers gehört, wird man es doch selten verabsäumen, die kleinen Kinder das Laufen zu lehren. Also werden wir uns doch mit der ach so unbeliebten Subtraktion beschäftigen.

Bei der Subtraktion ist es nicht anders als bei der Addition: So wie beim Addieren ein überschüssiger Sechzehner in die nächste Stelle übertra-

gen wird, so wird beim Subtrahieren einfach ein fehlender Sechzehner von der nächsthöheren Stelle „geborgt“, wenn stellenweise subtrahiert wird. Dieses Borgen ist immer dann nötig, wenn der Subtrahend an einer Stelle der Zeichenkombination größer ist als der Minuend.

Ehe es nun ernst wird, wollen wir schnell noch sagen, was es mit Minuend und Subtrahend auf sich hat. Beide Ausdrücke kommen (wie vieles Gelehrtes) aus dem Lateinischen:

Minuend ist die Zahl, die vermindert, also durch Subtraktion kleiner gemacht werden soll (*minùere* = verkleinern).
Subtrahend ist die Zahl, die vom Minuenden abgezogen wird (*subtràhere* = wegziehen).
Differenz ist das Ergebnis der Subtraktion. Es ist die Zahl, um die sich Minuend und Subtrahend unterscheiden (*differre* = sich unterscheiden).

Auf geht's:

Minuend:		1	7	**H**
Subtrahend:	–		8	**H**
		Differenz		

Zunächst stellen wir fest: An der Einerstelle ist der Subtrahend (8**H**) größer als der Minuend (7**H**). Wir beziehen also die 1 an der Sechzehnerstelle in unsere Rechnung ein und ziehen nicht acht von sieben ab, sondern acht von sieben plus sechzehn, also von dreiundzwanzig. Das Ergebnis ist fünfzehn. Sedezimal geschrieben:

Minuend:		1	7	**H**
Subtrahend:	–		8	**H**
Differenz:			F	**H**

Wie sind wir hier vorgegangen? Ganz genau so, wie Sie es auch bei schriftlicher Rechnung im Dezimalsystem tun: Wir haben eine 1 von der nächsthöheren Stelle „geborgt“ und diesen einen (hier einzigen) Sechzehner zu den zu wenigen Einern hinzu addiert. Dadurch ist der Minuend größer als der Subtrahend geworden; die Subtraktion wurde möglich, ohne daß wir „ins Minus“ gerieten.

Genau genommen müssen wir nun noch die zweite, also die Sechzehnerstelle betrachten, denn der Minuend ist ja zweistellig. Aber das ist hier müßig: Wir haben den einzig vorhandenen Sechzehner „weggeliehen“, und der Subtrahend war sowieso nur einstellig; er enthielt keinen Sechzehner. Für die zweite Stelle gilt also: Null minus nichts. Und das dürfen wir vergessen.

Das war einfach. Wir können uns jetzt an einer ordentlichen Aufgabe versuchen.

Minuend:		C	2	3	**H**
Subtrahend:	–	A	2	6	**H**
			Differenz		

Wir bearbeiten die Aufgabe stellenweise und geben den Stand der Rechnung jeweils nebenstehend auf dem Rand an.

Einer

	C	2'	3	H
			13	H
–	A	2	6	H
			D	H

Einer:
Der Subtrahend (sechs) ist größer als der Minuend (drei).
Damit wir die Subtraktion durchführen können, „borgen" wir einen Sechzehner von der nächsthöheren Stelle. Nebenstehend haben wir das Ergebnis dieses Borgens unter dem Minuenden rot eingetragen und durch Anstreichen der Ziffer an der Sechzehnerstelle des Minuenden angemerkt, daß ein Sechzehner ausgeliehen wurde. Dieses Anstreichen bewährt sich auch in der Praxis; das Anschreiben der rot eingetragenen Zeile kann man sich jedoch normalerweise sparen.

Anleihe: Sechzehn plus drei gleich neunzehn.
Neunzehn minus sechs gleich dreizehn.
Dreizehn = D**H** (Bild S 31.1)

Sechzehner

	C'	2'	3	H
		11		H
–	A	2	6	H
		F	D	H

Sechzehner:
Wegen des Ausborgens an die Einerstelle ist von den ursprünglich zwei Sechzehnern nur noch ein Sechzehner vorhanden. – Der Subtrahend ist nun größer als der Minuend.

Damit wir die Subtraktion durchführen können, wird eine Anleihe bei der nächsthöheren Stelle gemacht und diese Anleihe durch Anstreichen angemerkt. Die Zahl an der Sechzehnerstelle wird also um einen Zweihundertsechsundfünfziger erhöht. So kompliziert braucht man aber nicht zu denken, denn der Witz ist ja beim stellenweisen Rechnen gerade, daß man immer so tun kann, als hätte man es mit Einern oder höchstens mit Sechzehnern zu tun.

Anleihe: Sechzehn plus eins gleich siebzehn.
Siebzehn minus zwei gleich fünfzehn.
Fünfzehn = F**H** (Bild S 31.1)

Zweihundertsechsundfünfziger

	C'	2'	3	H
	B			H
–	A	2	6	H
	1	F	D	H

Zweihundertsechsundfünfziger:
Wegen des Ausborgens an die Sechzehnerstelle sind von den ursprünglich zwölf Zweihundertsechsundfünfzigern nur noch elf vorhanden. Der Subtrahend ist kleiner als der Minuend. Wäre das nicht der Fall, dann könnten wir die Subtraktion nicht ausführen, ohne ins Minus zu geraten, denn eine Anleihe an die wiederum nächsthöhere Stelle ist nun nicht mehr möglich.

Elf minus zehn gleich eins.
Eins = 1**H** (Bild S 31.1)

Das Gesamtergebnis unserer Rechnung sieht nun so aus:

Minuend:		C	2	3	**H**
Subtrahend:	–	A	2	6	**H**
Differenz:		1	F	D	**H**

Aufgabe S 36.1

Führen Sie schriftlich die folgenden Subtraktionen durch:

a) FC**H** – 32**H**
b) 467**H** – 45A**H**
c) 5A6**H** – 2C7**H**
d) FF7C**H** – AF5F**H**

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 6.

Die Darstellung von Zahlen durch Zeichen

(Fortsetzung)

Die Darstellung einer so geheimnisvollen Sache wie Zahlen durch Zeichen gehört zu den größten Leistungen des menschlichen Geistes. Durch unterschiedliche Darstellungs-Systeme sind ganz Kulturen gekennzeichnet. Die ausschließliche Verwendung des Dezimalsystems in unserem technischen Zeitalter ist gar nicht so selbstverständlich, wie man es vielleicht annehmen mag: Noch unseren Großeltern war es ganz geläufig, zwei Dutzend (zweimal zwölf) oder ein Schock (fünfmal zwölf) Eier einzukaufen. Offenbar handelt es sich dabei um eine Art Duodezimalsystem (Zwölfersystem). – Und denken Sie einmal daran, daß unsere Sprache ursprünglich wohl nicht dezimal orientiert war (Seite S14): Bis zwölf verwendet sie einsilbige Wörter, und dann geht es mit mehrsilbigen Wörtern weiter. Anscheinend konnten unsere Urväter nur bis zwölf zählen. Und das ist gar nicht so unpraktisch, denn es gibt zwölf Monate, das Ziffernblatt einer Uhr zeigt zwölf Stunden, zwölf läßt sich durch zwei, drei, vier und sechs teilen (zehn nur durch zwei und fünf) usw.

Aber jetzt ist's genug. Mit Mikroprozessoren hat das wenig zu tun. Wir wollten Ihnen lediglich andeuten, wie interessant die Beschäftigung mit solchen Systemen sein kann.

Interessant im Zusammenhang mit Mikroprozessoren ist das Sedezimalsystem und manchmal das Oktalsystem. Warum eigentlich? Wir haben das bereits früher anklingen lassen (Seite S 30): Bei diesen Systemen besteht ein direkter Zusammenhang zu dem System, mit dem der Mikroprozessor tatsächlich arbeitet: Dem Dualsystem.

Das Dualsystem

Der Name dieses Systems kommt vom lateinischen Wort *duo*, das „zwei" bedeutet. Es handelt sich also um ein System, mit dem sämtliche Zahlen durch die Kombination von Zeichen aus einem Vorrat von nur zwei unterschiedlichen Zeichen dargestellt werden können: Den Ziffern Ø und 1.

Wir wollen das Dualsystem zunächst ganz einfach als System zur Darstellung von Zahlen betrachten. Welche Vorteile es in der Technik bietet, gehört in den Abschnitt Hardware. Interessant ist jedoch, daß man sich bei diesem System Gedanken über die Zeichen selbst gemacht hat. Es ist grundsätzlich nicht sehr sinnvoll, in allen Systemen zur Darstellung von Zahlen die gleichen Zeichen wie im Dezimalsystem zu verwenden.

Diese Überlegungen haben dazu geführt, im Dualsystem die beiden Zeichen O und L zu verwenden. Da das Zeichen L in der Hardware jedoch als Abkürzung für LOW, also für einen niedrigen Spannungspegel benutzt wurde, ergaben sich unangenehme Verwechslungen, und man ist beim Dualsystem wieder zu den Zeichen Ø und 1 zurückgekehrt. Wir werden in diesem Lehrgang ebenfalls die Zeichen Ø und 1 verwenden.

Zunächst ordnen wir den Zeichen des Dualsystems zwei Zahlen zu:

Zahl:	Null	Eins
Zeichen:	Ø	1

Wenn wir beim Zwölfersystem noch einigermaßen gläubig hinnehmen, daß unsere Urväter vielleicht nur bis zwölf zählen konnten, weil offenbar nur zwölf unterschiedliche sprachliche Zeichen zur Darstellung von Zahlen verfügbar waren, dann müssen wir jetzt feststellen, daß das Dualsystem doch wohl nur sehr wenig Bildung voraussetzt. Selbst vom primitivsten Ureinwohner Australiens nehmen wir doch an, daß er wenigstens bis drei zählen konnte:

Eins – zwei – drei – ganz viele.

Aber es ist ja gerade recht: Das Dualsystem ist ausschließlich im Zusammenhang mit Computern interessant. Und weil ein Computer dumm im höchsten Grade ist, dürfen wir ihm die Intelligenzleistung, bis zwei zu zählen, keineswegs zumuten.

Und was ist, wenn er bis zwei, drei oder gar bis vierzehntausendundsieben zählen soll, wenn er doch nur mit zwei unterschiedlichen Zeichen arbeiten kann? Dann müssen wir ihn eben auf die Verarbeitung eines Stellenwertsystems programmieren. Wir stellen fest:

Das Dualsystem ist ein Stellenwertsystem.

Was es mit Stellenwertsystemen auf sich hat, haben wir bereits eingehend erörtert (Seiten S15 und S20). Bei den im Zeichenvorrat des Dualsystems vorhandenen, unterschiedlichen Zeichen sind die Stellen, an denen die Zeichen in einer Zeichenkombination erscheinen, mit Potenzen von zwei bewertet:

5. Stelle	4. Stelle	3. Stelle	2. Stelle	1. Stelle
Sechzehner	Achter	Vierer	Zweier	Einer
mal 2^4	mal 2^3	mal 2^2	mal 2^1	mal 2^0

An jeder dieser Stellen kann nur eins der beiden Zeichen Ø und 1 stehen. Mit der Erfahrung, die Sie beim Sedezimalsystem gewonnen haben, ist es nun leicht, eine dual dargestellte Zahl zu lesen.

Wir wählen als Beispiel die Zeichenfolge 11Ø1Ø. Da die gleichen Zeichen für null und eins wie im Dezimalsystem verwendet werden, ist man zunächst geneigt, diese Zeichenfolge als elftausendundzehn zu lesen. Stutzig wird man allerdings sofort, weil in dieser Zeichenkombination nur zwei unterschiedliche Zeichen vorkommen. Aber warum sollte es sich nicht wirklich um die dezimal dargestellte Zahl Elftausendundzehn handeln?

Wenn man sicher gehen will, dann kann man die Zeichenkombination als dual dargestellte Zahl mit einem nachgestellten Buchstaben **B** (von lateinisch **B**ini = jeweils zwei) kennzeichnen: 11Ø1Ø**B**. Um welche Zahl handelt es sich jetzt?

Man geht ähnlich vor wie bei einer sedezimal dargestellten Zahl:

Einmal sechzehn + einmal acht + nullmal vier + einmal zwei + nullmal eins

oder

2^0	=	1
2^1	=	2
2^2	=	4
2^3	=	8
2^4	=	16
2^5	=	32
2^6	=	64
2^7	=	128
2^8	=	256
2^9	=	512
2^{10}	=	1Ø24
2^{11}	=	2Ø48
2^{12}	=	4Ø96
2^{13}	=	8192
2^{14}	=	16384
2^{15}	=	32768

Bild S38.1
Die hier aufgelisteten Potenzen der Zahl Zwei braucht man beim Lesen dual dargestellter Zahlen.

einmal 2^4 + einmal 2^3 + nullmal 2^2 + einmal 2^1 + nullmal 2^0

Das ergibt sechsundzwanzig, und es gilt:

11Ø1Ø = 26

Sie erkennen bereits jetzt, daß man bei der Darstellung im Dualsystem schon bei recht kleinen Zahlen verhältnismäßig viele Stellen braucht.

Aufgabe S 39.1

Welches ist die größte Zahl, die mit fünf Stellen im Dualsystem dargestellt werden kann?

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 8.

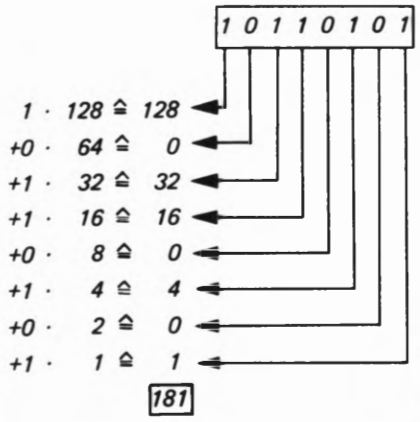


Bild S 39.1
Muster für die Umcodierung einer im Dualsystem dargestellten Zahl in die Darstellung im Dezimalsystem. Wir haben die zu addierenden Stellenwerte bereits im Dezimalsystem angeschrieben.

Bei der Darstellung größerer Zahlen im Dualsystem steigt der Stellenbedarf recht schnell an. Das ist der Preis, den man für das Arbeiten mit dem dummen Computer zahlen muß. Aber es lohnt sich offenbar, denn ein Computer kann so dargestellte Zahlen mit unglaublicher Geschwindigkeit verarbeiten.

Da die Stellenwerte bei der Dual-Darstellung von rechts gezählt jeweils um eins zunehmende Potenzen von zwei aufweisen, sollte man beim Lesen dual dargestellter Zahlen die Potenzen von zwei im Kopf haben. Es reicht allerdings meistens, wenn man aus der Auflistung in Bild S 38.1 die Potenzen bis 2^{10} auswendig weiß.

Im Bild S 39.1 zeigen wir am Beispiel der Zeichenfolge 1Ø11Ø1Ø1 die Umcodierung der dargestellten Zahl in das Dezimalsystem. Beginnend bei der (von rechts gezählt) ersten Stelle (Stellenwert $2^0 = 1$) hat die sechste Stelle den Stellenwert $2^5 = 32$, die siebte Stelle den Stellenwert $2^6 = 64$ und die achte Stelle den Stellenwert $2^7 = 128$. Mit der Kenntnis dieser Stellenwerte ist das Umcodieren kein Kunststück.

Aufgabe S 39.2

Wandeln Sie die folgenden Zeichenfolgen des Dualsystems in die Darstellung im Dezimalsystem um:

a) 1Ø1Ø b) 1Ø1Ø1 c) 11ØØØ1 d) 1Ø111ØØ e) 11ØØØ1Ø1

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 8.

Wie beim Sedezimalsystem ist die Darstellung einer vorgegebenen Zahl im Dualsystem etwas schwieriger als der umgekehrte Vorgang. Das Prinzip der Darstellung einer vorgegebenen Zahl in dualer Schreibweise ist jedoch das gleiche wie beim Sedezimalsystem. Wir können uns also kurz fassen:

Man untersucht die darzustellende Zahl daraufhin, welche höchste Potenz von zwei in ihr enthalten ist und merkt dieses Enthaltensein durch die Ziffer 1 an der entsprechenden Stelle an. – Dieser Vorgang ist beim Dualsystem besonders einfach, weil ja nur mit den beiden Ziffern

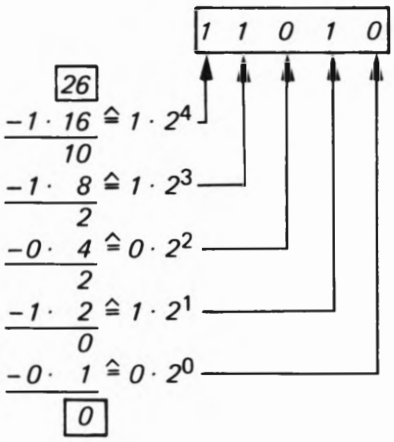


Bild S 40.1
Schema zur Umcodierung der Zeichenkombination 26 in die entsprechende Zeichenkombination im Dualsystem. Von der im Dualsystem darzustellenden Zahl müssen nacheinander sämtliche Potenzen von zwei abgezogen werden, die kleiner sind oder gleich der darzustellenden Zahl. Als letzte wird die nullte Potenz von zwei abgezogen.

S
40

Ø und 1 gearbeitet wird. Eine bestimmte höchste Potenz kann also nur nullmal oder einmal in der darzustellenden Zahl enthalten sein.

Anschließend zieht man diese höchste Potenz von zwei von der darzustellenden Zahl ab und schaut nun nach, ob die nächstkleinere Potenz im verbleibenden Rest vorhanden ist (ja: 1) oder nicht (nein: Ø). So fährt man fort, bis man bei der nullten Potenz von zwei ($2^0 = 1$) angekommen ist.

Wir haben diesen Vorgang für die Zahl Sechsundzwanzig im Bild S 40.1 schematisch dargestellt und wollen uns noch einmal Schritt für Schritt ansehen, wie es gemacht wird:

Sechsundzwanzig ist kleiner als die fünfte Potenz von 2 ($2^5 = 32$, Bild S 38.1). Also ist die höchste, in sechsundzwanzig enthaltene Potenz von zwei die vierte ($2^4 = 16$).

$26 : 2^4 = 26 : 16 = 1$	(Rest 1Ø)	1....
$1Ø : 2^3 = 1Ø : 8 = 1$	(Rest 2)	11...
$2 : 2^2 = 2 : 4 = Ø$	(Rest 2)	11Ø..
$2 : 2^1 = 2 : 2 = 1$	(Rest Ø)	11Ø1.
$Ø : 2^0 = Ø : 1 = Ø$	(Rest Ø)	11Ø1Ø

Man darf bei solchen Darstellungen, gleichgültig, welches Stellenwertsystem benutzt wird, niemals vergessen, die Einerstelle mit einem Zeichen zu besetzen; auch dann nicht, wenn dieses Zeichen Ø ist. Beim Lesen interpretiert man die (von rechts gezählt) erste Stelle automatisch als Einerstelle. Hat man dann etwa im hier behandelten Beispiel das Zeichen Ø an der Einerstelle vergessen anzuschreiben, dann liest man später 11Ø1, und das bedeutet dreizehn!

Aufgabe S 40.1

Wandeln Sie die folgenden dezimalen Zeichenfolgen in die duale Darstellung um:

a) 7 b) 13 c) 36 d) 436 e) 1977

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 8.

Wir fassen zusammen:

Das Dualsystem ist ein Stellenwertsystem, bei dem Zahlen aus einem Zeichenvorrat von zwei unterschiedlichen Zeichen benutzt werden.

Die Stellen bei einer dual dargestellten Zahl sind (von rechts beginnend) mit steigenden Potenzen von zwei bewertet.

Das Zusammenwirken von CPU und Speicher im Mikroprozessor-System

In der CPU eines Mikroprozessor-Systems geht es zu wie in einem ordentlichen Staatswesen: Ganz oben in der Wertskala steht eine perfekte Verwaltung. Wenn die nur richtig funktioniert, dann stellt sich die produktive Tätigkeit mit einem Minimum an zusätzlichem Aufwand (hoffentlich) von selbst ein.

Mit dieser (traurigen) Tatsache müssen wir uns auch beim Mikroprozessor abfinden. Ehe wir uns mit Befehlen beschäftigen, die mit den zu verarbeitenden Daten etwas wirklich Gescheites anfangen, müssen wir erst die reine Verwaltung dieser Daten kennenlernen. Und dazu gehört natürlich auch das Laden von Daten aus der CPU in den Speicher und vom Speicher in die CPU. Denken sollten Sie daran, daß darin noch nichts von der Intelligenz steckt, die man dem Mikroprozessor nachsagt. Es ist ja eine Verwaltung.

Da im Mikroprozessor-System Daten in Form von Bytes behandelt werden, die aus acht Bit bestehen, muß der Datentransport zwischen CPU und Speicher über acht elektrische Leitungen vorgenommen werden. Solche Leitungen zum parallelen Transport mehrerer Bits werden in der Fachsprache als **Bus** bezeichnet.

> Ein Bus ist ein Bündel von Leitungen, über das mehrere Informations-Sender und Empfänger miteinander verbunden werden.

Beim hier angesprochenen Transport von Daten spricht man vom **Datenbus**. Da dem Speicher gleichzeitig mit den übermittelten Daten auch mitgeteilt werden muß, bei welchen Adressen er die übermittelten Daten ablegen soll, braucht er als weitere Information jeweils eine Adresse, die mit einem aus sechzehn Leitungen bestehenden **Adreßbus** übertragen werden. Zusammen mit einem zusätzlichen **Steuerbus**, den wir hier zunächst nur erwähnen wollen, ergibt sich ein Bus-System, das in der Hardware von Mikroprozessor-Systemen eine wichtige Rolle spielt.

Direkter Datentransport zwischen Akkumulator und Speicher

Beim Z80-Mikroprozessor gibt es mehrere elegante Möglichkeiten, Daten über den Datenbus zwischen CPU und Speicher zu transportieren. Wir wollen uns hier für diesen Zweck auf die Verwendung des HL-Registerpaares als Memory-Pointer (Seite H44) zur Adressierung beliebiger Speicherzellen beschränken, wie sie auch beim 8085-Mikroprozessor praktiziert wird.

Ehe wir Ihnen zeigen, welche Rolle das HL-Registerpaar beim Adressieren von Speicherzellen spielt, sollen Sie noch eine scheinbar viel einfachere und zunächst einleuchtendere Methode kennenlernen, mit der man beliebige Bytes in beliebige Speicherzellen hineinbringen und

umgekehrt auch die Inhalte beliebiger Speicherzellen in das zentrale Register Akkumulator kopieren kann.

Versuch S42.1

Direktes Abspeichern des Akkumulator-Inhalts

Tasten Sie die folgenden Operationscodes und Operanden in Ihr System ein:

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand
1	START	LD	A,12	18ØØ	3E	12	
2		LD	(18AB),A	18Ø2	32	AB	18

Löschen Sie die Anzeigen der Register-Inhalte (vgl. Versuch H15.1) und lassen Sie den Befehl Nr. 1, LD A,12, in einem Einzelschritt ausführen. Die Abfrage der Register-Inhalte ergibt erwartungsgemäß, daß der Akkumulator mit dem Byte 12 geladen wurde.

Holen Sie nach der Register-Abfrage die Adresse des nächsten auszuführenden Befehls in die Anzeige zurück (Taste PC), und lassen Sie den Ihnen noch rätselhaften Befehl Nr. 2, LD (18AB),A, in einem Einzelschritt ausführen! – In der Anzeige erscheint als Adresse des nun folgenden, aber nicht mehr programmierten Befehls 18Ø5.

Kontrollieren Sie wieder die Inhalte der CPU-Register! Sie stellen fest, daß sich nach der Ausführung des zweiten Befehls kein Register-Inhalt geändert hat. Was hat der Befehl Nr. 2 denn nun bewirkt?

Fragen Sie den Inhalt der Speicherzelle mit der Adresse 18AB ab:

ADDR, 1, 8, A, B; Anzeige: 1 8 A B 1 2 Also: (18AB) = 12

Der Inhalt einer Speicherzelle wird – ähnlich wie der Inhalt eines CPU-Registers – durch die in Klammern gesetzte Adresse dieser Speicherzelle angegeben.

Natürlich kann es Zufall sein, daß Sie für diese Speicherzelle den gleichen Inhalt finden wie für den Inhalt (A) = 12. Das können Sie kontrollieren, wenn Sie den Inhalt der Speicherzelle mit der Adresse 18AB löschen (DATA, 0) und anschließend den gerade durchgeführten Ablauf der beiden Befehle LD A,12 und LD (18AB),A noch einmal in Einzelschritten wiederholen.

Fragen Sie anschließend wieder den Inhalt der vorher gelöschten Speicherzelle mit der Adresse 18AB ab! Sie finden dort wieder das Byte 12, und das kann nun kein Zufall mehr sein. – Wenn es Ihnen Freude macht, dann können Sie den Versuch noch einmal durchführen, aber vorher das Byte bei der Adresse 18Ø1 in z. B. FE ändern. Nach dem Ablauf der beiden Befehle in Einzelschritten finden Sie in der Speicherzelle mit der Adresse 18AB immer das vorher mit dem Befehl LD A,Konst in den Akkumulator geladene Byte.

Sie haben herausgefunden, daß mit dem Befehl LD (adr),A der Inhalt des Akkumulators in die Speicherzelle mit der Adresse adr kopiert wird. – Überzeugen Sie sich, daß es sich tatsächlich um einen **Kopier**-Befehl handelt, und fragen Sie nach dem Ablauf der beiden programmierten Befehle den Inhalt des Akkumulators ab! Nach der Ausfüh-

rung des LD (adr),A-Befehls hat sich der Akkumulator-Inhalt nicht geändert.

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD (adr),A	32	adr ← (A)	Der Inhalt des Akkumulators wird in die Speicherzelle mit der Adresse adr kopiert.

Im mnemonischen Code dieses Befehls wird – wie bei den LD r,r'-Befehlen – zuerst angegeben, wohin etwas kopiert werden soll; hinter dem Komma steht, woher kopiert werden soll.

In diesem Befehl bedeutet der Operand adr eine sedezimal vierstellig dargestellte Adresse (2 Bytes). Es handelt sich also um einen Drei-Byte-Befehl. Dabei ist – ähnlich wie bei den Drei-Byte-Befehlen LD rp,Konst (rp = Registerpaar) – eine wichtige Besonderheit zu beachten (vgl. Seite H37):

> Bei Drei-Byte-Befehlen, deren Operanden eine Adresse darstellen, enthält der erste Operand das niederwertige Adreß-Byte (LOB) und der zweite Operand das höherwertige Adreß-Byte (HOB).

Im vorangegangenen Versuch sieht der Befehl Nr. 2 also so aus:

Operation	Operand	Adresse	Opcode	1. Oper.	2. Oper.
LD	(18AB),A	18Ø2	32	AB	18
		HOB LOB		LOB	HOB
		Adresse		Adresse	

Im Bild S43.1 sehen Sie, wie solche Befehle im Speicher abgelegt sind:

Das **LOB** der Adresse ist bei der **niederen** Adresse abgelegt;
das **HOB** der Adresse ist bei der **höheren** Adresse abgelegt.

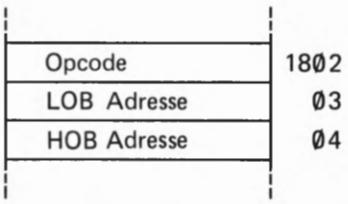


Bild S43.1
Ist der Operand eines Befehls eine Adresse, dann steht das LOB dieser Adresse im Speicher bei der kleineren Adresse; das HOB steht bei der höheren Adresse.

Das Gegenstück zum Befehl LD (adr),A bildet der Befehl LD A,(adr):

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD A,(adr)	3A	A ← (adr)	Der Inhalt der Speicherzelle mit der Adresse adr wird in den Akkumulator kopiert.

Auch bei diesem Befehl enthält der Operand adr des mnemonischen Codes eine sedezimal vierstellig dargestellte Adresse. Es handelt sich ebenfalls um einen Drei-Byte-Befehl, bei dem der erste Operand das LOB und der zweite Operand das HOB der Adresse enthält.

Die beiden hier vorgestellten Befehle stellen eine sehr einfache und einleuchtende Kommunikation zwischen der CPU und dem Speicher dar. Sie entsprechen etwa der Tätigkeit eines Lehrlings, der vom Büro-

Chef (dem Akkumulator) zum Ablegen oder Abholen jeder einzelnen Seite eines Vorgangs, den ein Sachbearbeiter benötigt, in die Registratur geschickt wird.

Unser Mikroprozessor enthält sehr viel elegantere Anweisungen zur Kommunikation zwischen CPU und Speicher. Sie können sich aber zunächst ansehen, wozu die beiden hier vorgestellten Befehle taugen.

Versuch S 44.1

Austausch der Inhalte zweier Speicherzellen

Laden Sie über das Tastenfeld die Speicherzelle mit der Adresse 183B mit dem Byte 12 und die Speicherzelle mit der Adresse 1923 mit dem Byte AB (Bild S 44.1a). – Tasten Sie anschließend ab der Adresse 18ØØ diese Befehlsfolge ein:

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand
1	START	LD	A,(183B)	18ØØ	3A	3B	18
2		LD	B,A	18Ø3	47		
3		LD	A,(1923)	18Ø4	3A	23	19
4		LD	(183B),A	18Ø7	32	3B	18
5		LD	A,B	18ØA	78		
6		LD	(1923),A	18ØB	32	23	19
7		RST	Ø	18ØE	C7		

Aufgabe S 44.1

In der aufgelisteten Befehlsfolge sind Ihnen alle Befehle bekannt bis auf die Anweisung Nr. 7, RST Ø. Diese Anweisung bewirkt das gleiche, als würde die Taste RS betätigt.

Das Programm ist so durchsichtig, daß sich die Aufstellung eines Programmablaufplans erübrigt. – Beschreiben Sie mit Worten die Wirkungsweise des Programms.

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 9.

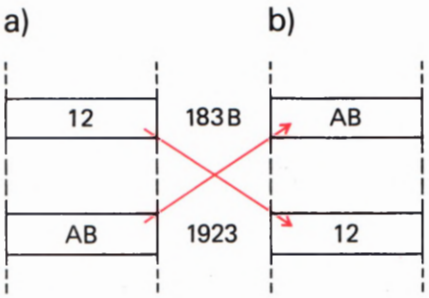


Bild S 44.1
Zu Versuch S 44.1: Die Inhalte der Speicherzellen mit den Adressen 183B und 1923 sollen ausgetauscht werden.

Starten Sie das Programm und sehen Sie sich anschließend die Inhalte der Speicherzellen mit den Adressen 183B und 1923 an! Es ist

(183B) = AB und (1923) = 12

Das Bild S 44.1b zeigt die neuen Inhalte der Speicherzellen.

Die Verwendung des HL-Registerpaares als Memory-Pointer

Das sechzehn Bit breite Register **Programmzähler** ist ein Pointer-Register (Seite H 44): Sein **Inhalt** wird von der CPU **immer** als **Adresse** einer Speicherzelle interpretiert. Den Inhalt dieser Speicherzelle betrachtet die CPU grundsätzlich als Operationscode oder als Operanden eines Befehls, den sie in einem internen Register „Befehlsdecoder" zur weiteren Verarbeitung ablegt. Das Bild S 45.1a stellt einen solchen Vorgang dar.

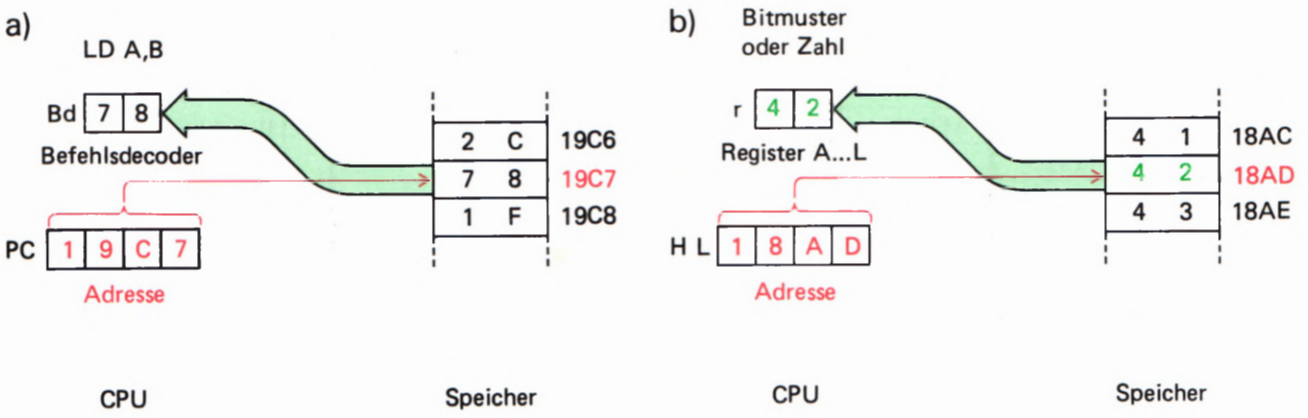


Bild S45.1
a) Der Inhalt des Programmzählers ist immer die Adresse des Operationscodes oder des Operanden eines Befehls.
b) Der Inhalt des HL-Registerpaares kann die Adresse einer Speicherzelle sein, in der ein Bitmuster oder eine Zahl abgelegt ist.

S
45

Der **Inhalt** des **HL-Registerpaares kann** von der CPU als **Adresse** interpretiert werden, je nach Art des jeweiligen Befehls. Der Inhalt der vom HL-Registerpaar adressierten Speicherzelle wird dann aber nicht als Befehl aufgefaßt, sondern als Bitmuster oder als sedezimal dargestellte Zahl, die in irgendeiner Form verarbeitet werden soll.

Die primitivste Art dieser Verarbeitung ist die Verwaltung, also das Kopieren des Bytes in eins der Register A bis L. Das Bild S45.1b zeigt als Beispiel eines solches Vorgangs, wie der Inhalt der vom HL-Registerpaar adressierten Speicherzelle in eins der CPU-Register kopiert wird.

Für diese Art der Verwendung des HL-Registerpaares stehen im Befehlssatz insgesamt sieben Befehle der Form LD r,(HL) zur Verfügung:

Mnemonischer Code	Operationscode	Operation	Erläuterung
LD A,(HL)	7E	r ← (HL)	Register r mit dem Inhalt der Speicherzelle laden, die durch den Inhalt des Registerpaares HL adressiert ist.
LD B,(HL)	46		
LD C,(HL)	4E		
LD D,(HL)	56		
LD E,(HL)	5E		
LD H,(HL)	66		
LD L,(HL)	6E		

Diese Befehle sind Ein-Byte-Befehle, weil die Adresse der angesprochenen Speicherzelle im HL-Registerpaar zur Verfügung steht. Voraussetzung ist natürlich, daß dieses Registerpaar vorher mit dieser Adresse geladen worden ist.

Es sei hier nur der Vollständigkeit darauf hingewiesen, daß die sechzehn Bit breiten Register IX und IY des Z80-Mikroprozessors (Seite T2) eine ganz ähnliche, aber noch weitergehende Funktion haben, als sie hier für das HL-Registerpaar beschrieben wurde. Mit entsprechenden Befehlen kann nicht nur die Speicherzelle mit der im jeweiligen Register abgelegten Adresse angesprochen werden, sondern auch eine Speicherzelle, die um einen bestimmten Betrag von dieser Adresse entfernt ist.

Nr.	Label	Operation	Operand	Adresse	Opcode	Opcode Operand	Operand	Operand	Bemerkungen
1	START	LD	IX, BUFF	18ØØ	DD	21	B6	1F	Vorbereitung
2	EIN	CALL	ADRESS	18Ø4	CD	ØD	18		(HL) = Eingabe
3		LD	A, (HL)	18Ø7	7E				A ← (HL)
4		CALL	DATADP	18Ø8	CD	71	Ø6		(A) anzeigen
5		JR	EIN	18ØB	18	F7			Nächste Adresse

Bild S 46.1
Befehlsfolge zum Versuch S 46.1. Zusätzlich müssen noch die im Bild S 46.2 aufgelisteten Bytes eingetastet werden.

S

46

Versuch S 46.1

Lesen einer Speicherzelle durch Adressierung mit (HL)

Dieser Versuch bildet stark vereinfacht die Funktion der Taste DATA nach: Sie tasten eine sedezimal vierstellig dargestellte Adresse ein und in der Anzeige erscheint der Inhalt der Speicherzelle mit der eingetasteten Adresse.

Laden Sie Ihr System zunächst ab der Adresse 18ØØ mit den Operationscodes und Operanden, die im Bild S 46.1 aufgelistet sind. In dieser Befehlsfolge wird ein Unterprogramm aus dem Betriebsprogramm verwendet. Außerdem werden zwei Unterprogramme aufgerufen, deren Bytes im Bild S 46.2 ab der Adresse 18ØD aufgelistet sind und die Sie zusätzlich eintasten müssen. Die Arbeitsweise dieser Unterprogramme ist hier uninteressant.

a)

18ØD CD 16 18 67 CD 16 18 6F
1815 C9

b)

1816 E5 C5 CD FE Ø5 Ø7 Ø7 Ø7
181E Ø7 4F CD FE Ø5 B1 C1 E1
1826 C9

Bild S 46.2
Die in den Teilbildern a und b aufgelisteten Bytes müssen für den Versuch S 46.1 zusätzlich zu den Befehlen im Bild S 46.1 eingegeben werden.

Starten Sie das Programm bei der Adresse 18ØØ! In der Anzeige erscheint links die Startadresse Ihres Programms und rechts das bei dieser Adresse abgelegte Byte DD. – Nach den folgenden Eingaben wird sich die Anzeige dieser Startadresse nicht ändern, und das soll Sie nicht kümmern. Sehen Sie einfach darüber hinweg.

Tasten Sie nun nacheinander die vier Ziffern einer beliebigen Adresse ein, zweckmäßig zunächst die Adresse 18ØD durch die aufeinanderfolgende Betätigung der Tasten 1, 8, 0 und D. Diese Eingabe wird in der Anzeige nicht quittiert, denn es geht uns in diesem Programm nur um die Funktion des Befehls LD A,(HL), und wir haben – im Gegensatz zur Funktion der Tasten ADDR und DATA, bei denen die Eingaben quittiert werden – auf jeden überflüssigen Komfort verzichtet.

Sofort nach der Eingabe der letzten der vier Ziffern für die Adresse erscheint an den beiden rechten Anzeigestellen das Byte CD. Sie können sich im Bild S 46.2a davon überzeugen, daß dieses das Byte ist, daß Sie vorher als Befehl bei der Adresse 18ØD eingegeben haben.

Tasten Sie beliebige andere Adressen ein: Jeweils nach der Eingabe der letzten Adreß-Ziffer erscheint in der Anzeige das bei dieser Adresse im Speicher stehende Byte.

Den Aufbau unseres Hauptprogramms können Sie leicht durchschauen. Der Befehl Nr. 1 dient der Vorbereitung des Programmablaufs und ist hier nicht von Interesse (vgl. Versuch H 32.1). – Der Befehl Nr. 2 bringt die von Ihnen eingegebenen vier Ziffern als zwei Bytes in das HL-Registerpaar. Sehen Sie sich das an:

RS	Programm anhalten.
ADDR, 1, 8, 0, 7	Bei dieser Adresse soll das Programm beim nächsten Lauf angehalten werden.
SBR	Die Taste SBR ist die von links gezählt vierte in der oberen Reihe des Tastenfeldes. Sie bewirkt das Anhalten des Programms bei der vorher eingegebenen Adresse.
ADDR, 1, 8, 0, 0, GO	Start des Programms.
1, 9, A, B	Eingabe der Adresse 19AB
Anzeige: 1808 CD	Dieser Befehl ist nicht mehr ausgeführt worden.
REG, HL	Abfrage des Inhalts des HL-Registerpaares
Anzeige: 19AB HL	(HL) = 19AB

Der folgende Befehl Nr. 3, LD A,(HL), ist der, auf den es uns eigentlich ankommt: Der Inhalt der vom HL-Registerpaar adressierten Speicherzelle wird in den Akkumulator kopiert. Wenn Sie wollen, können Sie sich durch eine ähnliche Tastenfolge wie die eben beschriebene davon überzeugen, daß nach diesem Befehl tatsächlich der Inhalt der adressierten Speicherzelle im Akkumulator steht. Wir verzichten auf diese Beschreibung, denn das mit dem Befehl Nr. 4 aufgerufene Unterprogramm macht den Inhalt des Akkumulators an den rechten beiden Anzeigestellen sichtbar. – Der letzte Befehl sorgt dafür, daß das Programm in einer Schleife läuft, so daß Sie verschiedene Adressen nacheinander aufrufen können.

Direkte und indirekte Adressierung von Speicherzellen

Die soeben vorgestellten LD r,(HL)-Befehle entsprechen grundsätzlich dem LD A,(adr)-Befehl: Daten aus der Speicherzelle mit der Adresse adr werden in ein CPU-Register kopiert. Mit dem LD A,(adr)-Befehl können allerdings nur Daten aus dem Speicher in den Akkumulator kopiert werden; mit den LD r,(HL)-Befehlen kann die Kopie in jedes beliebige der Register A bis L erfolgen.

Der wesentliche Unterschied ist jedoch ein anderer. Bei beiden Befehlen muß eine ganz bestimmte Speicherzelle adressiert werden. Der Drei-Byte-Befehl LD A,(adr) adressiert eine Speicherzelle direkt, denn er enthält in seinen beiden Operanden selbst die Adresse der Speicherzelle. Es handelt sich um eine **direkte Adressierung**.

Die Ein-Byte-Befehle LD r,(HL) dagegen haben gar keinen Platz für eine Adresse. Sie holen sich diese Adresse aus einem eigenen Memory-Pointer, nämlich aus dem HL-Registerpaar. Die betreffende Speicherzelle wird **indirekt adressiert**.

	Direkte Adressierung	Indirekte Adressierung
Speicher → CPU	LD A,(adr)	LD r,(HL) LD A,(BC) / LD A,(DE)
CPU → Speicher	LD (adr),A	LD (HL),r LD (BC),A / LD (DE),A

Drei dieser Befehle kennen Sie bereits; die restlichen fünf wollen wir Ihnen anschließend vorstellen.

Die Befehle LD (HL),r bilden das Gegenstück mit indirekter Adressierung zum Befehl LD (adr),A. Diese Befehle können jedoch mit Hilfe der im Memory-Pointer HL vorliegenden Adresse die Inhalte beliebiger CPU-Register r im Speicher ablegen:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD (HL),A	77	(HL) ← r	Inhalt des Registers r in der Speicherzelle ablegen, die durch den Inhalt des Registerpaares HL adressiert ist.
LD (HL),B	70		
LD (HL),C	71		
LD (HL),D	72		
LD (HL),E	73		
LD (HL),H	74		
LD (HL),L	75		

Die Verwendung dieser Befehle können Sie sich nachher in einem Versuch ansehen.

Eine Befehlsübersicht

Sicher haben Sie bereits jetzt das Gefühl, vor einer so großen Vielfalt an Befehlen zu stehen, daß Sie sie niemals werden behalten oder gar sinnvoll anwenden können. Dabei ist es sehr einfach.

LD r,r' Bild H30.1
LD r,(HL) Seite S45
LD (HL),r Seite S48

LD r,Konst Seite H28
LD (HL),Konst Seite S52
LD rp,Konst Seite H36

EX DE,HL Seite H38

LD A,(rp) Seite S51
LD (rp),A Seite S51

LD (adr),HL Seite S54
LD HL,(adr) Seite S54

Machen Sie sich zunächst von der Vorstellung frei, Sie müßten all' die vielen Operationscodes im Kopf behalten! Einige dieser Codes prägen sich zwar automatisch ein. Aber alle? Nein, die sieht man in einer Tabelle nach.

Bei den bisher vorgestellten Befehlen handelt es sich ausschließlich um Verwaltungs-Befehle, mit denen Bytes in der CPU umeinandergeschoben werden und die durch den mnemonischen Code LD gekennzeichnet sind.

Wenn Sie die Operationscodes vergessen und nur in mnemonischen Codes denken, dann wird die Sache sofort sehr einfach. Wenn wir ein CPU-Register mit r und ein Registerpaar mit rp bezeichnen, dann schrumpft die große Anzahl der bisher vorgestellten Befehle auf die nebenstehende Liste zusammen. Diese mnemonischen Codes sollten Sie sich allerdings merken.

Speicherzugriff mit indirekter Adressierung

Mit Speicherzugriff soll hier ganz allgemein die Kommunikation der CPU mit dem Speicher bezeichnet werden.

Die im folgenden Versuch verwendete Befehlsfolge ist programmtechnisch äußerst ungeschickt. Trotzdem soll sie zur Erläuterung des Befehls LD (HL),r verwendet werden.

Nr.	Label	Operation	Operand	Adresse	Opcode	Opcode Operand	Operand	Operand	Bemerkungen
1	START	LD	A, Ø1	18ØØ	3E	Ø1			(A) = Ø1
2		LD	BC, 2345	18Ø2	Ø1	45	23		(B) = 23, (C) = 45
3		LD	HL, 19ØØ	18Ø5	21	ØØ	19		19ØØ ← (A)
4		LD	(HL), A	18Ø8	77				
5		LD	HL, 19Ø1	18Ø9	21	Ø1	19		19Ø1 ← (C)
6		LD	(HL), C	18ØC	71				
7		LD	HL, 19Ø2	18ØD	21	Ø2	19		19Ø2 ← (B)
8		LD	(HL), B	1810	7Ø				
9		RST	Ø	1811	C7				System

Bild S49.1
Mit dieser Befehlsfolge zum Versuch S49.1 werden die Inhalte der Register A, B und C im Speicher abgelegt.

Versuch S49.1

Register-Inhalte in aufeinanderfolgenden Speicherzellen

Tasten Sie bitte die Operationscodes und Operanden der Befehlsfolge aus dem Bild S49.1 in Ihr System ein. – Löschen Sie anschließend die Inhalte der Speicherzellen mit den Adressen 19ØØ bis 19Ø2 (Tasten ADDR, 1, 9, 0, 0, DATA, 0, +, 0...).

Das Programm enthält ausschließlich Ihnen bereits bekannte Befehle. Der Befehl RST Ø beendet das Programm so, als wenn Sie im richtigen Augenblick die RS-Taste betätigen (vgl. Aufgabe S44.1). Der mnemonische Code RST kommt vom englischen **Re**S**T***art* (sprich: risstart) und bedeutet: Neuer Start (des Betriebsprogramms) (System-Meldung).

Die Bemerkungen in der Programm-Liste erläutern den Ablauf des Programms fast hinreichend: Mit den ersten beiden Befehlen werden der Akkumulator und die Register B und C mit den Werten Ø1, 23 und 45 geladen. Es folgen drei Gruppen mit jeweils zwei Befehlen. Dabei lädt der erste Befehl das HL-Registerpaar mit der Adresse der Speicherzelle, in welcher der Register-Inhalt abgelegt werden soll. Der dann folgende Befehl LD (HL),r besorgt die eigentliche Ablage.

Starten Sie das Programm und sehen Sie sich nach der anschließenden System-Meldung die Inhalte der im Bild S49.2 dargestellten Speicherzellen 19ØØ bis 19Ø2 an! Erst jetzt wird Ihnen wahrscheinlich auffallen, daß die Register-Inhalte in den aufeinanderfolgenden Speicherzellen keineswegs in der Reihenfolge A, B, C abgelegt sind. Der Akkumulator-Inhalt steht zwar an erster Stelle; der Inhalt des BC-Registerpaares ist jedoch in der Reihenfolge C, B abgelegt.

Wir sind bei der Ablage ganz bewußt dem Prinzip gefolgt, das der Natur unseres Mikroprozessors entspricht: Er legt grundsätzlich das niederwertige Byte (LOB) eines Registerpaares bei der niedrigeren Adresse und das höherwertige Byte (HOB) bei der höheren Adresse im Speicher ab. Das Prinzip gilt auch für Adressen, die als Operanden von direkt adressierenden Befehlen im Speicher abgelegt werden (vgl. Bild S43.1).

Hier liegt die Erklärung für die aus dem Rahmen der übrigen Register-Bezeichnungen fallenden Namen der Register H und L. Wenn diese beiden Register als Registerpaar im Sinne eines Memory-Pointers benutzt werden, dann steht im **H**-Register das **H**OB einer Adresse und im **L**-Register das **L**OB.

Akkumulator		Ø1	19ØØ
Register-	LOB	45	19Ø1
paar B,C	HOB	23	19Ø2

Bild S49.2
So werden die Inhalte des Akkumulators und eines Registerpaares im Speicher abgelegt.

Indirekte Adressierung über die Registerpaare BC und DE

Die Zusammenfassung der H- und L-Register zu einem Registerpaar erweist sich für die Verwendung als Memory-Pointer als sehr zweckmäßig: Mit dem LD HL,Konst-Befehl läßt sich mit einem Schlag eine indirekt zu adressierende Speicherzelle bestimmen.

Auch die Zusammenfassung der D- und E-Register zum Registerpaar DE hat in diesem Zusammenhang einen Sinn: Wenn in das DE-Registerpaar eine Adresse geladen wird, dann kann diese Adresse mit dem Befehl EX DE,HL (Seite H38) in den Memory-Pointer gebracht und zur indirekten Adressierung einer Speicherzelle herangezogen werden. Dabei geht die vorher im Memory-Pointer stehende Adresse nicht einmal verloren: Sie wird im DE-Registerpaar aufbewahrt (vgl. Bild H38.1).

Für die Zusammenfassung der B- und C-Register zu einem Registerpaar haben wir bisher noch keine Begründung gefunden. Sie ergibt sich sofort, wenn wir die Aufstellung der Befehle für die Kommunikation der CPU mit dem Speicher noch einmal etwas ausführlicher wiederholen. Dabei interessieren wir uns für die bisher noch nicht besprochenen Befehle LD A,(BC), LD A,(DE), LD (BC),A und LD (DE),A.

Adressierung	Direkt	Indirekt	
		über BC oder DE	über HL
Speicher → Akkumulator	LD A,(adr)	LD A,(BC) LD A,(DE)	
Speicher → Register			LD r,(HL)
Akkumulator → Speicher	LD (adr),A	LD (BC),A LD (DE),A	
Register → Speicher			LD (HL),r

Die Aufstellung zeigt, daß bei Verwendung der direkt adressierenden Befehle der Datentransport vom und zum Speicher stets über das zentrale Register Akkumulator läuft. Soll bei direkter Adressierung ein Byte zwischen dem Speicher und einem der Register B...L transportiert werden, dann muß zusätzlich zum LD A,(adr)- oder zum LD (adr),A-Befehl noch ein LD r,A- bzw. ein LD A,r'-Befehl programmiert werden.

Anders bei Verwendung der indirekt adressierenden LD r,(HL)- und LD (HL),r-Befehle: Mit diesen Befehlen kann ein Byte direkt zwischen einem beliebigen CPU-Register und dem Speicher transportiert werden, wenn die Adresse der betreffenden Speicherzelle in dem als Memory-Pointer arbeitenden HL-Registerpaar steht.

Die Befehle LD A,(BC) und LD A,(DE) bzw. LD (BC),A und LD (DE),A nehmen eine Mittelstellung ein. Es handelt sich ebenfalls um indirekt adressierende Ein-Byte-Befehle. Dabei kann aber die Adresse der betreffenden Speicherzelle im BC- oder DE-Registerpaar stehen. Die Verwandtschaft zu den direkt adressierenden LD A,(adr) und LD (adr),A besteht darin, daß der Datenverkehr nur zwischen Akkumulator und Speicher möglich ist; der unmittelbare Verkehr mit einem beliebigen CPU-Register ist nicht möglich.

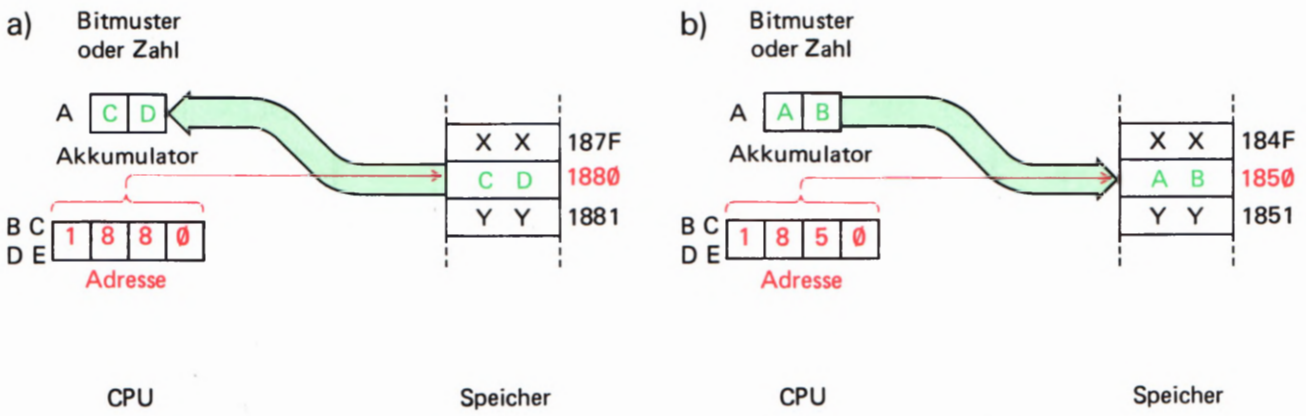


Bild S51.1
Auch die Inhalte der BC- und DE-Registerpaare können Adressen von Speicherzellen sein.
a) Befehl LD A,(rp);
b) Befehl LD (rp),A.

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD A,(BC) LD A,(DE)	ØA 1A	A ← (BC) A ← (DE)	Inhalt der Speicherzelle, die durch (BC) bzw. (DE) adressiert ist, in den Akkumulator kopieren.
LD (BC),A LD (DE),A	Ø2 12	(BC) ← A (DE) ← A	Inhalt des Akkumulators in die Speicherzelle kopieren, die durch (BC) bzw. (DE) adressiert ist.

Das Bild S51.1 macht die Wirkungweise der Befehle LD A,(rp) und LD (rp),A deutlich. Im Teilbild a zeigt der Inhalt eines der Registerpaare BC oder DE auf die Speicherzelle mit der Adresse 188Ø, deren Inhalt in den Akkumulator kopiert wird (LD A,(rp)). – Im Teilbild b ist der umgekehrte Vorgang mit dem Befehl LD (rp),A dargestellt: Der Inhalt des Registerpaares zeigt auf die Speicherzelle mit der Adresse 185Ø, und in diese Speicherzelle wird der Inhalt des Akkumulators kopiert.

Hingewiesen sei noch auf eine Besonderheit der hier vorgestellten Befehle: Sie gelten nur für die Registerpaare BC und DE, nicht aber für das Registerpaar HL. Das ist auch sinnvoll, denn wenn das HL-Registerpaar verfügbar ist, dann wird man die universelleren Befehle LD r,(HL) bzw. LD (HL),r verwenden.

Versuch S51.1

Die Befehle LD (rp),A und LD A,(rp)

Löschen Sie zunächst die Anzeigen der Register-Inhalte und laden Sie dann über das Tastenfeld die Speicherzelle mit der Adresse 188Ø mit dem Byte CD! – Tasten Sie anschließend die im Bild S52.1 aufgelisteten Befehle ein. – Wenn Sie nach der Eingabe des letzten Befehls mit dem Operationscode 1A noch einmal die Taste + betätigen, dann erscheint in der Anzeige die Adresse 18ØA. Setzen Sie bei dieser Adresse, die keinen Befehl unserer Befehlsfolge mehr enthält, mit der Taste SBR einen Breakpoint, der das Bearbeiten weiterer Befehle verhindert (vgl. Seite S26).

Nr.	Label	Operation	Operand	Adresse	Opcode	Opcode Operand	Operand	Operand	Bemerkungen
1	START	LD	BC, 185Ø	18ØØ	Ø1	5Ø	18		(BC) = 185Ø
2		LD	DE, 188Ø	18Ø3	11	8Ø	18		(DE) = 188Ø
3		LD	A, AB	18Ø6	3E	AB			(A) = AB
4		LD	(BC), A	18Ø8	Ø2				(A) → 185Ø
5		LD	A, (DE)	18Ø9	1A				A ← (188Ø)

Bild S52.1
Befehlsfolge zum Versuch S51.1 mit den Befehlen LD (rp),A und LD A,(rp).

Mit den in der Auflistung angegebenen Bemerkungen bedarf die Befehlsliste keines Kommentars. – Starten Sie das Programm bei der Adresse 18ØØ und sehen Sie sich nach dem Erscheinen der Adresse 18ØB mit den Tasten REG und AF den Inhalt des Akkumulators an! Sie finden das Byte CD, das der Befehl Nr. 5, LD A,(DE) aus der mit dem Registerpaar DE adressierten Speicherzelle geholt hat.

Kontrollieren Sie jetzt den Inhalt der Speicherzelle 185Ø, auf die das BC-Registerpaar zeigt! Sie finden dort das Byte AB, das Sie per Programm in den Akkumulator gebracht und von dort mit dem Befehl LD (BC),A in den Speicher kopiert haben.

Unmittelbares Laden des Speichers mit indirekter Adressierung

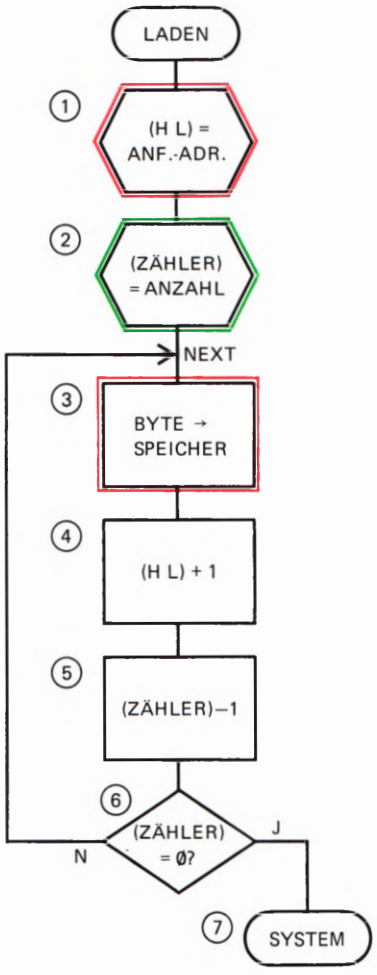


Bild S52.2
Ablaufplan eines Programms zum Laden eines Speicherbereichs mit einem bestimmten Byte.

Sicher wird Sie inzwischen die in der Überschrift benützte Fachsprache nicht mehr schrecken. Gemeint ist ja auch etwas recht einfaches: Mit einem indirekt adressierenden Befehl soll ein bestimmtes Byte (und nicht der Inhalt eines Registers!) in einer Speicherzelle abgelegt werden.

Eine ähnliche Aufgabe ist es, ein CPU-Register mit einem konstanten Byte zu laden. Sie erinnern sich: Das geschieht mit einem Befehl LD r,Konst (Seite H28).

Der mnemonische Code des hier interessierenden Befehls ist ähnlich:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD (HL),Konst	36	(HL) ← Konst.	Die vom HL-Registerpaar adressierte Speicherzelle wird mit Konst. geladen.

Es handelt sich um einen Zwei-Byte-Befehl, in dem das zweite Byte dasjenige ist, das in der vom HL-Registerpaar adressierten Speicherzelle abgelegt werden soll.

Die Wirkungsweise dieses Befehls können Sie an einem Programm erkennen, das als Programmbaustein recht nützlich sein kann: Es lädt bis zu 255 beliebige, aufeinanderfolgende Speicherzellen mit einem bestimmten Byte. Häufig ist dieses Byte das Lösch-Byte ØØ.

Das Bild S52.2 zeigt die Funktion des Programms als Programmablaufplan. Uns geht es dabei vor allem um die rot umrandeten Anweisungen. Die zugehörigen Befehle sind Ihnen bekannt; die übrigen Befehle stellen wir bei dieser Gelegenheit gleich mit vor.

Nr.	Label	Operation	Operand	Adresse	Opcode	Opcode Operand	Operand	Operand	Bemerkungen
1	LADEN	LD	HL,ANFA	1F90	21	00	18		(HL) = Anfangsadresse
2		LD	B,ANZAHL	1F93	06	10			(Zähler) = Anzahl
3	NEXT	LD	(HL),BYT	1F95	36	AB			Byte → Speicher
4		INC	HL	1F97	23				(HL) → nächste Zelle
5		DEC	B	1F98	05				(Zähler) – 1
6		JP	NZ,NEXT	1F99	C2	95	1F		(Zähler) = 00?
7		RST	0	1F9C	C7				System

Bild S53.1
Mit dieser Befehlsfolge wird ein Speicherbereich mit einem Byte geladen. Der Anfang des Bereichs wird im HL-Registerpaar abgelegt. Im B-Register steht die Anzahl der zu ladenden Speicherzellen. Das Byte ist im Befehl LD (HL),Konst enthalten.

Versuch S53.1

Speicherbereich mit Byte laden

Tasten Sie bitte die im Bild S53.1 aufgelisteten Befehle ab der Adresse 1F90 in Ihr System ein! Weil sich die Befehlsfolge als Hilfsprogramm beim Programmieren verwenden läßt, ordnen wir sie in einem Speicherbereich an, der zunächst nur selten benutzt wird.

Sehen Sie sich den derzeitigen Inhalt des Schreib-Lese-Speichers ab der Adresse 1800 bis zur Adresse 1815 an (ADDR, 1, 8, 0, 0, +, +...)! – Starten Sie jetzt das Programm bei der Adresse 1F90! Wenn Sie sich nach der System-Meldung noch einmal den vorher betrachteten Speicherbereich ansehen, dann stellen Sie fest, daß in den Bereich der Adressen 1800 bis 180F gerade soviele Bytes AB geladen wurden, wie es dem Inhalt des B-Registers entspricht (10H = 16).

Die wichtigste Anweisung der Befehlsfolge ist die mit der Nummer 3 (Bilder S52.2 und S53.1); sie bringt mit dem indirekt adressierenden Befehl LD (HL),Konst ein Byte in die Speicherzelle, deren Adresse mit der Anweisung Nr. 1 im HL-Registerpaar abgelegt wird.

Bei einmaligem Abarbeiten der Anweisung Nr. 3 würde nur eine einzige Speicherzelle mit dem gewünschten Byte geladen. Damit mehr als eine einzige Speicherzelle geladen wird, läßt man das Programm in einer Schleife (Seite S5) laufen, so daß die Anweisung immer wieder ausgeführt wird. Das hätte jedoch zunächst zur Folge, daß das gleiche Byte immer wieder in dieselbe Speicherzelle gebracht würde. Deshalb sorgt die Anweisung Nr. 4 dafür, daß der Inhalt des HL-Registerpaares bei jedem Schleifendurchlauf um eins erhöht wird, so daß jedesmal die nächstfolgende Speicherzelle, also die mit einer um eins höheren Adresse, adressiert wird.

Solches Erhöhen des Inhalts eines Registers (oder auch eines Registerpaares oder einer Speicherzelle) um eins bezeichnet man in der Fachsprache als **Inkrementieren**; umgekehrt bezeichnet man entsprechendes Erniedrigen als **Dekrementieren**. Der Befehlssatz unseres Mikroprozessors stellt dafür folgende Befehle zur Verfügung:

Mnemonischer Code	Operation	Erläuterung
INC r INC rp INC (HL)	(r) ← (r)+1 (rp) ← (rp)+1 (Zelle) ← (Zelle)+1	Der Inhalt eines Registers, eines Registerpaares oder einer vom HL-Registerpaar adressierten Speicherzelle wird inkrementiert.

Mnemonischer Code	Operation	Erläuterung
DEC r DEC rp DEC (HL)	(r) ← (r)–1 (rp) ← (rp)–1 (Zelle) ← (Zelle)–1	Der Inhalt eines Registers, eines Registerpaares oder einer vom HL-Registerpaar adressierten Speicherzelle wird dekrementiert.

Die Operationscodes dieser Befehle zeigt die folgende Tabelle:

	A	B	C	D	E	H	L	BC	DE	HL	Speicherzelle
INC	3C	04	0C	14	1C	24	2C	03	13	23	34
DEC	3D	05	0D	15	1D	25	2D	0B	1B	2B	35

In unserem Programm sorgen die Anweisungen Nr. 2 und Nr 5 dafür, daß die Schleife nur so oft durchlaufen wird, wie es der (sedezimal dargestellte) Inhalt des B-Registers bestimmt. Das B-Register bildet also einen **Zähler**. Mit der Anweisung Nr. 5 wird der Inhalt dieses Zählers bei jedem Schleifendurchlauf dekrementiert. Solange er noch nicht bei 0 angekommen ist, wird die Schleife ein weiteres Mal durchlaufen. Erst wenn nach dem Dekrementieren (B) = 00 ist, wird die Schleife verlassen und das Programm abgebrochen. (Wir werden Ihnen später erläutern, wie dieser Mechanismus funktioniert.)

Aufgabe S 54.1

Welche Änderungen müssen Sie in der im Bild S 53.1 aufgelisteten Befehlsfolge vornehmen, wenn Sie ab der Adresse 1900 einhundert Speicherzellen löschen wollen?

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 9.

Das Aufbewahren von Speicheradressen im Speicher

Zum Schluß dieses Abschnitts wollen wir Ihnen noch Verwaltungs-Befehle vorstellen, die dann nützlich sind, wenn man in einem Programm zwischenzeitlich die Adressen von Speicherzellen aufbewahren will, zu denen ein späterer Zugriff notwendig ist. Ein solcher Zugriff erfolgt mit indirekter Adressierung. Das bedeutet: Der Inhalt des HL-Registerpaares muß in den Speicher gebracht und dort auch wieder abgeholt werden können. Dazu dienen die folgenden beiden Befehle:

Mnemonischer Code	Operations-code	Operation	Erläuterung
LD (adr),HL	22	adr ← (HL)	(HL) in den Speicherzellen adr und adr+1 ablegen.
LD HL,(adr)	2A	(HL) ← adr	HL mit dem Inhalt der Speicherzellen adr und adr+1 laden.

Der Befehlssatz unseres Mikroprozessors

Befehle haben naturgemäß nur dann einen Sinn, wenn etwas da ist, dem man Befehle erteilen kann. Für unseren Lehrgang ist der Befehlsempfänger der Mikroprozessor. Es wird per Befehl angewiesen, irgendwelche Daten zu bearbeiten.

Sinnvoll können Befehle nur dann sein, wenn man über die Struktur des Befehlsempfängers einigermaßen Bescheid weiß. Aus diesem Grunde haben wir Sie mit den ersten Befehlen aus dem Befehlssatz unseres Mikroprozessors im Abschnitt Hardware bekannt gemacht, obwohl gerade die Befehle in den Abschnitt Software gehören.

Im folgenden Abschnitt stellen wir Ihnen zunächst eine Übersicht über die verfügbaren Befehle vor und befassen uns dann mit den Befehlen, die über die reine Verwaltung von Daten hinausgehen.

Die Befehlsarten für den Mikroprozessor

Die im Befehlssatz eines Mikroprozessors verfügbaren Befehle lassen sich unter den folgenden sechs Stichworten zusammenfassen:

1. Verwaltung von Daten
2. Rechnen (arithmetische Verknüpfungen)
3. Verknüpfungen (logische Verknüpfungen)
4. Sprünge
5. Unterprogramme
6. Sonderbefehle

Die wichtigsten der zur ersten Gruppe gehörenden Befehle haben wir bereits auf der Seite S48 zusammengestellt. Diese Befehle haben keine andere Aufgabe, als die in Registern oder Speicherzellen abgelegten Daten innerhalb des Systems zu transportieren. Dabei handelt es sich nicht eigentlich um Transport-Befehle, sondern um Kopier-Befehle. Die zunächst einzige Ausnahme bildet der Austausch-Befehl EX DE,HL (Seite H38), der Daten tatsächlich aus dem einen Registerpaar weg in ein anderes transportiert.

Die **Rechenbefehle** kommen offenbar der häufig benutzten Bezeichnung Mikro**computer** (*compute* = rechnen) am meisten entgegen. Rechenbefehle bewirken die arithmetische Verknüpfung von Daten, die im System als Zahlen interpretiert werden. Diese Befehle werden jedoch entgegen ihrer scheinbaren Bedeutung keineswegs besonders häufig verwendet. Ein Mikroprozessor-System wird doch verhältnismäßig selten als reine Rechenmaschine benutzt.

Die im Befehlssatz verfügbaren Rechenbefehle sind im wesentlichen Anweisungen zur Addition und bei etwas komfortableren CPUs auch einige Subtraktions-Anweisungen. Damit ist der Vorrat an Rechenbefehlen meist aber auch schon erschöpft. Kompliziertere Rechenarten (Multiplizieren, Dividieren usw.) müssen mit Additions- und Subtraktionsbefehlen konstruiert werden.

Eigentlich von größerer Bedeutung als die Rechenbefehle sind beim Erstellen von Programmen die Befehle zur **logischen Verknüpfung**

von Daten, die im System als Bitmuster interpretiert werden. Man versteht darunter die UND-Verknüpfung, die ODER-Verknüpfung und – von besonderer Bedeutung bei Mikroprozessoren – die Exklusiv-ODER-Verknüpfung. Diese Befehle haben wir bereits bei den Versuchen S 25.1 bis S 30.1 verwendet. Bei Mikroprozessoren haben diese Befehle jedoch eine etwas andere Bedeutung als die einfache Signalverknüpfung in digitalen Hardware-Schaltungen. Wir kommen auf diese Bedeutung noch zurück.

In der landläufigen Vorstellung ist der Begriff Intelligenz recht eng mit dem Begriff Rechnen gekoppelt. Es sei dahingestellt, ob man einem Computer und damit auch einem Mikroprozessor-System zu Recht Intelligenz zuspricht (was eine Frage der Definition von Intelligenz ist). Spricht man ihm aber wirklich eine Art von Intelligenz zu, dann hängt diese Intelligenz sicher nicht mit den im Befehlssatz verfügbaren Rechenbefehlen zusammen. Rechnen kann auch ein normaler Taschenrechner und es soll sogar Pferde geben, denen man das Wurzelziehen beigebracht hat. Intelligenz möchte man aber wohl weder einem Pferd noch einem Taschenrechner zusprechen.

Die „Intelligenz" eines Mikroprozessor-Systems ist in der Tatsache begründet, daß es folgerichtige Entscheidungen treffen kann. Eine Entscheidung bedeutet hier die Auswahl zwischen zwei möglichen Verhaltensweisen. Und genau das kann der Mikroprozessor, wenn sich im Ablaufplan des Programms eine Verzweigung befindet (Seite S 5).

Eine Verzweigung wird immer mit einem Befehl für einen bedingten Sprung programmiert. Man darf also feststellen, daß die Gruppe der **Sprungbefehle**, zu der auch Befehle für unbedingte Sprünge gehören, die Befehle enthält, die dem Mikroprozessor zu seiner Bedeutung verholfen haben.

Über das Prinzip der **Unterprogramme** haben Sie bereits auf den Seiten S 5 und S 6 gelesen. Der Aufruf eines Unterprogramms kann für den Mikroprozessor eine Reihe von Problemen aufwerfen, auf die wir noch zurückkommen. Der Befehlssatz unseres Mikroprozessors enthält eine Reihe von Befehlen, die Anweisungen für sehr elegante Lösungen dieser Probleme bilden.

Die Sonderbefehle des Befehlssatzes entsprechen etwa dem Ordner „Verschiedenes", der in keinem Büro fehlen darf. Diese Befehle passen in keine der Rubriken der oben genannten Befehle. Wir werden diese oft nützlichen Befehle da vorstellen, wo sie uns gerade begegnen.

Befehle zur logischen Exklusiv-ODER-Verknüpfung

Eine Verknüpfung bedeutet – sehr einfach ausgedrückt – das Entstehen eines neuen Wertes aus zwei verschiedenen oder auch gleichen anderen Werten. Der sich einstellende, neue Wert hängt – außer von den beiden anfänglich vorliegenden Werten – von der Art der gewählten Verknüpfung ab.

Sind die beiden anfangs vorliegenden Werte z. B. die Zahlen Sechs und Zwei, dann können diese beiden Werte addierend, subtrahierend,

multiplizierend oder dividierend oder auch noch auf kompliziertere Art und Weise miteinander verknüpft werden. Das Verknüpfungs-Ergebnis ist jedesmal ein anderes.

Daß diese arithmetischen Verknüpfungen keineswegs die einzig möglichen Verknüpfungen sind, mußten viele Eltern mit Schrecken erkennen, wenn ihr Kind in der Schule mit der Mengenlehre Bekanntschaft machte. Es lernte dort konjunktive (UND-) und disjunktive (ODER-) Verknüpfungen kennen, wenn diese Verknüpfungen in der Schule vielleicht auch etwas andere Namen haben. Solche Verknüpfungen bezeichnet man als **logische Verknüpfungen**. Das bedeutet nun keineswegs , daß etwa eine addierende Verknüpfung unlogisch ist. Wir wollen uns hier jedoch über die Bezeichnung keine Gedanken machen und nur feststellen, daß logische Verknüpfungen für uns im Zusammenhang mit Bitmustern in Registern und Speichern interessant sind.

Das logische Verknüpfen von binären Werten im Mikroprozessor hat meist ein etwas anderes Ziel als die (im Prinzip gleichen) Verknüpfungen in digitalen Schaltgliedern. Beim Mikroprozessor liegt eher eine Ähnlichkeit mit der addierenden Verknüpfung vor: Er unternimmt die logischen Verknüpfungen stellenweise.

Diese Überlegung läßt bereits jetzt ahnen, daß die logische Verknüpfung der Inhalte von zwei Registern ein nichtssagendes Ergebnis liefert, wenn man die Register-Inhalte und das Verknüpfung-Ergebnis in sedezimaler Schreibweise betrachtet. Sinnvoll für das Verständnis wird die Sache erst bei der Betrachtung der Bitmuster.

Von den vielen, möglichen logischen Verknüpfungen sind beim Programmieren eigentlich nur drei interessant: Die UND-Verknüpfung, die ODER-Verknüpfung und die Exklusiv-ODER-Verknüpfung. Für diese Verknüpfungen gibt es eigene Befehle, die wir Ihnen vorstellen wollen. Dabei wird sich herausstellen, daß sich diese Befehle in überraschender Weise nutzbringend einsetzen lassen.

Als erste Verknüpfungsart wollen wir die **Exklusiv-ODER-Befehle** vorstellen, obwohl diese Verknüpfung in der digitalen Hardware fast ein wenig exotisch ist. Sie werden sehen, daß mit dieser Verknüpfung z. B. elegant die Möglichkeit besteht, zwei Bitmuster auf Gleichheit oder Ungleichheit zu prüfen.

An der Ausführung eines Befehls zur logischen Verknüpfung müssen immer zwei Bitmuster zu je acht Bit beteiligt sein. Das eine dieser Bitmuster steht bei unserem Mikroprozessor immer im Akkumulator. Das zweite Bitmuster kann in einem der ansprechbaren Register A, B, C, D, E, H oder L oder aber in einer vom HL-Registerpaar indirekt adressierten Speicherzelle abgelegt sein. Die entsprechenden Befehle sind dann Ein-Byte-Befehle. – Das zweite Bitmuster kann aber auch eine Konstante sein, die vom Befehl selbst vorgegeben wird. Dieser Befehl ist dann ein Zwei-Byte-Befehl; die das Bitmuster sedezimal darstellende Konstante ist der Operand des Befehls.

Das Verknüpfungsergebnis steht nach der Ausführung des Verknüpfungsbefehls im Akkumulator. Es überschreibt (und vernichtet dadurch) das eine der anfangs vorliegenden Bitmuster.

Dies sind die hier interessierenden Exklusiv-ODER-Befehle für unseren Mikroprozessor:

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	START	LD	HL,1900	1800	21	00	19	(HL) ← 1900
2		LD	A,3D	1803	3E	3D		A ← 3D
3		LD	B,2B	1805	06	2B		B ← 2B
4		LD	(HL),4C	1807	36	4C		(1900) ← 4C
5		XOR	B	1809	A8			A ← (A) ≠ (B)
6		XOR	(HL)	180A	AE			A ← (A) ≠ (1900)
7		XOR	5A	180B	EE	5A		A ← (A) ≠ 5A

Bild S 58.1
Befehlsfolge zum Versuch S 58.1, in dem die Exklusiv-ODER-Befehle vorgestellt werden.

Mnemonischer Code	Operation	Erläuterung
XOR r XOR (HL)	A ← (A) ≠ (r) A ← (A) ≠ (HL)	Der Inhalt eines Registers oder einer vom HL-Registerpaar adressierten Speicherzelle wird mit dem Inhalt des Akkumulators Exklusiv-ODER-verknüpft.
XOR Konst	A ← (A) ≠ Konst	Der Inhalt des Akkumulators wird mit einem konstanten Wert Konst Exklusiv-ODER-verknüpft.

Die Operationscodes dieser Befehle zeigt die folgende Tabelle:

	A	B	C	D	E	H	L	Speicherzelle	Konstante
XOR	AF	A8	A9	AA	AB	AC	AD	AE	EE

Die Wirkung der Exklusiv-ODER-Befehle sehen Sie sich am besten in einem Versuch an:

Versuch S 58.1

Exklusiv-ODER-Befehle

Tasten Sie bitte die im Bild S 58.1 aufgelistete Befehlsfolge ein! Mit den Befehlen Nr. 1 bis Nr. 4 werden der Akkumulator und das B-Register sowie die vom HL-Registerpaar adressierte Speicherzelle 1900 mit konstanten Werten geladen.

Die Befehle Nr. 5 bis Nr. 7 kennen Sie aus der vorangegangenen Aufstellung; Sie wissen allerdings noch nicht, was diese Befehle bewirken. Da das Ergebnis dieser Befehle jeweils im Akkumulator steht, lassen Sie diese Befehle zweckmäßig in Einzelschritten ablaufen und schauen sich nach der Ausführung des Befehls jeweils an, was im Akkumulator steht. – Lassen Sie zunächst die ersten vier Befehle abarbeiten:

ADDR, 1, 8, 0, 0, STEP, STEP, STEP, STEP

Die Betätigung der Tasten REG, AF, BC und HL zeigt Ihnen die Inhalte (A) = 3D, (B) = 2B und (HL) = 1900. – Überzeugen Sie sich noch davon, daß in der vom HL-Registerpaar adressierten Speicherzelle 1900 der Wert 4C steht: ADDR, 1, 9, 0, 0. – Betätigen Sie anschließend

bitte die Taste PC, um Ihr System mit der Anzeige der Adresse 18Ø9 auf die Ausführung des Befehls Nr. 5 vorzubereiten.

Lassen Sie jetzt den Befehl Nr. 5, XOR B in einem Einzelschritt ausführen und notieren Sie anschließend den Inhalt des Akkumulators:

STEP, REG, AF Anzeige: (A) = 16

Die jetzt ausgeführte Verknüpfung haben wir zusammen mit dem Verknüpfungsergebnis nebenstehend angeschrieben. Diese Gleichung ist wegen der sedezimalen Darstellung zunächst nichtssagend. 3D ⩛ 2B = 16

Ehe wir uns ansehen, warum die Exklusiv-ODER-Verknüpfung der Bytes 3D und 2B das Byte 16 im Akkumulator bewirkt, sollen zunächst die Ergebnisse der folgenden Befehle festgestellt werden. Denken Sie bitte daran: Jetzt ist der Inhalt des Akkumulators (A) = 16! – Lassen Sie den Mikroprozessor den nächsten Einzelschritt ausführen und sehen Sie sich anschließend wieder den Inhalt des Akkumulators an:

PC
STEP, REG, AF Anzeige: (A) = 5A

Mit dem Befehl XOR (HL) ist der Inhalt 4C der Speicherzelle, die das HL-Registerpaar mit 19ØØ adressiert hat, mit dem Akkumulator-Inhalt 16 Exklusiv-ODER-verknüpft worden. Das Verknüpfungsergebnis steht wieder im Akkumulator; der vorherige Inhalt 16 ist verschwunden. – Lassen Sie auch den letzten Befehl als Einzelschritt ausführen, nachdem Sie vorher die Taste PC betätigt haben: 16 ⩛ 4C = 5A

PC
STEP, REG, AF Anzeige: (A) = ØØ

Der Befehl XOR 5A ist abgelaufen; er hat den Inhalt 5A des Akkumulators mit dem vom Befehl im Operanden vorgegebenen Wert Exklusiv-ODER-verknüpft. – Die nebenstehende Gleichung ist interessant: Die Exklusiv-ODER-Verknüpfung zweier gleicher Bytes ergibt das Byte ØØ. Natürlich könnte das ein Zufall sein. Es ist aber keiner. Hier erkennen Sie eine der wichtigsten Eigenschaften der Exklusiv-ODER-Befehle: 5A ⩛ 5A = ØØ

> Die Exklusiv-ODER-Verknüpfung von zwei gleichen Bytes ergibt das Byte ØØ.
>
> Die Exklusiv-ODER-Verknüpfung des Akkumulator-Inhalts mit sich selbst bei Verwendung des Befehls XOR A löscht den Akkumulator-Inhalt.

Der zweite Teil dieses Merksatzes ergibt sich aus dem ersten, da bei den Befehlen zur logischen Verknüpfung eines der beiden zu verknüpfenden Bytes grundsätzlich im Akkumulator steht. Mit dem Befehl XOR **A** wird dann das Byte im Akkumulator noch einmal angesprochen. Dieser Ein-Byte-Befehl XOR A kann also oft vorteilhaft anstelle des Zwei-Byte-Befehls LD A,ØØ verwendet werden. Dabei ist aber Vorsicht geboten: Im Gegensatz zum LD A,Konst-Befehl beeinflußt der XOR A-Befehl den Inhalt des Flag-Registers. Was es damit auf sich hat, werden wir später noch erwähnen.

Das Bild S 60.2 macht deutlich, wie es zu den merkwürdigen Ergebnissen des Versuchs S 58.1 kommt. Sie erinnern sich: Die logische Ver-

knüpfung von zwei Bytes ergibt für das Verständnis nur dann einen Sinn, wenn die Bytes als Bitmuster dargestellt werden und die Verknüpfung dann bitweise an den einzelnen Stellen vorgenommen wird.

Grundlage für die Exklusiv-ODER-Verknüpfung zweier Bits ist die Funktionstabelle, die im Bild S60.1 noch einmal dargestellt ist. Sie erkennen: Die Verknüpfung liefert immer dann ein Ø-Bit, wenn zwei gleichwertige Bits (Ø mit Ø oder 1 mit 1) verknüpft werden. Werden zwei ungleichwertige Bits verknüpft, dann ist das Ergebnis stets ein 1-Bit.

e_1	e_2	a
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

Bild S60.1
Funktionstabelle der Exklusiv-ODER-Verknüpfung.

Jetzt ist die Sache sehr einfach: Im Bild S60.2 ist die Ausführung der Befehle XOR B, XOR (HL) und XOR 5A bitweise skizziert. Sie sehen deutlich, daß nach Ausführung eines jeden Exklusiv-ODER-Befehls an den einzelnen Bit-Stellen des Akkumulators immer dann ein 1-Bit erscheint, wenn an den beiden darüberstehenden Stellen unterschiedliche Bits stehen. Ein Ø-Bit erscheint dann, wenn die beiden darüber angeordneten Bits gleich sind.

```
Nr.  7  6  5  4  3  2  1  0
A    Ø  Ø  1  1  1  1  Ø  1   3D
XOR B                      ≠
B    Ø  Ø  1  Ø  1  Ø  1  1   2B
-------------------------------
A    Ø  Ø  Ø  1  Ø  1  1  Ø   16
XOR (HL)                   ≠
M    Ø  1  Ø  Ø  1  1  Ø  Ø   4C
-------------------------------
A    Ø  1  Ø  1  1  Ø  1  Ø   5A
XOR 5A                     ≠
I    Ø  1  Ø  1  1  Ø  1  Ø   5A
-------------------------------
A    Ø  Ø  Ø  Ø  Ø  Ø  Ø  Ø   ØØ
```

Bild S60.2
Hier ist die Ausführung der Exklusiv-ODER-Befehle in der im Bild S58.1 aufgelisteten Befehlsfolge bitweise dargestellt.

Aufgabe S60.1

Welche Ergebnisse stellen sich bei folgenden Exklusiv-ODER-Verknüpfungen ein:

a) 2B ≠ EA b) 75 ≠ 1Ø c) AB ≠ AB d) 3C ≠ FF

Schreiben Sie die Ergebnisse in sedezimaler Schreibweise an!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü10.

Die Verwendbarkeit der Exklusiv-ODER-Befehle bei der Mikroprozessor-Programmierung zeigt die Lösung der Teilaufgaben c und d. Die Lösung der Teilaufgabe c führt zu folgender Feststellung, die Sie jetzt leicht verstehen:

> Die Exklusiv-ODER-Verknüpfung von zwei Bytes, deren eines im Akkumulator abgelegt ist, führt immer dann zum Akkumulator-Inhalt ØØ, wenn die beiden Bytes gleich sind.
>
> Sind die beiden Bytes untereinander ungleich, dann steht nach der Exklusiv-ODER-Verknüpfung im Akkumulator ein von ØØ verschiedenes Byte.

Die Lösung der Teilaufgabe d zeigt eine Verwendbarkeit der Exklusiv-ODER-Befehle, die Ihnen vielleicht zunächst gar nicht aufgefallen ist:

Versuch S60.1

Invertierung eines Bitmusters

Tasten Sie bitte die drei im Bild S61.1 aufgelisteten Befehle ein! Die ersten beiden Befehle brauchen wir nicht zu erläutern. Sie sorgen dafür, daß das Byte 5B im Akkumulator mit dem Byte FF Exklusiv-

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	START	LD	A,5B	18ØØ	3E	5B		(A) ← 5B
2		XOR	FF	18Ø2	EE	FF		(A) ← (A) ≠ FF
3		RST	3Ø	18Ø4	F7			Breakpoint

Bild S61.1
Befehle zum Versuch S60.1: Mit dem XOR FF-Befehl kann ein Bitmuster im Akkumulator invertiert werden.

ODER-verknüpft wird. – Der Befehl RST 3Ø hat eine ähnliche Wirkung wie der Befehl RST Ø (vgl. Aufgabe S44.1). Er ersetzt jedoch per Software nicht die Betätigung der Taste RS, sondern setzt einen Breakpoint, der das Programm anhält und das anschließende Auslesen der aktuellen Register-Inhalte erlaubt.

Starten Sie den Befehls-Ablauf bei der Adresse 18ØØ und sehen Sie sich anschließend den Inhalt des Akkumulators an (REG, AF). Es ist (A) = A4, aber das sagt Ihnen zunächst nicht viel.

Sehen Sie sich bitte das Bild S61.2 an, in dem die Exklusiv-ODER-Verknüpfung des Bytes 5B im Akkumulator mit dem Byte FF in Bitmustern dargestellt ist. Zur Verdeutlichung des Vorgangs haben wir die Ø-Bits blau dargestellt und die 1-Bits rot. Vergleichen Sie die Akkumulator-Inhalte vor und nach der Ausführung des Exklusiv-ODER-Befehls! Sie erkennen deutlich, was passiert ist:

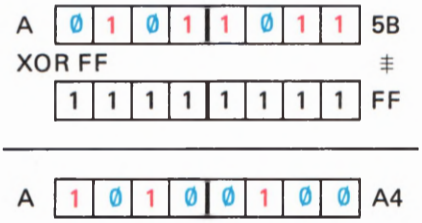


Bild S61.2
Die Betrachtung der Bitmuster zeigt, wie der Befehl XOR FF die Bits im Akkumulator invertiert.

> Die Exklusiv-ODER-Verknüpfung eines Bytes mit dem Byte FF bewirkt, daß jedes einzelne Bit seinen Wert ändert: Aus Ø-Bits werden 1-Bits und aus 1-Bits werden Ø-Bits.

Der XOR FF-Befehl kann in der Software oft den Einsatz von Hardware-NICHT-Gliedern ersetzen. – Überzeugen Sie sich von der Richtigkeit unserer Aussage, indem Sie im Versuch den Operanden des Befehls Nr. 1 bei der Adresse 18Ø1 durch beliebige andere Bytes ersetzen. Schreiben Sie jeweils die Bitmuster an!

Befehle zur logischen UND-Verknüpfung

Wenn man den Mikroprozessor bei der Programmierung als elektronisches Bauelement betrachtet, dann ist die Kenntnis seiner Architektur von ausschlaggebender Wichtigkeit. Der Elektroniker muß sich liebevoll mit jedem einzelnen Bit eines Bytes beschäftigen, denn jedes Bit bedeutet ein digitales Signal, das auf einer speziellen Leitung transportiert wird. Mag der Computer-Programmierer das auch verächtlich als „Bit-Popelei" bezeichnen, dann hat es die Elektronik nun einmal mit einzelnen Signalen auf separaten Leitungen zu tun. Speziell für solche „Bit-Popelei" sind die Befehle für logische Verknüpfungen geeignet.

Für die UND-Verknüpfung von zwei Bytes gilt im Prinzip das gleiche, was wir vorher für die Exklusiv-ODER-Verknüpfung gesagt haben: Sie führt zu einem nichtssagenden Ergebnis, wenn man die Register-Inhalte und das Verknüpfungs-Ergebnis in sedezimaler Schreibweise betrachtet. Einen Sinn ergibt erst die Betrachtung des Bitmusters.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	START	LD	A,3B	1800	3E	3B		A ← 3B
2		LD	B,94	1802	06	94		B ← 94
3		AND	B	1804	A0			A ← (A) ∧ (B)
4		RST	30	1805	F7			Breakpoint

Bild S 62.1
Befehlsfolge zum Versuch S 62.1: Der Inhalt des B-Registers wird mit dem Inhalt des Akkumulators UND-verknüpft.

e_1	e_2	a
0	0	0
0	1	0
1	0	0
1	1	1

Bild S 62.2
Funktionstabelle der UND-Verknüpfung.

Versuch S 62.1

UND-Verknüpfung der Inhalte von Akkumulator und B-Register

Tasten Sie in Ihr System bitte die im Bild S 62.1 aufgelisteten Befehle ein! – Die Anweisung Nr. 3 gehört zur Gruppe der UND-Verknüpfungsbefehle, die wir gleich anschließend zusammenstellen. Der Befehl bewirkt die UND-Verknüpfung des Akkumulator-Inhalts mit dem Inhalt des B-Registers. Wie bei den XOR-Befehlen steht das Verknüpfungsergebnis nach Ausführung des Befehls im Akkumulator und überschreibt das ursprünglich im Akkumulator stehende Byte 3B.

Starten Sie den Programmablauf bei der Adresse 1800 und sehen Sie sich anschließend den Inhalt des Akkumulators an (REG, AF). Es ist

3B ∧ 94 = 10

Ändern Sie versuchsweise den Operanden des LD A,Konst-Befehls in CE und den Operanden des LD B,Konst-Befehls in A3 und wiederholen Sie den Versuch! Sie erhalten:

CE ∧ A3 = 82

Einen Ihnen geläufigen, mathematischen Zusammenhang werden Sie kaum erkennen; auch dann nicht, wenn Sie CE als zweihundertundsechs, A3 als einhundertdreiundsechzig und das Verknüpfungsergebnis als einhundertunddreißig identifizieren. Sinnvoll wird die Sache erst, wenn Sie die ursprünglichen Register-Inhalte als Bitmuster darstellen und die Inhalte dann bitweise nach der Vorschrift der Funktionstabelle im Bild S 62.2 miteinander verknüpfen.

Das Bild S 62.3 zeigt diesen Vorgang. Sie erkennen, daß die Verknüpfung dann und nur dann bei einer bestimmten Bit-Nummer im Akkumulator ein 1-Bit liefert, wenn vorher bei der gleichen Bit-Nummer im Akkumulator UND im B-Register ebenfalls ein 1-Bit steht. In allen anderen Fällen liefert die UND-Verknüpfung 0-Bits.

Die in unserem Lehrgang interessierenden UND-Verknüpfungs-Befehle sehen so aus:

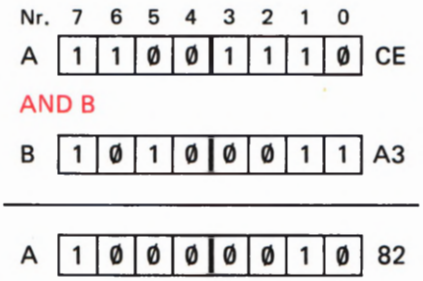


Bild S 62.3
Der Befehl AND B bewirkt die bitweise UND-Verknüpfung des Inhalts des B-Registers mit dem Inhalt des Akkumulators.

Mnemonischer Code	Operation	Erläuterung
AND r AND (HL)	A ← (A) ∧ (r) A ← (A) ∧ (HL)	Der Inhalt eines Registers oder einer vom HL-Registerpaar adressierten Speicherzelle wird mit dem Inhalt des Akkumulators UND-verknüpft.
AND Konst	A ← (A) ∧ Konst	Der Inhalt des Akkumulators wird mit einem konstanten Wert Konst UND-verknüpft.

Die Operationscodes dieser Befehle zeigt die folgende Tabelle:

	A	B	C	D	E	H	L	Speicherzelle	Konstante
AND	A7	AØ	A1	A2	A3	A4	A5	A6	E6

Aufgabe S63.1

Welches Verknüpfungsergebnis bringt die Ausführung des Befehls AND A, also die UND-Verknüpfung des Akkumulator-Inhalts mit sich selbst?

Machen Sie einen Versuch!

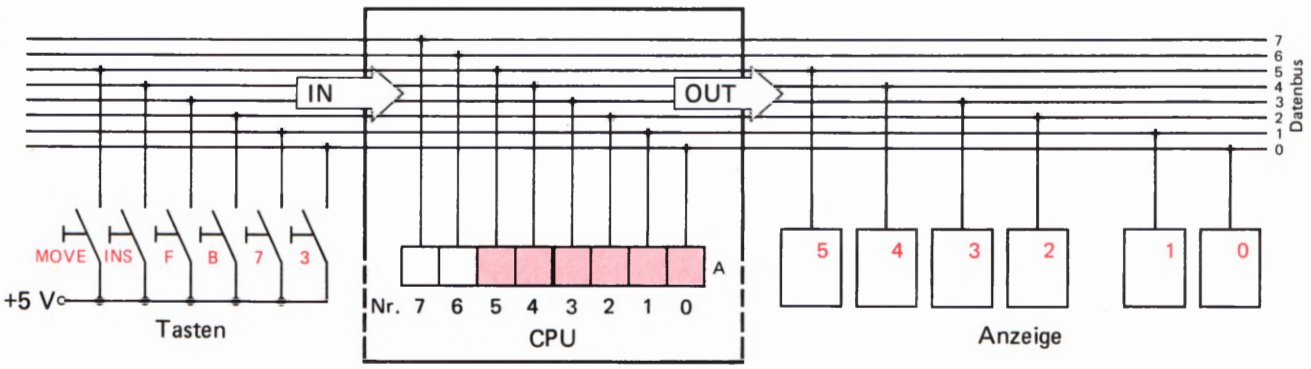
Die Lösung dieser Aufgabe finden Sie auf der Seite Ü10.

Im Gegensatz zur Exklusiv-ODER-Verknüpfung, die sich zum Vergleichen von Bytes anbietet, ist die Brauchbarkeit der UND-Verknüpfungsbefehle nicht so offensichtlich. Tatsächlich werden jedoch beim Programmieren AND-Befehle weit häufiger verwendet als XOR-Befehle.

Was es mit den UND-Verknüpfungsbefehlen auf sich hat, sollen Sie sich in einem Versuch mit dem in den Bildern S64.1 und S64.2 aufgelisteten Programm ansehen. Das Programm bewirkt zunächst, daß sich Ihr System so verhält, als sei es entsprechend dem sehr vereinfachten Schaltplan im Bild S63.1 aufgebaut. Tatsächlich sieht der im Begleitbuch Ihres Systems dargestellte Schaltplan wesentlich anders aus, aber das soll uns hier nicht stören. Wichtig ist die Wirkung des Programms, das den Tasten 3, 7, B, F, INS und MOVE über den Datenbus (vgl. Seite S41) des Systems jeweils ein Bit im Akkumulator zuordnet. (Sie erkennen, daß über den Datenbus des Mikroprozessor-Systems nicht nur Daten zwischen CPU und Speicher transportiert werden, sondern auch zwischen der CPU und sogenannten peripheren Einheiten. Solche peripheren Einheiten sind z. B. ein Tastenfeld oder eine Anzeige, aber auch ein Drucker oder eine gesteuerte Maschine. Jede Peripherie-Einheit hat dann – ähnlich wie eine Speicherzelle – eine eigene Adresse oder deren mehrere.)

Bei der in unserem Versuch simulierten Anordnung bewirkt die Betätigung einer der dargestellten Tasten den Wert 1 eines bestimmten Akkumulator-Bits. Nicht betätigte Tasten bewirken Ø-Bits im Akkumulator. So kann das Bitmuster im Akkumulator bitweise beeinflußt werden.

Bild S63.1
Dieses Bild entspricht nicht der wirklichen Schaltung des Mikro-Professors! Das dargestellte Schema wird im Versuch S64.1 imitiert.



Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	START	LD	A,FE	1800	3E	FE		Tasten Spalte 1
2		OUT	02,A	1802	D3	02		
3		IN	A,00	1804	DB	00		Tasten – Muster holen
4		CPL		1806	2F			invertieren
5		LD	C,A	1807	4F			→ C
6		LD	A,DF	1808	3E	DF		Tasten – Spalte 6
7		OUT	02,A	180A	D3	02		
8		IN	A,00	180C	DB	00		Tasten – Muster holen
9		CPL		180E	2F			invertieren
10		OR	C	180F	B1			Tasten – Muster ein Byte
11		NOP	NOP	1810	00	00		
12		LD	C,A	1812	4F			→ C
13		CALL	DISP	1813	CD	18	18	Tasten – Muster anzeigen
14		JR	START	1816	18	E8		Nächstes Muster

Bild S64.1
Programm zum Versuch S64.1. Dieses Programm ist nur lauffähig, wenn zusätzlich die Bytes für das Unterprogramm DISP entsprechend dem nächstfolgenden Bild eingetastet werden.

Ehe Sie die Fähigkeiten der AND-Befehle untersuchen, wollen wir Ihnen das benutzte Programm vorstellen.

Versuch S64.1

Eingabe von Bits in den Akkumulator über das Tastenfeld

Schalten Sie Ihr System kurzzeitig ab, damit ein vorher gesetzter Breakpoint gelöscht wird. Tasten Sie das Programm aus dem Bild S64.1 und zusätzlich ab der Adresse 1818 die im Bild S64.2 aufgelisteten Bytes in Ihr System ein. Starten Sie das Programm bei der Adresse 1800. In der Anzeige erscheinen lauter Ziffern 0.

DISP

```
1818 06 06 3E 01 CB 09 F5 D4
1820 37 18 DC 3C 18 F1 F6 C0
1828 D3 02 E6 3F CB 07 C5 06
1830 FF 10 FE C1 10 E6 C9 3E
1838 BD D3 01 C9 3E 30 D3 01
1840 C9
```

Bild S64.2
Diese Bytes bilden das im Hauptprogramm (Bild S64.1) aufgerufene Unterprogramm DISP.

Betätigen Sie jetzt nacheinander die Tasten 3, 7, B und F in der rechten Spalte des Tastenfeldes und beobachten Sie dabei die Anzeige! Im Zusammenhang mit der Darstellung im Bild S63.1 erkennen Sie, welchen Anzeige-Stellen und damit welchen Bits im Akkumulator die einzelnen Tasten zugeordnet sind. – Betätigen Sie außerdem die Tasten INS (von links gezählt dritte Taste der oberen Reihe des Tastenfeldes) und MOVE (zweite Taste der oberen Reihe). Diese beiden Tasten sind den beiden linken Anzeige-Stellen und damit den Akkumulator-Bits Nr. 4 und 5 zugeordnet. (Auf eine Tasten-Zuordnung zu den Akkumulator-Bits Nr. 6 und 7 haben wir verzichtet, weil nur sechs Anzeige-Stellen verfügbar sind.) – Betätigen Sie auch mehrere der genannten Tasten gleichzeitig und sehen Sie zu, was passiert!

Wir wollen das Programm im Bild S64.1 kurz erläutern. Im Bild S65.1 erkennen Sie, daß die Tasten in einer Matrix angeordnet sind. Wir interessieren uns hier nur für vier Tasten in der ersten und für zwei Tasten in der sechsten Matrix-Spalte. – Der Befehl Nr. 1 lädt das Byte FE mit dem Bitmuster 1111 1110 in den Akkumulator; der Befehl Nr. 2 bringt sechs Bits dieses Bitmusters mit einem OUT- (Ausgangs-)Befehl in sechs zur Peripherie „Tastenfeld" gehörende Flipflops, die die Spezial-Adresse 02 haben. Das 0-Bit aktiviert die erste Matrix-Spalte.

Die sechs Zeilen der Tasten-Matrix sind über Widerstände auf HIGH-Potential (+5 V) gelegt. Nur die Betätigung einer Taste in der ersten

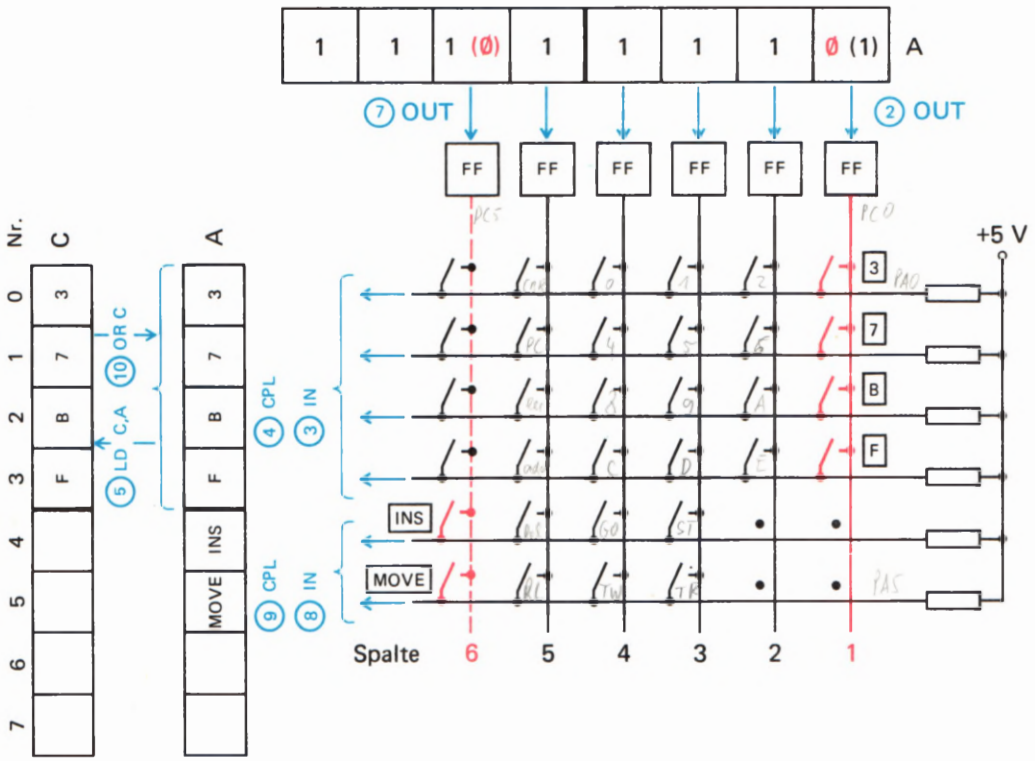


Bild S 65.1
Das Tastenfeld des Mikro-Professors bildet eine Matrix. Die Spalten werden nacheinander aktiviert und die jeweiligen Tasten-Bitmuster in den Akkumulator eingelesen.

Spalte kann die zugehörige Zeile auf LOW-Potential (0 V) ziehen, weil alle anderen Spalten über 1-Signale von den Flipflops auf HIGH-Potential liegen. – Nur betätigte Tasten in der ersten Spalte melden sich also mit Ø-Signalen in der zugehörigen Zeile. Das Zeilen-Bitmuster wird mit dem Eingangs-Befehl IN (Nr. 3) von der Spezial-Adresse ØØ der Peripherie in den Akkumulator geholt und dort mit dem **Ein-Byte-Befehl** Nr. 4, **CPL** (*Com***PL***ement Accumulator*) **invertiert.** (Dieser Befehl kann den Zwei-Byte-Befehl XOR FF – Seite S 61 – ersetzen.) – Der Befehl Nr. 5, LD C,A legt dieses Bitmuster zunächst einmal im C-Register ab.

Die Befehle Nr. 6 und Nr. 7 entsprechen in ihren Wirkungen den Befehlen Nr. 1 und Nr. 2: Sie aktivieren mit dem Ø-Bit Nr. 5 im Akkumulator (Byte 11Ø1 1111 = DF**H**) die sechste Matrix-Spalte. Anschließend wird – wie mit dem Befehl Nr. 3 – das Zeilen-Bitmuster mit dem IN-Befehl Nr. 8 in den Akkumulator geholt und mit dem CPL-Befehl Nr. 9 invertiert. – Den nun folgenden Befehl OR-Befehl werden wir Ihnen im folgenden Abschnitt vorstellen. Hier sei nur festgestellt, daß dieser Befehl die Eingaben mit Tasten der ersten Matrix-Spalte, die im C-Register abgelegt sind, mit den Eingaben über die sechste Matrix-Spalte im Akkumulator kombiniert. Nach Ausführung des Befehls Nr. 10 – und das ist wichtig – steht jetzt im Akkumulator ein Bitmuster, dessen Bits Nr. Ø bis Nr. 5 die im Bild S 65.1 markierten Tasten abbilden: Ein Ø-Bit entspricht einer nicht betätigten Taste; ein 1-Bit entspricht einer betätigten Taste.

Unter der Nr. 11 in der Programm-Liste haben wir zwei Ein-Byte-NOP-Befehle farbig eingetragen. – Der mnemonische Code **NOP** kommt vom englischen **N***o* **OP***eration*, keine Operation. Dieser Befehl wird programmiert, wenn man sich für spätere Verwendung eine Speicherzelle frei halten möchte, oder wenn man programmierte Befehls-Bytes unwirksam machen möchte.

Mnemonischer Code	Operations-code	Operation	Erläuterung
NOP	ØØ	PC ← (PC)+1	Der Inhalt des Programmzählers wird inkrementiert. Keine weitere Wirkung in der CPU

Wir werden im folgenden Versuch an der Stelle der NOP-Befehle einen Zwei-Byte-AND-Befehl einsetzen.

Der Befehl Nr. 12 legt das Eingabe-Bitmuster im C-Register ab, und anschließend wird mit dem Unterprogramm-Aufrufbefehl CALL (vgl. Seite H32) die im Bild S64.2 aufgelistete Anzeige-Routine DISP aufgerufen, mit der das Bitmuster in der Anzeige sichtbar gemacht wird und deren Funktion uns hier nicht zu interessieren braucht. – Der letzte Befehl (JR Start, vgl. Seite H35) sorgt dafür, daß das Programm in einer Schleife läuft, so daß immer neue Tasten-Kombinationen erfaßt und angezeigt werden.

Mit Hilfe des jetzt verfügbaren Programms können Sie sich von der Verwendbarkeit der UND-Verknüpfungs-Befehle recht eindrucksvoll überzeugen.

S

66

Versuch S66.1

Bit-Maskierung mit einem UND-Verknüpfungs-Befehl

Ersetzen Sie die beiden farbig markierten NOP-Befehle in dem im Bild S64.1 aufgelisteten Programm bei den Adressen 181Ø und 1811 durch den AND FD-Befehl:

RS, ADDR, 1, 8, 1, 0, DATA, E, 6, +, F, D

Starten Sie das Programm bei der Adresse 18ØØ und betätigen Sie wie im vorhergehenden Versuch die Tasten 3, 7, B, F, INS und MOVE nacheinander einzeln und in beliebigen Kombinationen! Was hat sich gegenüber dem vorigen Versuch geändert?

Ihr System benimmt sich offenbar so, als sei die Taste 7 defekt. Aber keine Sorge; das ist wieder hinzukriegen.

Halten Sie das Programm an und ändern Sie bei der Adresse 1811 den Operanden des AND Konst-Befehls in F7. Starten Sie das Programm wieder bei der Adresse 18ØØ und wiederholen Sie den Versuch! Jetzt benimmt sich die Taste 7 wieder normal; dafür gelingt es diesmal nicht, über die Taste F ein 1-Bit an die gehörige Stelle in der Anzeige zu bringen.

Was ist da passiert? Das Bild S66.1 erläutert die Zusammenhänge. In den beiden Teilbildern gehen wir davon aus, daß sämtliche sechs im Bild S65.1 markierten Tasten gleichzeitig betätigt werden, daß also nach der Ausführung des Befehls Nr. 10, OR C, die Bits Nr. Ø bis Nr. 5 die Werte 1 haben. Unterschiedlich ist in den beiden Teilbildern die jeweils folgende UND-Verknüpfung. Im Bitmuster des Operanden des AND-Befehls haben im Teilbild a alle Bits bis auf das Bit Nr. 1 die Werte 1; im Teilbild b haben alle Bits bis auf das Bit Nr. 3 die Werte 1.

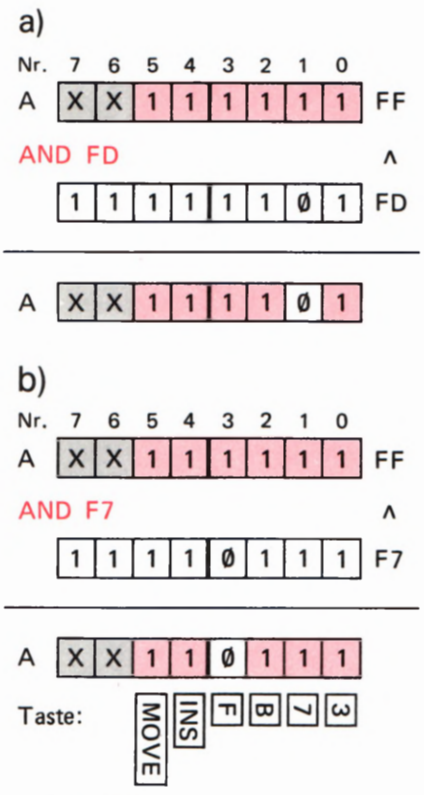


Bild S66.1
Zu Versuch S66.1: Das Byte im Akkumulator wird a) mit dem Byte FD, b) mit dem Byte F7 UND-verknüpft.

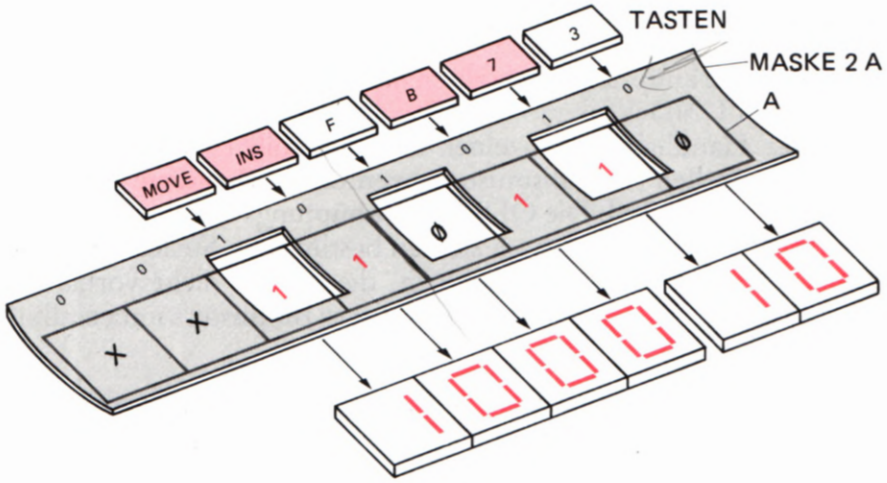


Bild S67.1
Zu Versuch S67.1: Der Operand des UND-Verknüpfungs-Befehls AND 2A wirkt als Maske für den Akkumulator-Inhalt.

Es ist ganz deutlich: Der AND Konst-Befehl bewirkt, daß nach seiner Ausführung nur an den Bit-Stellen im Akkumulator 1-Bits stehen können, an denen im Operanden des AND-Befehls ebenfalls 1-Bits stehen.

Für unseren Versuch bedeutet das, daß bei gleichzeitigem Betätigen der sechs im Programm berücksichtigten Tasten zwar sechs 1-Bits in den Akkumulator gebracht werden; nach der Ausführung des AND-Befehls können aber jeweils die 1-Bits nicht überleben, die an ihrer Bit-Stelle im Operanden des AND-Befehls Ø-Bits vorfinden. Dort kann in der Anzeige also kein 1-Bit ankommen.

Im Bild S67.1 haben wir einen solchen Vorgang etwas anders dargestellt. Wir haben angenommen, daß die Tasten 7, B, INS und MOVE betätigt sind (rote Markierung) und an den Akkumulator das Bitmuster XX11 Ø11Ø geliefert wird. Über den Akkumulator wird nun der Operand des AND-Befehls mit dem Bitmuster ØØ1Ø 1Ø1Ø als **Maske** gelegt. Diese Maske hat an den Stellen Öffnungen, an denen ihr Bitmuster 1-Bits hat. Ø-Bits bedeuten Undurchlässigkeit der Maske. Sie erkennen, daß die Maske nur die 1-Bits aus dem Akkumulator zur Anzeige kommen läßt, die unter einer Öffnung der Maske liegen. Im dargestellten Fall sind das die 1-Bits an den Bitstellen Nr. 1 und Nr. 5.

Versuch S67.1

Maskierung mit dem Byte 2A

Ändern Sie den Operanden des AND Konst-Befehls bei der Adresse 1811, den Sie im vorhergehenden in das Programm im Bild S64.1 gebracht haben, in den Wert 2A! – Starten Sie das Programm, betätigen Sie die Tasten 7, B, INS und MOVE gleichzeitig und vergleichen Sie die Anzeige mit der Darstellung im Bild S67.1!

> Maskieren ist ein Verfahren zur gezielten Auswahl bestimmter Binärstellen in einem Bitmuster.

Wir haben die Maskierung mit einem AND Konst-Befehl vorgenommen. Selbstverständlich kann man auch einen der anderen AND-Befehle verwenden, wenn die Maske in einem CPU-Register oder in einer vom HL-Registerpaar adressierten Speicherzelle abgelegt ist.

S
68

Befehle zur logischen ODER-Verknüpfung

Die ODER-Verknüpfungs-Befehle stellen gewissermaßen das Gegenstück zu den UND-Verknüpfungs-Befehlen dar. Auch diese Befehle dienen der Handhabung einzelner Bits. Sie sollen jedoch nicht an bestimmten Stellen eines Bitmusters eventuell vorhandene 1-Bits eliminieren. Im Gegenteil: Die ODER-Verknüpfungs-Befehle können in eleganter Weise dazu dienen, gezielt an bestimmten Stellen eines Bitmusters zusätzliche 1-Bits einzufügen, die vorher nicht vorhanden waren. Dabei können alle übrigen Bits dieses Bitmusters unbeeinflußt bleiben.

Ehe wir Ihnen die einzelnen ODER-Verknüpfungs-Befehle vorstellen, sollen Sie sich in einem Versuch von der Wirkungsweise eines solchen Befehls überzeugen.

Versuch S68.1

Das Hinein-ODERn eines 1-Bits

Dem Versuch liegt das Programm aus den Bildern S64.1 und S64.2 zugrunde. Tasten Sie anstelle der beiden dort farbig eingetragenen NOP-Befehle die Bytes F6 und 1Ø ein:

RS, ADDR, 1, 8, 1, 0, DATA, F, 6, +, 1, 0

Starten Sie das Programm bei der Adresse 18ØØ und betätigen Sie die im Bild S65.1 markierten Tasten einzeln und in beliebigen Kombinationen. Was fällt Ihnen auf?

Halten Sie das Programm mit der Taste RS an und ändern Sie das Byte bei der Adresse 1811 in das Byte Ø2! Betätigen Sie die Tasten und beobachten Sie die Anzeige!

In beiden Fällen steht – unabhängig davon, ob eine der vom Programm berücksichtigten Tasten betätigt war oder nicht – an einer bestimmten Stelle des Bitmusters ein 1-Bit. Dieses 1-Bit haben wir unabhängig von der eigentlich zugehörigen Taste per Programm mit einem ODER-Befehl in das Bitmuster hineingebracht.

Sehen Sie sich bitte noch einmal das Programm im Bild S64.1 an! Nach der Anweisung Nr.10, OR C, steht das jeweils aktuelle, von den Tasten gelieferte Bitmuster im Akkumulator (vgl. Seite S65).

Anstelle der beiden anschließenden NOP-Befehle haben wir jetzt einen der gleich noch vorzustellenden ODER-Verknüpfungsbefehle eingesetzt, der das Bitmuster im Akkumulator so beeinflußt, wie es die Bilder S68.1a und b erläutern. Im Teilbild a haben wir angenommen, daß die Taste B betätigt wurde und daß dementsprechend im Akkumulator das Byte XXØØ Ø1ØØ erscheint. Der jetzt eingefügte ODER-Verknüpfungsbefehl bewirkt eine bitweise ODER-Verknüpfung dieses Bytes im Akkumulator mit dem Operanden 1Ø des Befehls nach der Vorschrift der ODER-Verknüpfungstabelle im Bild S69.1. (Vgl. die Aufgabe S29.1 und deren Lösung auf der Seite Ü4.)

Das Teilbild b erläutert die gleiche Verknüpfung für den Fall, daß der Operand des ODER-Verknüpfungs-Befehls den Wert Ø2 hat und die Taste F betätigt wurde. – In beiden Fällen wird in unserem Programm

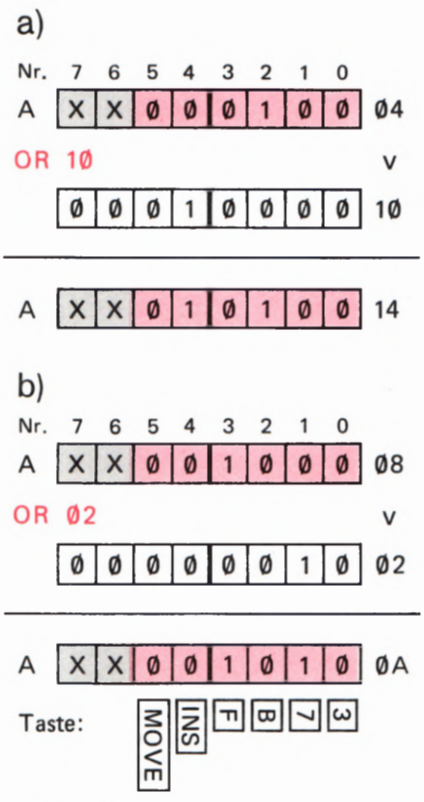


Bild S68.1
Zu Versuch S68.1: Das Byte im Akkumulator wird a) mit dem Byte 1Ø, b) mit dem Byte Ø2 ODER-verknüpft.

der DISP-Routine ein Byte übergeben, das außer dem von einer Taste bewirkten 1-Bit ein zusätzlich hineingeODERtes Bit enthält, das immer mit der Ziffer 1 in der Anzeige gemeldet wird, unabhängig von den im Programm abgefragten Tasten.

Für die ODER-Verknüpfung stehen im Befehlssatz des Mikroprozessors die gleichen Möglichkeiten zur Verfügung wie für die Exklusiv-ODER- und für die UND-Verknüpfung. – Die hier interessierenden ODER-Verknüpfungs-Befehle sehen so aus:

Mnemonischer Code	Operation	Erläuterung
OR r OR (HL)	A ← (A) ∨ (r) A ← (A) ∨ (HL)	Der Inhalt eines Registers oder einer vom HL-Registerpaar adressierten Speicherzelle wird mit dem Akkumulator-Inhalt ODER-verknüpft.
OR Konst	A ← (A) ∨ Konst	Der Inhalt des Akkumulators wird mit einem konstanten Wert Konst ODER-verknüpft.

Die Operationscodes dieser Befehle zeigt die folgende Tabelle:

	A	B	C	D	E	H	L	Speicherzelle	Konstante
OR	B7	BØ	B1	B2	B3	B4	B5	B6	F6

Nach dieser Vorstellung verstehen Sie nun auch, wie der Befehl Nr. 10, OR C, in dem im Bild S64.1 aufgelisteten Programm arbeitet: Er „ODERt" die mit dem Befehl Nr. 5, LD C,A, zwischenzeitlich im C-Register abgelegten 1-Bits von betätigten Tasten in der ersten Matrix-Spalte in den Akkumulator zurück.

Im übrigen gilt für die ODER-Befehle eine entsprechende Feststellung, wie wir sie auf der Seite S67 für die UND-Befehle getroffen haben: Das „HineinODERn" von 1-Bits in ein Bitmuster kann statt mit einem OR Konst-Befehl auch mit einem anderen ODER-Befehl vorgenommen werden, wenn diese 1-Bits in einem CPU-Register oder in einer vom HL-Registerpaar adressierten Speicherzelle abgelegt sind. – Sehen Sie sich diese Möglichkeit in einem Versuch an.

e_1	e_2	a
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

Bild S69.1
Funktionstabelle zur ODER-Verknüpfung von zwei Variablen.

Versuch S69.1

Das „hineinzuODERnde" Bit steht in einer Speicherzelle

Benutzen Sie für diesen Versuch bitte wieder das in den beiden Bildern S64.1 und S64.2 aufgelistete Programm und ersetzen Sie die farbig ein-

Bild S69.2
Zu Versuch S69.1: Mit dem Befehl Nr. 11 wird der Ansprung eines „Rucksacks" programmiert. Der Rucksack hat den Label RUCKS und schließt mit einem Rücksprung in das Hauptprogramm ab.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
11		JR	RUCKS	181Ø	18	2F		Rucksack anspringen
	RUCKS	LD	(HL),19ØØ	1841	21	ØØ	19	Zeiger auf 19ØØ
		OR	(HL)	1844	B6			Bits einODERn
		JR	CBH	1845	18	CB		Rücksprung

getragenen Bytes ØØ für die NOP-Befehle ab der Adresse 181Ø durch die beiden Bytes 18 und 2F (Nr. 11 im Bild S 69.2). Tasten Sie außerdem bitte ab der Adresse 1841 die ebenfalls im Bild S 69.2 aufgelisteten Befehls-Bytes ein. – Legen Sie zum Schluß bei der Adresse 19ØØ das Byte Ø2 ab und starten Sie das Programm bei der Adresse 18ØØ. Betätigen Sie die im Bild S 65.1 markierten Tasten und beobachten Sie die Anzeige!

Ihr System verhält sich so, wie es im Bild S 68.1b dargestellt ist: Im Akkumulator steht – unabhängig davon, ob die Taste 7 betätigt ist oder nicht – an der Bit-Stelle Nr. 1 ein 1-Bit, und dieses Bit wird mit der Ziffer 1 an der (von rechts gezählt) zweiten Anzeigestelle gemeldet.

Das 1-Bit Nr. 1 wird im Prinzip genau auf die gleiche Weise in das Akkumulator-Bitmuster hineingeODERt, wie es das Bild S 68.1b zeigt. Allerdings wird das jetzt nicht mit einem OR Konst-Befehl erreicht, sondern mit dem im Bild S 69.2 farbig markierten Befehl OR (HL). –

Damit das funktioniert, mußten wir einen üblen Programm-Trick anwenden, der beim „sauberen" Programmieren höchst unbeliebt ist. Das Bild S 70.1 zeigt, wie wir das gemacht haben:

Um das in der Speicherzelle mit der Adresse 19ØØ abgelegte Byte in das Akkumulator-Bitmuster ODERn zu können, muß der Inhalt des HL-Registerpaares auf diese Speicherzelle zeigen. Vor dem OR (HL)-Befehl muß dazu der Drei-Byte-Befehl LD HL,19ØØ (Seite H 36) programmiert werden. Zusammen mit dem Ein-Byte-OR (HL)-Befehl brauchen wir also für das HineinODERn vier Speicherzellen. Im Hauptprogramm unseres Versuchs (Bild S 64.1) haben wir jedoch nur zwei Adressen zum Einfügen von logischen Verknüpfungsbefehlen freigelassen.

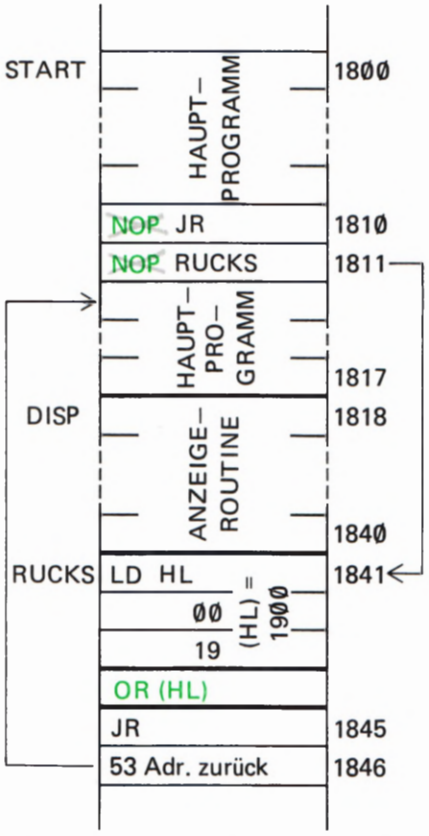


Bild S 70.1
Darstellung des Speicher-Inhalts bei einem angehängten Programm-„Rucksack".

Wir helfen uns, indem wir anstelle der beiden NOP-Befehle im Hauptprogramm einen Sprungbefehl zu einem „Rucksack" am Ende des Programms einsetzen. Dort können ab dem Label RUCKS nun beliebig viele Befehle programmiert werden. Wir haben beim Label RUCKS die beiden Befehle LD HL,19ØØ und OR (HL) angeschrieben und nach diesen beiden Befehlen einen Sprungbefehl programmiert, der die Fortsetzung des Programms nach den beiden ursprünglichen NOP-Befehlen verursacht. – Die Darstellung im Bild S 70.1 macht diesen Vorgang deutlich.

Programmtechnisch sauberer wäre es natürlich gewesen, den auf die beiden NOP-Befehle folgenden Teil des Hauptprogramms und das anschließende Unterprogramm so weit zu verschieben, daß Platz für den zusätzlichen LD HL,19ØØ-Befehl entstanden wäre. Das hätte jedoch bedeutet, daß diese Programmteile neu eingetastet werden müssen. Außerdem hätte sich die Anfangsadresse des im Hauptprogramm aufgerufenen Unterprogramms DISP geändert. Bei kurzen Programmen lohnt sich ein solcher Aufwand; bei längeren Programmen greift man dann lieber zu dem hier beschriebenen Trick mit dem „Rucksack".

Wir werden Ihnen noch zeigen, welche Möglichkeiten der Mikro-Professor bietet, ganze Programmteile ohne Neu-Eingabe problemlos zu verschieben.

Der Befehlssatz unseres Mikroprozessors

(Fortsetzung)

Wir haben bereits früher festgestellt, daß die von einem Mikroprozessor ausführbaren Sprungbefehle die sogenannte Intelligenz dieses Bauelements ausmachen und ihm zu seiner Bedeutung verholfen haben. Im einfachsten Fall handelt es sich dabei um den Befehl für einen unbedingten Sprung, also um einen Befehl, der unabhängig von irgendwelchen Bedingungen auf jeden Fall ausgeführt wird, und der nichts anderes tut, als die Fortsetzung eines Programms bei einem anderen als dem bei der nächstfolgenden Adresse stehenden Befehl zu veranlassen. Dieser Befehl und die Art seiner Verarbeitung soll uns hier beschäftigen. Mit „Intelligenz" hat dieser Befehl jedoch noch nichts zu tun.

Richtig interessant wird die Sache erst bei den Befehlen für bedingte Sprünge. Wie die Bezeichnung andeutet, bewirken diese Befehle die Fortsetzung eines Programms bei einer vorherbestimmbaren Adresse nur unter bestimmten Bedingungen. Wenn diese Bedingungen nicht erfüllt sind, dann wird der Sprung nicht ausgeführt. Die Befehle für bedingte Sprünge erlauben also die Programmierung von Verzweigungen und Schleifen (vgl. ab Seite S3).

Der Programmzähler als Sprung-Dirigent

Wir haben bereits auf der Seite H44 darauf hingewiesen, daß das sechzehn Bit breite Register Programmzähler ein Pointer-Register ist. Sein Inhalt ist eine sedezimal vierstellig dargestellte Zahl, die der Mikroprozessor immer als Adresse einer Speicherzelle interpretiert. Darüber hinaus betrachtet der Mikroprozessor den Inhalt einer vom Programmzähler adressierten Speicherzelle immer als Operationscode oder als Operand eines Befehls.

Lesen Sie bitte in diesem Zusammenhang noch einmal ab der Seite S44 nach, welche Funktionen das HL-Registerpaar und die BC- und DE-Registerpaare haben, damit Ihnen die Sonderstellung des Programmzählers ganz klar wird.

Die Bezeichnung Programmzähler für dieses CPU-Register ist nicht sehr geschickt. Es handelt sich um eine wörtliche Übersetzung der englischen Bezeichnung **P***rogram* **C***ounter* (sprich: prográm kaunter), aus der die häufig benutzte Abkürzung PC für dieses Register gebildet wird. Das Register ist nicht eigentlich ein Zähler und noch weniger werden Programme gezählt. Die oft benutzte Bezeichnung **Befehlszähler** kommt der Sache schon näher.

Ehe wir uns den eigentlich interessierenden Sprungbefehlen zuwenden, wollen wir kurz die Arbeitsweise des Programmzählers ansehen, denn dieses Register ist letztlich für die Ausführung von Sprungbefehlen zuständig.

Als Pointer-Register „zeigt" der Programmzähler immer mit seinem Inhalt auf eine Speicherzelle. Vor dem Abarbeiten eines Befehls ist der Inhalt dieser Speicherzelle in jedem Fall der Operationscode eines Befehls (Bild S72.1). (Natürlich kann man den Programmzähler auch

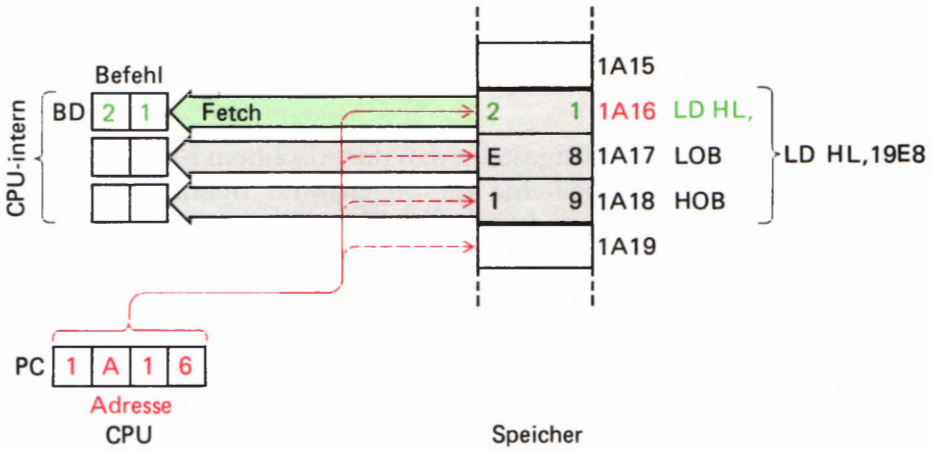


Bild S72.1
Vor der Ausführung eines Befehls holt sich die CPU den Operationscode und die Operanden in interne Register (Fetch-Zyklen). Nach jedem Einholen wird der Programmzähler PC inkrementiert.

zwangsweise auf eine Speicherzelle zeigen lassen, die ein per Programm zu verarbeitendes Bitmuster oder eine zu verarbeitende Zahl enthält. Der Mikroprozessor interpretiert den Inhalt dieser Speicherzelle dann als Operationscode und reagiert ganz unsinnig.)

Zum Abarbeiten des Befehls holt der Mikroprozessor den Inhalt der vom Programmzähler adressierten Speicherzelle in seinen Befehlsdecoder (Seiten S44 und S45) und inkrementiert gleich darauf automatisch den Inhalt des Programmzählers. In unserer Darstellung im Bild S72.1 wird der Inhalt des Programmzählers also sofort nach dem Einholen (englisch: *Fetch*, sprich: fetsch) des Operationscodes 21 für den Befehl LD HL,Konst vom Wert 1A16 auf den Wert 1A17 inkrementiert.

Jetzt schaut sich der Mikroprozessor den Inhalt des Befehlsdecoders daraufhin an, ob es sich um den Operationscode eines Ein-Byte- oder eines Mehr-Byte-Befehls handelt. Hat er den Operationscode eines Ein-Byte-Befehls geholt, dann kann er sich sofort an die Ausführung dieses Befehls machen, denn dann hat er bereits alle codierten Informationen, die er zur Ausführung dieses Befehls braucht. Handelt es sich jedoch um einen Mehr-Byte-Befehl, z. B., wie in unserer Darstellung um einen Drei-Byte-Befehl, dann sorgt sein internes **Mikroprogramm** dafür, daß vor der Ausführung des Befehls auch der Inhalt der bereits vom Programmzähler adressierten, folgenden Speicherzelle (im Bild der Inhalt der Speicherzelle mit der Adresse 1A17) in die CPU geholt wird. Dieser Inhalt (im Bild: E8) wird in einem CPU-internen, dem Programmierer nicht zugänglichen Register abgelegt. Sofort darauf wird der Inhalt des Programmzählers wiederum (im Bild von 1A17 auf 1A18) inkrementiert. – Wenn es sich um einen Drei-Byte-Befehl handelt, dann folgt wieder ein Einhol-Zyklus; der Inhalt der bereits adressierten Speicherzelle wird in ein weiteres, CPU-internes Register geladen und sogleich anschließend erfolgt noch einmal eine Inkrementierung des Programmzählers.

Die CPU hat jetzt den kompletten Befehl gespeichert und kann ihn ausführen. Das Bild S73.1 zeigt in stark vereinfachter Form den Ablaufplan des Mikroprogramms für die Ausführung eines Befehls.

Uns interessiert in diesem Zusammenhang vor allem das Verhalten des Programmzählers. Wir haben deshalb im Ablaufplan des Mikroprogramms die den Programmzähler betreffenden Anweisungen rot unterlegt. Sie erkennen, daß der Programmzähler nach jedem Einhol-

Zyklus inkrementiert wird. Sein Inhalt zeigt also – und diese Feststellung ist besonders wichtig! – bereits vor der Ausführung des gerade aktuellen Befehls auf die Speicherzelle, die den Operationscode des nächsten, auszuführenden Befehls enthält.

Das hier erläuterte, automatische Inkrementieren des Programmzähler-Inhalts ist der Grund dafür, daß ein im Speicher abgelegtes Programm vom Mikroprozessor Schritt für Schritt abgearbeitet wird. Das Mikroprogramm sorgt dafür, daß nach der Ausführung eines aus dem Speicher geholten Befehls ganz selbstverständlich der in der nächstfolgenden Speicherzelle abgelegte Operationscode in die CPU geholt und als Befehl ausgeführt wird.

Sehen Sie sich bitte noch einmal das im Bild S64.1 aufgelistete Programm zum Versuch S64.1 und die zugehörige Erläuterung an! – Wir wollen annehmen, Sie hätten nach dem Einschalten Ihres Systems die zu diesem Programm gehörenden Befehle eingetastet, die Eingabe jedoch nach dem Eintasten des Befehls Nr. 13 mit dem Byte 18 bei der Adresse 1815 abgebrochen. Das Programm enthält in diesem Fall nicht mehr den Befehl Nr. 14, JR START (vgl. Seite S66).

Was geschieht, wenn Sie nach diesen Eingaben das Programm bei der Adresse 18ØØ starten? – Sie brauchen den Versuch nicht erst zu machen; wir können voraussagen, was passiert: Nach dem Start des Programms wird der Mikroprozessor die eingegebenen Befehle ausführen und das vorgefundene Tasten-Muster anzeigen. Nach der Ausführung des Unterprogramms DISP, das mit dem Befehl Nr. 13 (CALL DISP) aufgerufen wird, zeigt der Programmzähler auf die Speicherzelle mit der Adresse 1816 und holt das dort vorgefundene Byte in seinen Befehlsdecoder Bd. Er betrachtet dieses Byte als Operationscode des nächsten, auszuführenden Befehls.

Welchen Befehl wird er vorfinden? Das ist in keiner Weise vorherzusagen, denn Sie haben bei dieser Adresse selbst keinen Befehl mehr eingetastet. Es wird irgendein x-beliebiges, zufälliges Byte sein, das sich im Speicher nach dem Einschalten des Systems eingestellt hat. Trotzdem wird der Mikroprozessor dieses Byte sehr ernst nehmen und den entsprechenden Befehl (wenn es überhaupt einer ist!) gewissenhaft ausführen. Danach holt es sich das nächste, zufällig im Speicher stehende Byte als Operationscode und führt auch die darin enthaltene Anweisung getreulich aus usw. Das Programm „läuft in die Wüste".

Das müssen wir verhindern. Aber wie? Am einfachsten scheint es natürlich zu sein, dem Befehl Nr. 13 zum Aufruf des Unterprogramms DISP weitere, sinnvolle Befehle folgen zu lassen. Aber irgendwann hat auch das ein Ende und wir stehen wieder vor dem gleichen Problem.

Die Lösung ist ganz einfach: Wir müssen dafür sorgen, daß der Programmzähler nach der Ausführung des Unterprogramms DISP nicht auf die nächstfolgende Speicherzelle zeigt, in der ein Byte für einen in diesem Zusammenhang sinnlosen Befehl steht, sondern daß er so geladen wird, daß ab dem Label START das nächste Bitmuster abgefragt wird. Dazu muß der Programmzähler nach der Ausführung des Befehls Nr. 13, also nach dem Abarbeiten des Unterprogramms DISP, mit der Adresse 18ØØ geladen werden.

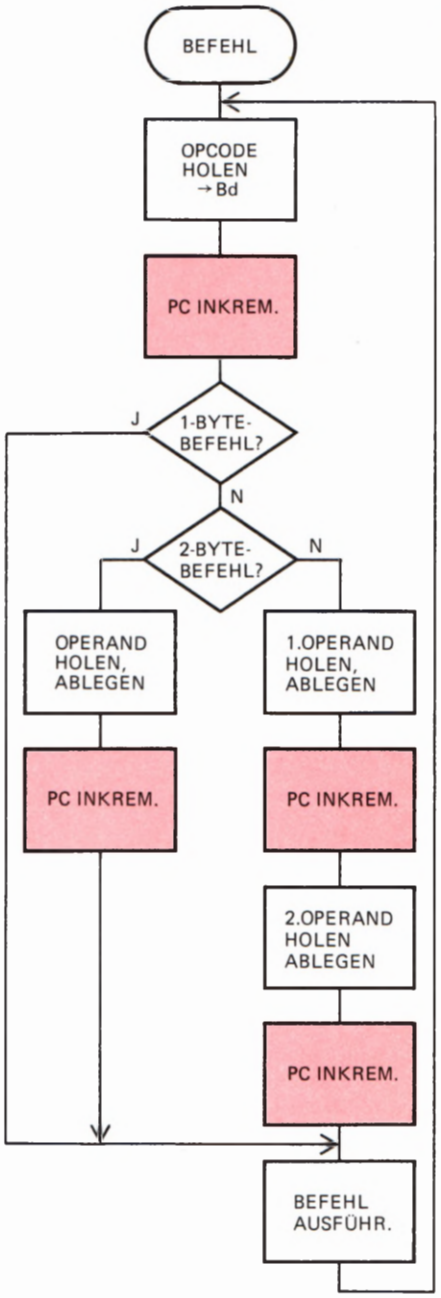


Bild S73.1
Stark vereinfachter Ablaufplan des Mikroprogramms zur Ausführung eines Befehls.

Der Befehlssatz unseres Mikroprozessors enthält einen Befehl, mit dem der Inhalt des Programmzählers auf einen beliebigen Wert gesetzt werden kann. Dieser Wert muß vorher im HL-Registerpaar stehen. Es handelt sich also um einen Kopier-Befehl für den Inhalt des HL-Registerpaares in den Programmzähler:

Mnemonischer Code	Operations-code	Operation	Erläuterung
JP (HL)	E9	PC ← (HL)	Der Inhalt des HL-Registerpaares wird in den Programmzähler kopiert.

Dieser Befehl ist ein Ein-Byte-Befehl, dessen mnemonischer Code vom englischen Wort **J***um***P** (sprich: dschamp) kommt, das „springe" bedeutet. Gemeint ist mit dem mnemonischen Code, daß der Programmzähler mit dem Inhalt des HL-Registerpaares geladen wird und daß das Programm zu der Adresse springt, die im HL-Registerpaar abgelegt ist.

Versuch S74.1

Umladen des Programmzählers

Tasten Sie für diesen Versuch das Programm für den Versuch S64.1 (Bilder S64.1 und S64.2) ein! (Wenn Sie dieses Programm auf Cassette geladen haben, dann haben Sie es jetzt einfacher: Sie brauchen es nur mit dem Recorder wieder in den Speicher einzuspielen).

Im Versuch S64.1 haben wir mit dem Befehl Nr. 14, JR Start, den Ablauf des Programms in einer Schleife erreicht (vgl. Seite S66). Diesen Befehl werden wir Ihnen nachher vorstellen. Zunächst wollen wir von diesem Befehl keinen Gebrauch machen und die Wiederholung des Programm-Ablaufs ab dem Label START nach dem Abarbeiten des Unterprogramms DISP, also nach der Ausführung des Befehls CALL DISP, durch Laden des Programmzählers mit der Adresse 18ØØ erreichen.

Es muß dazu zunächst das HL-Registerpaar mit der Adresse 18ØØ (Label START) geladen (Drei-Byte-Befehl) und anschließend der Befehl JP (HL) (Ein-Byte-Befehl) ausgeführt werden. Für die Fortsetzung des Programms beim Label START nach der Ausführung des Unterprogramms DISP werden also vier Bytes benötigt. Zwei dieser Bytes stehen durch die in diesem Programm nicht benötigten NOP-Befehle bei den Adressen 181Ø und 1811 zur Verfügung; zwei weitere Speicherzellen werden dadurch frei, daß wir den Zwei-Byte-Befehl JP START ab der Adresse 1816 hier nicht verwenden wollen (Bild S64.1). Das Unterprogramm DISP braucht also nicht – wie bei der Lösung der Aufgabe H53.1 – verschoben zu werden.

Bild S74.1
Zu Versuch S74.1: Die vier Bytes der Befehle Nr. 14 und Nr. 15 laden den Programmzähler so um, daß das Programm bei der Adresse 18ØØ fortgesetzt wird.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
12		LD	C,A	181Ø	4F			Tasten-Muster → C
13		CALL	DISP	1811	CD	18	18	Tasten-Muster anzeigen
14		LD	HL,18ØØ	1814	21	ØØ	18	HL laden
15		JP	(HL)	1817	E9			PC mit (HL) laden

Ändern Sie ab der Adresse 181Ø das Programm aus dem Bild S 64.1 entsprechend den im Bild S74.1 aufgelisteten Befehlen! Im Vergleich mit dem Bild S64.1 erkennen Sie, daß der Befehl LD C,A jetzt um zwei Speicherzellen weiter nach unten eingetragen ist. Die beiden neuen Befehle LD HL,18ØØ und JP (HL) besorgen den Sprung des Programms zum Label Start.

Sie sollen sich ansehen, wie der Mechanismus des geänderten Programmzähler-Inhalts funktioniert. Setzen Sie dazu bei der Adresse 1814, also nach dem Ablauf des Unterprogramms DISP, einen Breakpoint (vgl. Seite S26):

ADDR, 1, 8, 1, 4, SBR

und starten Sie dann das Programm bei der Adresse 18ØØ. Ihr System führt den Befehl LD HL,18ØØ bei der Adresse 1814 noch aus und unterbricht dann den Programm-Ablauf. Wie bereits erläutert, wird der Programmzähler-Inhalt nach dem Einholen des letzten Bytes, das zum Befehl LD HL,18ØØ gehört, noch einmal erhöht. Nach dem Anhalten des Programms wird der jetzt aktuelle Programmzähler-Inhalt 1817 angezeigt. Der Inhalt der Adresse 1817, auf die der Programmzähler zeigt, wird ebenfalls angezeigt: Es ist der Operationscode des nicht mehr ausgeführten Befehls JP (HL).

Sehen Sie sich den Inhalt des HL-Registerpaares an:

REG, HL Anzeige: 1 8 0 0 H L

Vergessen Sie nun nicht, mit der Taste PC den aktuellen Inhalt des Programmzählers wieder in die Anzeige zu holen. – Jetzt können Sie in einem Einzelschritt den Befehl JP (HL) ausführen lassen:

STEP Anzeige: 1 8 0 0 3. E.

Es wird Ihnen wieder der aktuelle Inhalt des Programmzählers angezeigt: Es ist der Inhalt, den der gerade ausgeführte Befehl JP (HL) aus dem HL-Registerpaar in den Programmzähler kopiert hat. Der Programmzähler zeigt also erwartungsgemäß auf die Speicherzelle, in der der Operationscode 3E des ersten Befehls (LD A,FE) unseres Programms (Bild S64.1) abgelegt ist.

Sie können den vorhin gesetzten Breakpoint jetzt wieder löschen, so daß unser Programm nach neuerlichem Start nicht wieder bei der Adresse 1817 anhält, indem Sie die Taste CBR betätigen. CBR ist die Abkürzung des englischen **C***lear* **BR***eakpoint* (bereinige den Breakpoint). Sie finden diese Taste als (von links gezählt) vierte in der zweiten Reihe des Tastenfeldes.

Starten Sie das Programm bei der Adresse 18ØØ: Es verhält sich so, wie es im Versuch S64.1 beschrieben worden ist.

Der Befehl für einen unbedingten Sprung

Die Sprung-Anweisung, die wir im Versuch S64.1 verwendet haben, hat deutlich gemacht, daß ein Sprung einfach durch gezieltes Verändern des Programmzähler-Inhalts programmiert werden kann. Die von uns gewählte Sprung-Programmierung ist jedoch recht umständlich. Sie bedarf zweier Befehle und ist dadurch vier Byte lang. Außer-

dem muß das HL-Registerpaar benutzt werden, das vielleicht gerade mit wichtigen und unverzichtbaren Bytes geladen ist.

Der JP (HL)-Befehl ist dann von Interesse, wenn in Abhängigkeit des Inhalts des HL-Registerpaares unterschiedliche Programmteile angesprungen werden sollen.

Der Befehl JP (HL) erlaubt nur ein indirektes Laden des Programmzählers über das HL-Registerpaar, das seinerseits mit einem Befehl LD HL,Konst direkt geladen werden kann. Viel eleganter für Sprunganweisungen wäre es doch, wenn es einen direkten Ladebefehl für den sechzehn Bit „breiten" Programmzähler gäbe, der den direkten Ladebefehlen für die ebenfalls sechzehn Bit „breiten" Registerpaare HL,BC und DE (Seite H36) entspräche. Ein solcher Befehl wäre ein Drei-Byte-Befehl mit zwei Operanden und würde bewirken, daß die CPU ihren nächsten Befehl bei der Speicherzelle abholt, deren Adresse in Form der Operanden des Ladebefehls für den Programmzähler angegeben ist.

Ein solcher direkter Ladebefehl für den Programmzähler bewirkt die Fortsetzung eines Programms bei einer willkürlich vorgebbaren Adresse. Im Befehlssatz unseres Mikroprozessors hat er den bereits mehrfach erwähnten mnemonischen Code JP:

Mnemonischer Code	Operations-code	Operation	Erläuterung
JP adr	C3	PC ← adr	Direktes Laden des Programmzählers. Das Programm wird bei der Adresse adr fortgesetzt.

Der Operand adr enthält eine sedezimal vierstellig dargestellte Adresse. Es handelt sich also um einen Drei-Byte-Befehl. Dabei ist als sehr wichtige Tatsache zu beachten:

Das **LOB** der Adresse wird im Speicher bei der **niederen Adresse** abgelegt.

Das **Hob** der Adresse wird im Speicher bei der **höheren Adresse** abgelegt.

Wir haben auf diese Tatsache bereits bei den Befehlen für den direkten Datenverkehr zwischen CPU und Speicher auf der Seite S43 hingewiesen.

Versuch S76.1

Sprung mit absoluter Adressierung

Auch für diesen Versuch verwenden wir wieder das Programm zum Versuch S64.1 (Bilder S64.1 und S64.2). Wie im Versuch S74.1 benutzen wir jedoch die Speicherzellen mit den Adressen 181Ø und 1811, in denen NOP-Befehle programmiert waren, da wir zunächst für die verwendeten Befehle etwas mehr Platz gebrauchen als im ursprünglichen Programm.

Ändern Sie das Programm ab der Adresse 181Ø so, wie es im Bild S77.1 aufgelistet ist! – Am Unterprogramm DISP brauchen keine Änderungen vorgenommen zu werden.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
12		LD	C,A	181Ø	4F			Tasten-Muster → C
13		CALL	DISP	1811	CD	18	18	Tasten-Muster anzeigen
14		JP	START	1814	C3	ØØ	18	Sprung nach START
15		NOP		1817	ØØ			Nicht benutzt

Bild S77.1
Zu Versuch S76.1: In dieser Befehlsfolge bewirken die drei Bytes des Befehls Nr.14 die Fortsetzung des Programms bei der Adresse 18ØØ.

Sie können sich wieder ansehen, wie der jetzt programmierte Sprung mit absoluter Adressierung arbeitet. Dazu wird ein Breakpoint so in das Unterprogramm DISP gesetzt, daß es gerade ganz abgearbeitet wird, daß das Hauptprogramm jedoch anhält, ehe der Sprung zum Label START stattfindet:

ADDR, 1, 8, 3, 6, SBR

Starten Sie das Programm bei der Adresse 18ØØ! Das Unterprogramm wird abgearbeitet, und anschließend meldet die Anzeige den Programmzähler-Inhalt 1814 und den bei dieser Adresse stehenden Operationscode C3 des Sprungbefehls JP (Bilder S77.1 und Seite S76). Mit der Taste STEP können Sie diesen Befehl in einem Einzelschritt ausführen lassen. Ihr System meldet wie erwartet als aktuellen Programmzähler-Inhalt 18ØØ und zeigt damit an, daß der weitere Ablauf des Programms beim Label START beginnt.

Löschen Sie den Breakpoint (Taste CBR, vgl. Seite S75). Mit der Taste PC können Sie den aktuellen Programmzähler-Inhalt 18ØØ in die Anzeige zurückholen und den kontinuierlichen Ablauf des Programms anschließend mit der Taste GO starten. Das Programm verhält sich wieder so, wie es im Versuch S64.1 beschrieben ist.

Den jetzt vorgestellten Sprung-Befehl haben wir als Befehl mit absoluter Adressierung bezeichnet, weil die Operanden dieses Befehls in eindeutiger Form die Adresse des Befehls enthalten, der als nächster ausgeführt werden soll. Mit anderen Worten: Die Operanden des JP-Befehls enthalten die Bytes, mit denen das Register Programmzähler geladen wird.

Wenn ein Programm, das einen oder mehrere JP-Befehle enthält, z. B. mit einem MOVE-Kommando Ihres Systems (vgl. ab Seite H55) im Speicher verschoben wird, dann bewirken die JP-Befehle vor und nach der Verschiebung das gleiche: Nach Ausführung eines dieser Befehle wird das Programm bei der Adresse fortgesetzt, die im Operanden des jeweiligen JP-Befehls programmiert werden soll. Diese Tatsache kann sich als vorteilhaft erweisen, z. B. dann, wenn der Programm-Teil mit dem Sprung-Ziel nicht verschoben worden ist, wenn sich die Ziel-Adressen also nicht geändert haben. Sie kann aber auch nachteilig sein, wenn z. B. der Sprung-Befehl die Fortsetzung eines Programms bei einem Label veranlassen soll, dessen Adresse bei der Ausführung des MOVE-Kommandos ebenfalls verschoben worden ist. In einem solchen Fall muß nach der Ausführung der Verschiebung eines Daten-Blocks mit Programm-Daten dieses verschobene Programm sorgfältig daraufhin untersucht werden, ob ggf. die Operanden von JP-Befehlen den neuen Adressen der Sprung-Ziele angepaßt werden müssen.

Wir werden Ihnen im folgenden Abschnitt einen Sprungbefehl vorstellen, der diese Nachteile vermeidet und der außerdem nur zwei Byte lang ist.

Unbedingter Sprung mit relativer Adressierung

Sie haben bisher in unserem Lehrgang vorzugsweise solche Befehle kennengelernt, die sowohl im Befehlssatz des Mikroprozessors 8085 als auch in dem des Z 80 enthalten sind. Tatsächlich enthält der Mikroprozessor Z 80, der in Ihrem System verwendet wird, eine Reihe weiterer Befehle, die wir im Rahmen dieses Lehrgangs nicht berücksichtigen können.

Im Zusammenhang mit den Sprungbefehlen sollen Sie hier jedoch einen Z 80-spezifischen Befehl kennenlernen, der im Befehlssatz des 8085-Mikroprozessors nicht enthalten ist und der einen unbedingten Sprung veranlaßt, welcher als Operand keine absolute Adresse enthält, der vielmehr das Sprungziel Programmzähler-relativ angibt. Dieser Sprung-Befehl veranlaßt also nicht die Fortsetzung des Programms bei einer fest vorgegebenen Adresse, sondern sagt: Der nächstfolgend auszuführende Befehl ist von der augenblicklich im Programmzähler stehenden Adresse um einen bestimmten Betrag entfernt. Das Sprung-Ziel wird demnach auf den derzeitigen Inhalt des Programmzählers bezogen; der Sprung wird Programmzähler-relativ ausgeführt.

Die Programmierung eines solchen Sprungs hat den Vorteil, daß der Sprung-Befehl auch dann unverändert bleiben kann, wenn ein Datenblock mit Programm-Daten im Speicher verschoben wird und wenn das Sprung-Ziel innerhalb des verschobenen Datenblocks liegt.

Praktisch läuft eine solche Programmierung darauf hinaus, daß der Programmzähler-Inhalt vom Sprung-Befehl nicht grundsätzlich geändert wird, sondern daß der Abstand (im Englischen bezeichnet man diesen Abstand als *Displacement* – sprich: displäissment –) des Sprung-Ziels zum derzeitigen Inhalt des Programmzählers einfach addiert bzw. von ihm subtrahiert wird. Das Ergebnis dieser Rechnung, die vom Mikroprogramm in der CPU vorgenommen wird, ist dann die Adresse des als nächsten auszuführenden Befehls im Programmzähler.

Sehen Sie sich bitte das Bild S 79.1a an! Wir haben schematisch die Anordnung des absolut adressierenden Sprung-Befehls JP NEXT mit dem Operationscode C3 und den Operanden 16 und 19 im Speicher dargestellt und angenommen, daß dem Label NEXT die Adresse 1916 zugeordnet ist. Bei der Ausführung dieses Befehls zeigt der Programmzähler-Inhalt nach dem Einholen des zweiten Operanden (19) auf die Speicherzelle mit der Adresse 1913. Wenn der Befehl jetzt CPU-intern ausgeführt wird, dann werden die beiden eingeholten Operanden als Adresse 1916 in den Programmzähler gebracht und als nächster wird der bei dieser Adresse stehende Befehl ausgeführt. – Dieser Vorgang ist Ihnen bereits geläufig.

Sehen Sie sich jetzt bitte das Teilbild b an! Dort steht im Speicher der Befehl JR NEXT, der offenbar die gleiche Wirkung wie der Befehl JP NEXT hat, aber um ein Byte kürzer ist.

Sie können sich die Ausführung dieses Befehls in einem Versuch ansehen. Dabei erkennen Sie bereits jetzt, daß es sich um keinen absolut adressierenden Befehl handelt.

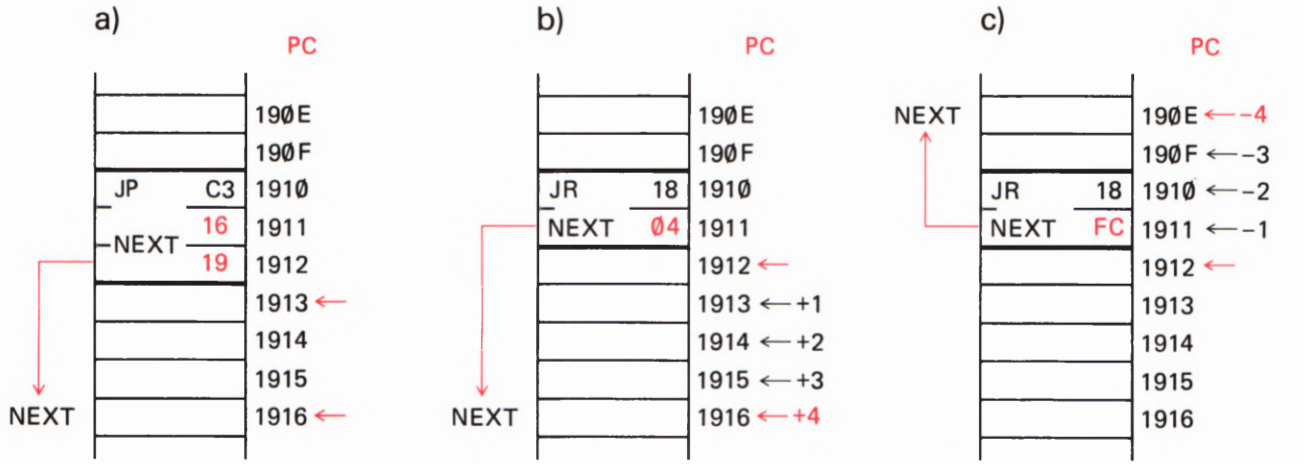


Bild S79.1
a) Der JP-Befehl enthält als Operanden die Adresse, bei der das Programm fortgesetzt werden soll. In den Teilbildern b und c wird dargestellt, wie der Operand des JR-Befehls zum Inhalt des Programmzählers addiert bzw. von dessen Inhalt subtrahiert wird.

Versuch S79.1

Sprung mit relativer Adressierung

Tasten Sie in Ihr System bei der Adresse 191Ø das Byte 18 ein und bei der folgenden Adresse das Byte Ø4. – Lassen Sie den jetzt eingetasteten Befehl in einem Einzelschritt ausführen! Sie brauchen dazu nur mit der Taste – die Adresse 191Ø einzustellen und die Taste STEP zu betätigen. In der Anzeige erscheint nach Ausführung des Befehls als aktueller Inhalt des Programmzählers die Adresse 1916. Beim weiteren Ablauf des Programms würde der Befehl ausgeführt, dessen Operationscode bei dieser Adresse programmiert ist.

Das Bild S79.1b macht deutlich, was bei der Ausführung des Befehls JR NEXT geschehen ist: Nach dem Einholen des Operanden Ø4 dieses Befehls zeigt der Inhalt des Programmzählers auf die Adresse 1912. Wenn es sich bei dem gerade in Arbeit befindlichen Befehl, z. B. um einen Verknüpfungbefehl handelte, dann würde jetzt CPU-intern die Verknüpfung durchgeführt und anschließend der Befehl ausgeführt, dessen Operationscode bei der Adresse 1912 steht, auf die der Programmzähler jetzt zeigt.

Beim Befehl JR NEXT handelt es sich aber um keinen Verknüpfungsbefehl. Der JR-Befehl bewirkt vielmehr, daß der Operand dieses Befehls zum aktuellen Inhalt des Programmzählers addiert wird. In unserem Versuch ist der Operand Ø4 und die Addition 1912**H** + Ø4**H** ergibt 1916**H**, also gerade die Adresse des Labels NEXT, zu der der Sprung JR NEXT führen soll.

Aufgabe S79.1

Welchen Operanden muß ein bei der Adresse 18AB programmierter Befehl JR (Operationscode 18) haben, wenn dieser Befehl die Fortsetzung des Programms bei der Adresse 18F5 bewirken soll?

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü13.

Das Bild S79.1c erläutert, daß der Befehl mit dem mnemonischen Code JR auch einen Rückwärts-Sprung im Speicher bewirken kann, also die Fortsetzung eines Programms bei einer niedrigeren Adresse. Ein solcher Sprung muß ausgeführt werden, wenn ein Programm in einer Schleife laufen soll.

Versuch S 80.1

Rückwärts-Sprung mit relativer Adressierung

Tasten Sie in Ihr System ab der Adresse 191Ø die beiden Bytes 18 und FC für den Befehl JR NEXT ein! Lassen Sie diesen Befehl in einem Einzelschritt ausführen! – Nach dem Betätigen der Taste STEP wird Ihnen als aktueller Inhalt des Programmzählers die Adresse 19ØE angezeigt.

Dezimal	Sedez. Byte
+127	7F
+126	7E
.	.
.	.
.	.
+101	65
+100	64
+ 99	63
.	.
.	.
.	.
+ 3	Ø3
+ 2	Ø2
+ 1	Ø1
— 0 —	ØØ —
− 1	FF
− 2	FE
− 3	FD
.	.
.	.
.	.
− 99	9D
−100	9C
−101	9B
.	.
.	.
.	.
−127	81
−128	8Ø

Das Bild S 79.1c erläutert den Vorgang: Nach dem Einholen des Operanden FC zeigt der Programmzähler auf die Adresse 1912. Das Byte FC stellt sedezimal den Wert minus vier dar; der Befehl JR NEXT addiert in diesem Fall zur Adresse 1912 den negativen Wert vier: 1912**H** + (FC**H**) = 19Ø**E**H.

Ohne die Theorie der Darstellung negativer Zahlen in sedezimaler Schreibweise hier aufzugreifen, zeigen wir auf dem Rand dieser Seite, wie man negative Zahlen bis −128 sedezimal anschreiben kann. Wir werden Ihnen gleich zeigen, wie Sie mit Ihrem System die Operanden relativ adressierender Sprungbefehle ganz einfach ermitteln können. Vorher stellen wir Ihnen aber wie gewohnt den Zwei-Byte-Befehl für einen relativ adressierenden Sprung-Befehl vor:

Mnemonischer Code	Operations-code	Operation	Erläuterung
JR op	18	PC←(PC)+op	Der Operand op des Befehls wird zum aktuellen Inhalt des Programmzählers addiert.

Der mnemonische Code JR kommt vom englischen **J***ump* **R***elative.*

Die sehr komfortable Möglichkeit, mit Ihrem System den Operanden op des relativ adressierenden, unbedingten Sprungbefehls JR op zu berechnen, zeigen wir Ihnen am Beispiel des Befehls Nr. 14, JR START im Programm im Bild S 64.1. Wir nehmen dabei an, Sie hätten gerade den Operationscode 18 des JR-Befehls bei der Adresse 1816 eingetastet und wollten nun den (noch unbekannten) Operanden eingeben.

Am besten vollziehen Sie die hier beschriebenen Schritte an Ihrem System gleich nach:

Betätigen Sie nach der Eingabe des Operationscodes 18 bei der Adresse 1816 **nicht** die Taste + zur Adressen-Inkrementierung, sondern stattdessen die Taste RELA (von links gezählt zweite Taste in der zweiten Reihe des Tastenfeldes). Die jetzt erscheinende Anzeige 1.8.1.6. – S weist die Adresse des Operationscodes des Sprung-Befehls als **S**tart-Adresse des Sprungs aus. – Betätigen Sie jetzt die Taste +! Die Anzeige X.X.X.X. – D verlangt die Eingabe der Ziel-Adresse des Sprungs (D von **D***estination* – sprich: destinäischen –, Bestimmung). Im Bild S 64.1 liegt die Ziel-Adresse bei Label START, also bei der Adresse 18ØØ. – Wenn Sie diese Adresse eintasten und anschließend die Taste GO betätigen, dann stellen Sie fest, daß Ihr System bei der Adresse 1817 gleich den richtigen Operanden E8 für den Befehl JR START eingesetzt hat.

Bedingt auszuführende Befehle

Im Anschluß an die Befehle JP (HL) und JP (adr) für unbedingte Sprünge sollen hier die Sprungbefehle erläutert werden, die nur bedingt ausgeführt werden. Bei dieser Art von Befehlen wird die Befehls-Ausführung, hier also die Fortsetzung des Programms bei einer anderen als der nächstfolgenden Adresse, von der Einhaltung bestimmter Bedingungen abhängig gemacht. Ist die jeweils vorausgesetzte Bedingung erfüllt, dann wird der Befehl ausgeführt. Ist die Bedingung jedoch nicht erfüllt, dann tut der Mikroprozessor so, als sei der Befehl überhaupt nicht vorhanden. Er führt dann die Anweisung aus, die in der nächstfolgenden Speicherzelle als Befehl abgelegt ist.

Das Flag-Register

Es gibt drei Befehls-Gruppen, deren Ausführung von der Erfüllung einer bestimmten Bedingung abhängig gemacht werden kann. Letztlich handelt es sich bei allen drei Gruppen um die bedingte Fortsetzung des Programms bei einem anderen als dem im Speicher nächstfolgenden Befehl. Hier sollen uns einfache Sprung-Anweisungen – ähnlich dem JP-Befehl – beschäftigen. Dabei ergeben sich jedoch zwei ganz grundsätzliche Fragen:

1. Von welchen Bedingungen kann die Ausführung eines Befehls abhängig gemacht werden? Und:
2. Woher weiß der Mikroprozessor, ob eine Bedingung, von der die Ausführung des Befehls abhängig gemacht wird, erfüllt ist oder nicht?

Natürlich kann man die Ausführung des Befehls nicht ohne weiteres vom gerade herrschenden Wetter abhängig machen oder davon, ob das Mittagessen angebrannt ist oder nicht. Durchaus sinnvoll dagegen ist es, als Bedingung für die Ausführung eines Befehls das Ergebnis eines vorher ausgeführten Befehls zu setzen.

Der Befehlssatz unserers Mikroprozessors enthält eine Reihe von Befehlen, welche die Inhalte von Registern oder per Programm konstant eingegebene Werte mit dem Inhalt eines anderen Registers verknüpfen. Von diesen Befehlen haben Sie die Befehle für logische Verknüpfungen bereits kennengelernt.

Eine weitere Gruppe solcher Befehle dient arithmetischen Verknüpfungen, also der Addition oder Subtraktion von festen Werten oder Register-Inhalten zum oder vom Inhalt eines anderen Registers. Das Ergebnis der Ausführung solcher Befehle hängt vom Inhalt der Register ab, die an der Befehlsausführung beteiligt sind.

Der Witz der Sache ist nun, daß der Mikroprozessor bestimmte Merkmale des Ergebnisses solcher Verknüpfungen in einer abgekürzten Form speichert. Wohlgemerkt: Die CPU speichert nicht das ganze Ergebnis einer solchen Verknüpfung. Das ist häufig nicht einmal so sehr interessant. Viel interessanter ist es, ob das Ergebnis bestimmte, charakteristische Merkmale zeigt. Solche charakteristischen Merkmale sind z. B., ob das Ergebnis einer Verknüpfung eine positive oder negative Zahl darstellt, ob sich diese Zahl sedezimal zweistellig darstel-

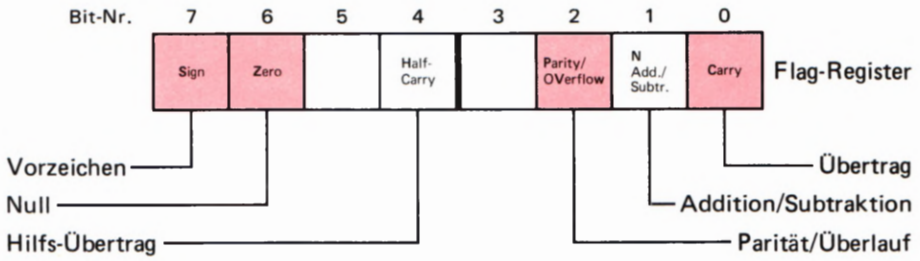


Bild S82.1
Die Bits im Flag-Register haben die hier angeschriebenen Bedeutungen. Die nicht bezeichneten Bits Nr. 3 und Nr. 5 werden nicht verwendet.

len läßt oder ob das Ergebnis vielleicht gerade null ist. Manchmal möchte man auch wissen, ob die Anzahl der 1-Bits im sich ergebenden Bitmuster gerade oder ungerade ist.

Für alle von uns aufgezählten Merkmale gilt, daß sie entweder vorhanden sind oder nicht vorhanden sind. Keines dieser Merkmale kann mehr oder weniger oder vielleicht nur ein bißchen vorhanden sein. Das heißt: Die aufgezählten Merkmale lassen sich durch die Zeichen Ø oder 1 eindeutig charakterisieren. Jedes Merkmal ist durch ein Bit dargestellt.

Ein Merkmal ist nichts anderes als ein Merker, also ein Flag (Seite S7). Wir haben vier Merkmale für die Ergebnisse der Ausführung bestimmter Befehle aufgeführt. Tatsächlich kann der Z80-Mikroprozessor sechs Merker zur Kennzeichnung des Ergebnisses der Ausführung bestimmter Befehle setzen. (Beim 8085-Mikroprozessor sind es fünf.) Für diese Merker ist in der CPU sowohl im Haupt-Registersatz als auch im 2. Registersatz (vgl. Seite T2) jeweils ein eigenes Register angelegt worden, das – wie es sich für einen Acht-Bit-Mikroprozessor gehört – natürlich acht Bit „breit" ist. Dabei nimmt man es in Kauf, daß zwei (beim 8085 drei) Bits dieses Registers unbenutzt bleiben. Das Register wird als **Flag-Register** bezeichnet (Seite T2). Da nach der Ausführung von Befehlen, die Bits in diesem Register setzen oder löschen, die CPU in einen bestimmten Zustand gebracht wird, bezeichnet man dieses Register auch als **Status-Register**. (Status = Zustand.)

Das Bild S82.1 zeigt die Bedeutung der einzelnen Bits im Flag-Register. Die Merker (Flags) scheinen sämtlich nur im Zusammenhang mit arithmetischen Verknüpfungen von Interesse zu sein. In Programmen wird jedoch weit häufiger in ganz anderen Zusammenhängen nach den Werten von Status-Bits gefragt. Das gilt insbesondere für das Bit Nr. 6, das Zero- (Null-) Bit.

Wir können im Rahmen dieses Lehrgangs nicht auf die vielfältigen Möglichkeiten eingehen, die sich durch die Nutzung der verschiedenen Status-Bits ergeben. Hier stellen wir Ihnen nur die allerwichtigsten Bedeutungen von vier der Merker im Flag-Register vor.

Das **Sign-Bit** (Nr. 7) ist ein Merkmal dafür, ob nach der Ausführung bestimmter Befehle das Byte im Akkumulator eine positive oder eine negative Zahl darstellt. Das englische Wort *sign* (sprich: ssein) bedeutet wörtlich ein Zeichen, oder in mathematischem Zusammenhang ein Vorzeichen.

Wenn Sie die sedezimal angeschriebenen Bytes auf dem Rand der Seite S80 als Bitmuster betrachten, dann erkennen Sie, daß bei der Darstellung negativer Zahlen das Bit Nr. 7 immer den Wert 1 hat. (Lesen Sie

noch einmal ab der Seite H31 nach!) Wenn sich als Ergebnis einer arithmetischen Operation des Mikroprozessors im Akkumulator ein Byte einstellt, in dem das Bit Nr. 7 den Wert 1 hat, dann wird dieses 1-Bit automatisch an die Bit-Stelle Nr. 7 des Flag-Registers kopiert. Der Mikroprozessor merkt sich dadurch, daß ein negatives Rechenergebnis vorliegt. Hat das Bit Nr. 7 im Akkumulator nach der arithmetischen Operation den Wert Ø, dann erscheint an der Bit-Stelle Nr. 7 des Flag-Registers der Wert Ø. Damit wird angemerkt, daß das Rechenergebnis positiv war. Interessant dabei ist, daß die Zahl Null als positive Zahl behandelt wird.

Das **Zero-Bit** (Nr. 6) zeigt an, ob nach der Ausführung bestimmter Befehle im gerade angesprochenen Register der Wert ØØ erschienen ist. Bitte merken Sie sich: Wenn im angesprochenen Register nach einem solchen Befehl das **Byte ØØ** steht, dann (und nur dann!) nimmt das **Zero-Bit** im Flag-Register den **Wert 1** an. **Zero-Bit = 1** bedeutet also: Die **Null-Bedingung** im Register ist **erfüllt.**

Das englische Wort *Zero* (sprich: sierou) bedeutet wörtlich: Null. Das Zero-Bit ist das beim Programmieren wohl am häufigsten abgefragte Bit. Der Wert ØØ in einem Register ist so charakteristisch, daß sich von seinem Vorhandensein sehr elegante Programm-Verzweigungen abhängig machen lassen.

Das **Parity/Overflow-Bit** wird dagegen in Programmen verhältnismäßig selten abgefragt. Wir wollen hier nur erwähnen, daß es mit dem Wert 1 nach bestimmten Operationen im angesprochenen Register eine geradzahlige, also ohne Rest durch zwei teilbare Anzahl von 1-Bits meldet. Außerdem spielt sein Wert bei bestimmten arithmetischen Operationen eine Rolle.

Das **CY-Bit** wird hauptsächlich im Zusammenhang mit arithmetischen Operationen verwendet. Es kann als Ein-Bit-Erweiterung eines Acht-Bit-Registers betrachtet werden. Die Bezeichnung CY ist eine Abkürzung des englischen Worts **C***arr***Y** (sprich: kärrie). In mathematischem Zusammenhang bedeutet *carry* einen beim Addieren entstehenden Übertrag.

Obwohl wir uns mit den interessanten Fähigkeiten des Carry-Bits in diesem Lehrgang kaum beschäftigen können, sollen Sie wissen, daß dieses Bit das einzige Bit im Flag-Register ist, das durch spezielle Befehle manipuliert werden kann:

Mnemonischer Code	Operations-code	Operation	Erläuterung
SCF	37	CY ← 1	Das CY-Bit wird auf den Wert 1 gesetzt.
CCF	3F	CY ← $(\overline{CY})$	Das CY-Bit wird invertiert.

Der mnemonische Code SCF kommt vom englischen **S***et* **C***arry*-**F***lag* und bedeutet: Carry-Flag Setzen. Der mnemonische Code CCF kommt vom englischen **C***omplement* **C***arry*-**F***lag.* Dieser Befehl bewirkt eine Umkehrung des Werts des CY-Flags.

Zunächst mag Ihnen die Beschreibung der Eigenschaften von Bits im Flag-Register reichlich theoretisch vorkommen. Erinnern Sie sich aber

bitte: Die im Flag-Register abgelegten Informationen stellen die Bedingungen dafür dar, ob ein Befehl für einen bedingten Sprung ausgeführt werden soll oder nicht. Die Befehle für bedingte Sprünge machen es möglich, Verzweigungen zu programmieren. Und in dieser Möglichkeit liegt die ganze sogenannte Intelligenz eines Mikroprozessors. Will man davon sinnvollen Gebrauch machen, dann muß man natürlich die Verzweigungs-Bedingungen kennen, die als Informationen im Flag-Register abgelegt sind.

Je nach den Verzweigungs-Bedingungen führt der Mikroprozessor bestimmte Befehle aus, oder er verweigert die Ausführung dieser Befehle. Natürlich kann auch ein Hund die Ausführung eines Befehls verweigern. Das ist aber ganz sicher kein Zeichen von Intelligenz. Der Hund verweigert die Ausführung eines Befehls, weil er den Befehl einfach nicht ausführen kann. Entweder ist der Befehl unsinnig („sag' mal Papa!"), oder das arme Tier hat sich ein Bein gebrochen. Wenn er einen Befehl ausführen kann und tut es trotzdem nicht, dann ist der Hund nicht intelligent, sondern ungezogen. Ein Zeichen von Intelligenz wäre es, wenn er den Befehl bekommt, den Briefträger zu beißen und der Hund tut es nicht, weil er überlegt: Beiße ich den Briefträger, dann kommt Herrchen ins Gefängnis. Also tue ich es nicht. Einen solchen Hund suchen wir auch noch. (Oder vielleicht doch nicht?)

Hier liegt der Unterschied: Ein Mikroprozessor ist nicht ungezogen und verweigert die Ausführung eines Befehls nicht nach Lust und Laune. Er prüft vielmehr die ihm vorgegebenen Bedingungen sorgfältig und entscheidet dann absolut logisch: Tu ich's oder tu ich's nicht?

Die Bedingungen, von denen der Mikroprozessor seine Entscheidungen abhängig machen kann, sind die Werte der Merker-Bits im Flag-Register, die aus diesem Grunde auch **Bedingungs-Bits** genannt werden.

Sprungbefehle, deren Ausführung vom Zero-Bit abhängig ist

Auf der Seite S 83 haben wir bereits angedeutet, daß das Zero-Bit Z das wohl am häufigsten abgefragte Bit des Flag-Registers ist. Sie erinnern sich: Das Zero-Bit meldet mit seinem Wert, ob das zuletzt mit einem logischen oder arithmetischen Verknüpfungsbefehl angesprochene Register den Inhalt ØØ hat oder nicht. Daß die Tatsache des Inhalts ØØ mit dem Wert Z = 1 gekoppelt ist, brauchen Sie während des Programmierens keineswegs im Kopf zu behalten. Das ist nur dann interessant, wenn Sie ein Programm zur Kontrolle z. B. in Einzelschritten ablaufen lassen und sich dann den Inhalt des Flag-Registers ansehen.

> Das Zero-Bit im Flag-Register wird – abgesehen von den Befehlen INC rp, DEC rp, ADD HL,rp und CPL – von sämtlichen logischen und arithmetischen Verknüpfungsbefehlen beeinflußt. Alle anderen Befehle lassen den Inhalt des Zero-Bits unbeeinflußt.

(Die Ihnen in der Aufzählung unbekannten Befehle haben wir nur der Vollständigkeit aufgeführt.)

Es gibt zwei Sprungbefehle, die nur in Abhängigkeit vom Wert des Zero-Bits ausgeführt werden, die aber in verschiedenen Versionen verwendet werden können:

Mnemonischer Code	Operationscode	Operation	Erläuterung
JP Z,adr	CA	PC←adr wenn Z=1	Wenn das Zero-Bit mit dem Wert 1 den Register-Inhalt ØØ meldet, wird das Programm bei der absolut angegebenen Adresse adr bzw. bei der um den Wert op vom aktuellen PC-Inhalt entfernten Adresse fortgesetzt. Andernfalls wird der Befehl ignoriert.
JR Z,op	28	PC←(PC)+op wenn Z=1	
JP NZ,adr	C2	PC←adr wenn Z=Ø	Wenn das Zero-Bit mit dem Wert Ø den Register-Inhalt ≠ ØØ meldet, wird das Programm bei der absolut angegebenen Adresse adr bzw. bei der um den Wert op vom aktuellen PC-Inhalt entfernten Adresse fortgesetzt. Andernfalls wird der Befehl ignoriert.
JR NZ,op	2Ø	PC←(PC)+op wenn Z=Ø	

Die mnemonischen Codes JP (von **J***um***P**) und JR (von **J***ump* **R***elative*) sind Ihnen bereits bekannt (Seiten S74 und S80). Im Gegensatz zu den Befehlen für einen unbedingten Sprung wird bei den hier vorgestellten Befehlen die Sprung-Bedingung in den mnemonischen Code einbezogen: Z für gesetztes Zero-Bit (=1) im Flag-Register, also für den Wert Ø Ø im jeweiligen Register, und NZ für nicht gesetztes Zero-Bit (=Ø) im Flag-Register, also für einen vom Wert ØØ abweichenden Wert im jeweiligen Register.

Die Befehle JP Z,adr und JP NZ,adr sind Drei-Byte-Befehle mit zwei Operanden, die das HOB und das LOB der Ziel-Adresse des Sprungs enthalten. Auch hier gilt wieder, was bei allen Drei-Byte-Befehlen gilt, deren Operand eine Adresse darstellen kann:

Das **LOB** wird im Speicher bei der **niederen Adresse** abgelegt;
das **HOB** wird im Speicher bei der **höheren Adresse** abgelegt.
(Vgl. Seite S76 und Bild S72.1.)

Im Gegensatz zu den JP-Befehlen, die der Mikroprozessor 8085 in gleicher Weise wie der Z80 ausführen kann, gibt es die JR-Befehle nur beim Mikroprozessor Z80. Die hier vorgestellten JR-Befehle für bedingte Sprünge sind – wie der JR-Befehl für einen unbedingten Sprung (Seite S80) – Zwei-Byte-Befehle. Für die Berechnung des Displacements, das im Operanden des Befehls angegeben wird, gelten die gleichen Überlegungen, die wir ab der Seite S78 angestellt haben.

S
86

Auch bei diesen Befehlen kann die komfortable Möglichkeit des Mikro-Professors zur Berechnung des Operanden op ausgenutzt werden, die wir auf der Seite S 80 beschrieben haben.

Die vom Wert des Zero-Bits bedingten Sprünge können Sie sich in einem Versuch ansehen. Betrachten Sie bitte den Programmablaufplan im Bild S 86.1! Über das Tastenfeld Ihres Systems werden nacheinander zwei sedezimal einstellig dargestellte Zahlen (Ø...F) eingegeben. Jede dieser Zahlen erscheint in der Anzeige. Außerdem werden die beiden Zahlen in unterschiedlichen CPU-Registern gespeichert.

Mit der Anweisung Nr. 2 wird die erste der eingetasteten Zahlen in den Akkumulator geholt und anschließend (Nr. 3) wird die erste mit der zweiten Zahl verknüpft. Wir verwenden im folgenden Versuch eine subtrahierende, also eine arithmetische Verknüpfung, da dann die Verknüpfung besonders übersichtlich ist.

Ab der Anweisung Nr. 4 wird das Verknüpfungsergebnis angezeigt. Je nachdem, ob die erste der eingegebenen Zahlen größer oder kleiner als die zweite Zahl ist, stellt sich ein positives oder negatives Verknüpfungsergebnis ein. Sind beide eingegebenen Zahlen gleich, dann ist das Ergebnis null.

Die Anweisung Nr. 7 enthält einen durch das Ergebnis der Verknüpfung bedingten Sprung. Wir setzen den Befehl JP Z,adr ein. Das bedeutet: Dann (und nur dann), wenn das Ergebnis der Verknüpfung null ist, wenn also im Flag-Register nach Ausführung der subtrahierenden Verknüpfung das Zero-Bit gleich 1 gesetzt ist, erfolgt ein Sprung zu einem Programmteil ALARM. Ist das Ergebnis der Verknüpfung ungleich null, dann wird der Befehl JP Z,adr nicht ausgeführt. (In der Auflistung der mnemonischen Codes haben wir statt der allgemeinen Angabe adr den Label ALARM eingesetzt, der für die Adresse des ersten Befehls der ALARM-Routine steht.) Das Programm geht in diesem Fall sofort zur Anweisung Nr. 8 über, die einen unbedingten Sprung zum Label RECHN verursacht. Es können dann sofort zwei neue Zahlen eingegeben werden.

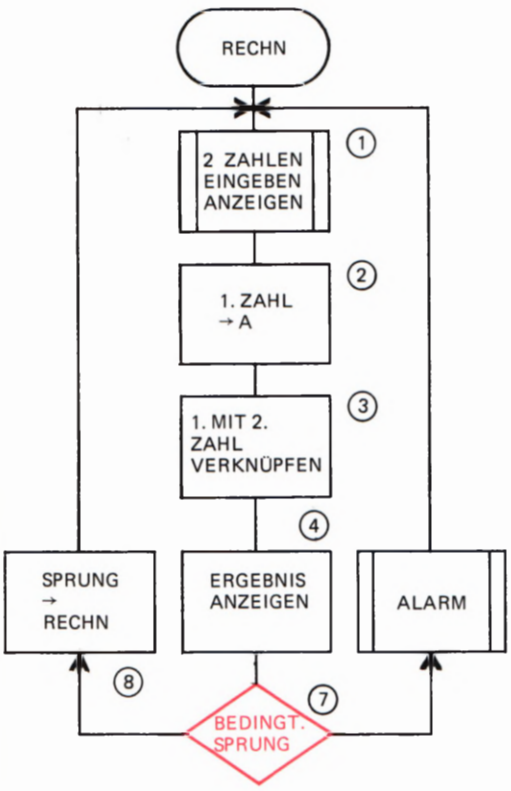


Bild S 86.1
Das diesen Ablaufplan entsprechende Programm springt den Programmteil ALARM in Abhängigkeit vom Verknüpfungsergebnis zweier Bytes an.

Versuch S 86.1

Bedingter Sprung beim Ergebnis null

Das Programm zu diesem Versuch, dessen Ablauf im Bild S 86.1 dargestellt ist, haben wir im Bild S 87.1 aufgelistet. Der Programmteil EINGAB ist als Unterprogramm ausgeführt, um das Hauptprogramm übersichtlich zu halten. Dieses Unterprogramm und auch den Programmteil ALARM haben wir im Bild S 87.2 in Form eines sogenannten Hex-Dumps (sprich: hex-damp) aufgelistet, also nur die hexadezimal (sedezimal, vgl. Seite S 14) dargestellten Bytes angegeben, weil die Struktur dieser Programmteile hier nicht interessiert.

Tasten Sie bitte das Programm aus dem Bild S 87.1 und anschließend ab der Adresse 181Ø die Bytes aus dem Bild S 87.2 in Ihr System ein. – Starten Sie das Programm bei der Adresse 18ØØ!

In der Anzeige erscheinen zunächst ganz undefinierte Sieben-Segment-Muster, die Sie nicht zu stören brauchen. Betätigen Sie eine beliebige Zifferntaste, z. B. die Taste 9. In der Anzeige erscheint in diesem Fall die Ziffer 9 mit nachfolgenden Minus-Zeichen. (Wir haben ja ein Subtraktions-Programm programmiert.)

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	RECHN	CALL	EINGAB	1800	CD	10	18	2 Zahlen eingeb., anzeig.
2		LD	A,E	1803	7B			1. Zahl → A
3		SUB	D	1804	92			1. mit 2. Zahl verknüpfen
4		PUSH	AF	1805	F5			}
5		CALL	HEX7SG	1806	CD	78	06	} Ergebnis → Anzeige
6		POP	AF	1809	F1			}
7		JP	Z,ALARM	180A	CA	3E	18	Sprung, bed. durch Ergebnis
8		JR	RECHN	180D	18	F1		Nächste Eingabe

Bild S87.1
Hauptprogramm zum Versuch S86.1. Damit das Programm lauffähig ist, müssen auch die als Hex-Dump angegebenen Programmteile im Bild S87.2 eingegeben werden.

Betätigen Sie eine zweite Zifferntaste, z.B. die Taste 4. Die Anzeige wird jetzt zu 9 − 4 = 05 ergänzt. Diese Anzeige bedarf keines Kommentars. − Betätigen Sie nacheinander beliebige Paare von Zifferntasten! Nun, das ist sicher ein ganz hübsches Programm, aber damit könnte man bestenfalls einem Erstklässler imponieren. So scheint es. Bemerkenswert ist höchstens, daß beim Subtrahieren einer größeren Zahl von einer kleineren im Ergebnis an der zweiten Stelle von rechts die Ziffer F erscheint. Das Ergebnis der Subtraktion sechs minus neun, das gleich minus drei ist, wird vom Mikroprozessor als FD angezeigt. Vergleichen Sie diese Anzeige mit der Aufstellung auf dem Rand der Seite S80! Der Versuch bestätigt die Richtigkeit der Tabelle.

Auf dieses Ergebnis kommt es uns hier gar nicht an. − Betätigen Sie zweimal nacheinander eine gleiche Zifferntaste! Zum Beispiel: A − A = 00. Was geschieht?

Der Subtraktionsbefehl Nr. 3, SUB D, bringt das Byte 00 in den Akkumulator und setzt wegen dieses Ergebnisses das Zero-Bit im Flag-Register auf 1. Im Gegensatz zu allen vorhergehenden Rechen-Ergebnissen wird jetzt der Befehl JP Z,ALARM ausgeführt und der Programmteil ALARM wird angesprungen. Am Ende dieses Programmteils haben wir bei der Adresse 1850 einen unbedingten Sprung zum Label RECHN programmiert, so daß nach dem Ablauf der ALARM-Routine neue Zahlen eingetastet werden können. (Während des Ablaufs dieser Routine kann die Anzeige vom Programm nicht bedient werden. Deshalb erscheint das Ergebnis 00 erst, wenn der Alarm abgelaufen ist.)

```
EINGAB
 1810 DD 21  00  1A  21  06  1A  CD
 1818 28  18  CD 35  18  5A  36  02
 1820 CD 28  18  36  82  2E  00  C9
 1828 2B  E5  CD FE  05  57  CD 89
 1830 06  E1  77  2B  C9  36  00  2D
 1838 F2  35  18  2E  04  C9

ALARM
 183E                        06  18
 1840 C5  21  18  00  CD DE  05  21
 1848 10  00  CD E2  05  C1  10  F0
 1850 C3  00  18
```

Bild S87.2
Die hier als Hex-Dump angegebenen Programmteile gehören zu dem im Bild S87.1 angeschriebenen Hauptprogramm, das im Versuch S86.1 verwendet wird.

Aufgabe S87.1

Am Ende des Unterprogramms ALARM (Bild S87.1) ist ab der Adresse 1850 mit den Bytes C3 00 18 der absolut adressierende, unbedingte Sprung JP 1800 zum Label RECHN (Adresse 1800) programmiert. (Vgl. Seite S76.)

Der Befehl JP RECHN = JP 1800 soll durch den Programmzähler-relativen, unbedingten Sprungbefehl JR RECHN ersetzt werden. − Mit welchen Tasten veranlassen Sie Ihr System, den Operanden dieses JR-Befehls anzuzeigen? (Vgl. Seiten S80 und Ü13, zu Aufgabe S79.1.)

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü13.

Das Tastenfeld des Mikro-Professors bietet Ihnen die Möglichkeit, den Inhalt des Flag-Registers nach der Ausführung des JP Z,ALARM-Befehls zu inspizieren. − Halten Sie dazu das Programm mit der Taste RS an und setzen Sie einen Breakpoint (Seite S26) bei der Anfangs-

adresse 183E der ALARM-Routine, die nur beim Rechenergebnis ØØ angesprungen wird:

ADDR, 1, 8, 3, E, SBR

Starten Sie danach das Programm wieder bei der Adresse 18ØØ und lassen Sie einige Rechnungen mit einem Rechenergebnis ungleich null ausführen! Ihr System kümmert sich dabei um den gesetzten Breakpoint überhaupt nicht. Sobald Sie aber eine Rechnung mit dem Ergebnis ØØ ausführen lassen, wird die Breakpoint-Adresse 183E angesprungen und der dort programmierte Befehl LD B,18 ausgeführt. Dann bleibt das Programm stehen und in der Anzeige erscheint die Adresse 184Ø des nächstfolgenden Befehls.

Betätigen Sie die Tasten REG und AF! Es erscheint die Anzeige, die wir im Bild S 88.1a dargestellt haben. Die beiden mittleren Anzeigestellen melden Ihnen den Inhalt des Flag-Registers, den wir im Teilbild b als Bitmuster dargestellt haben. Wir interessieren uns hier nur für den rot unterlegten Wert des Bits Zero, das erwartungsgemäß den Wert 1 hat.

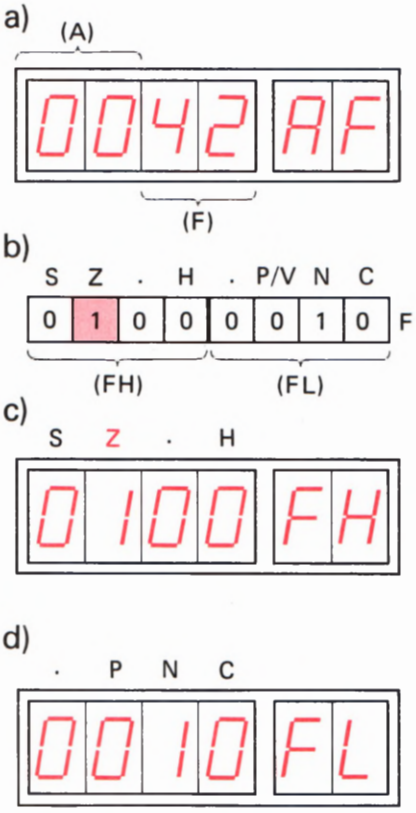


Bild S 88.1
a) Anzeige nach Betätigung der Tasten REG und AF. b) Inhalt des Flag-Registers. Nach Betätigen der Tasten SZ.H bzw. .PNC erscheinen die in den Teilbildern c und d dargestellten Bitmuster aus dem Flag-Register.

Der Mikro-Professor macht Ihnen die Inspektion des Flag-Register-Inhalts noch leichter. Betätigen Sie die Taste SZ.H an der (von rechts gezählt) vierten Stelle der oberen Reihe des Tastenfelds! Im Teilbild c ist die jetzt erscheinende Anzeige dargestellt: Die Buchstaben FH an den rechten beiden Anzeigestellen sagen, daß vom Flag-Register-Inhalt die obere (High) Hälfte als Bitmuster in den linken vier Anzeigestellen gezeigt wird. Die Beschriftung SZ.H der betätigten Taste gibt Ihnen die Bedeutung der einzelnen Bits an; der Punkt sagt, daß das zugehörige Bit Nr. 5 keine Bedeutung hat (vgl. S 82.1).

Nur interessehalber können Sie jetzt auch die Taste .PNC rechts neben der Taste SZ.H betätigen. Vom Flag-Register wird die untere (Low) Hälfte als Bitmuster angezeigt. (Vergleichen Sie die Teilbilder c und d mit dem im Teilbild b dargestellten Inhalt des Flag-Registers!)

Heben Sie den in diesem Versuch gesetzten Breakpoint bei der Adresse 183E durch die Betätigung der Tasten RS und CBR wieder auf!

Versuch S 88.1

Bedingter Sprung beim Ergebnis ungleich null

Ersetzen Sie bei der Adresse 18ØA den Operationscode CA des Befehls JP Z,adr durch den Operationscode C2 des Befehls JP NZ,adr! – Starten Sie das Programm bei der Adresse 18ØØ und lassen Sie einige Rechnungen mit Ergebnissen ungleich null und gleich null ausführen!

Im jetzt geänderten Programm ist die Sprungbedingung beim Rechenergebnis ungleich null erfüllt und damit ertönt der Alarm immer dann, wenn nach Ausführung des Subtraktions-Befehls SUB D (Bild S 87.1, Nr. 3) im Akkumulator das Byte ungleich ØØ steht.

Setzen Sie wieder bei der Anfangsadresse 183E des ALARM-Programms einen Breakpoint, so wie es im vorhergehenden Versuch beschrieben ist, und sehen Sie sich das Zero-Bit im Flag-Register nach der Ausführung des JP NZ,ALARM-Befehls an! – Es braucht Sie

dabei nur der Wert des Z-Bits zu interessieren. Am einfachsten ist es, wenn Sie nach dem Anhalten des Programms bei der Adresse 184Ø die Tasten REG und SZ.H betätigen. In der dann erscheinenden Anzeige X O X X F H ist das an der zweiten Stelle von links stehende Bit das Zero-Bit. Sein Wert Ø meldet, daß das Ergebnis der Ausführung des Subtraktions-Befehls SUB D **ungleich null** war.

Vergessen Sie bitte nicht, mit den Tasten RS und CBR den Breakpoint wieder zu löschen!

Aufgabe S89.1

Welche Änderung müssen Sie in dem im Bild S87.1 aufgelisteten Programm vornehmen, wenn Sie den im vorhergehenden Versuch eingesetzten Befehl Nr. 7, JP NZ,ALARM, durch den Programmzähler-relativen Befehl JR NZ,op für einen bedingten Sprung zum Programm-Teil ALARM ersetzen wollen?

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü13.

Sprungbefehle, deren Ausführung vom Vorzeichen-Bit abhängig ist

Das Vorzeichen-Bit S im Flag-Register meldet mit seinem Wert, ob im zuletzt mit einem logischen oder arithmetischen Verknüpfungsbefehl angesprochenen Register das ganz links stehende Bit Nr. 7 (MSB, vgl. Seite H34) den Wert Ø oder den Wert 1 hat. Der Wert 1 des Vorzeichen-Bits erlaubt es, das im Register stehende Bitmuster als negative Zahl zu interpretieren. Der Wert S = 1 des Vorzeichen-Bits S im Flag-Register kennzeichnet eine negative Zahl, der Wert S = Ø eine positive Zahl in einem Register.

Die Möglichkeit, die Ausführung eines Sprungbefehls vom Vorzeichen einer Zahl abhängig zu machen, ist natürlich vor allem im Zusammenhang mit arithmetischen Operationen von Interesse. Trotzdem wird das Vorzeichen-Bit auch von den meisten logischen Verknüpfungsbefehlen beeinflußt:

> Das Vorzeichen-Bit im Flag-Register wird – abgesehen von den Befehlen INC rp, DEC rp, ADD HL,rp und CPL – von sämtlichen logischen und arithmetischen Verknüpfungsbefehlen beeinflußt. Alle anderen Befehle lassen den Inhalt des Vorzeichen-Bits unbeeinflußt.

Wie bei den Sprungbefehlen, deren Ausführung vom Wert des Zero-Bits abhängig ist, gibt es auch zwei Sprungbefehle, deren Ausführung vom Wert des Vorzeichen-Bits abhängig ist. Programmzähler-relative Befehle in Abhängigkeit vom Wert des Vorzeichen-Bits hat jedoch auch der Mikroprozessor Z80 nicht in seinem Befehlsvorrat.

Mnemonischer Code	Operations-code	Operation	Erläuterung
JP P,adr	F2	PC←adr wenn S=Ø	Das Programm wird bei der Adresse adr fortgesetzt, wenn das Vorzeichen-Bit mit dem Wert Ø einen Register-Inhalt meldet, der größer als null oder gleich null ist. Andernfalls wird der Befehl ignoriert.
JP M,adr	FA	PC←adr wenn S=1	Das Programm wird bei der Adresse adr fortgesetzt, wenn das Vorzeichen-Bit mit dem Wert 1 einen Register-Inhalt meldet, der kleiner als null ist. Andernfalls wird der Befehl ignoriert.

Im mnemonischen Code JP P bedeutet das P Positiv. Dabei ist wieder zu beachten, daß die Zahl Null als positive Zahl betrachtet wird (vgl. Seite S83). Der Sprung wird also bei positivem Ergebnis einer Verknüpfung ausgeführt. – Das M im mnemonischen Code JP M kommt von Minus; der Sprung wird in diesem Fall bei negativem Ergebnis einer Verknüpfung ausgeführt.

Die Operanden dieser Befehle bilden eine sedezimal vierstellig dargestellte Adresse, deren LOB im Speicher bei der niedrigeren Adresse und deren HOB bei der höheren Adresse abgelegt wird.

Die Wirkung dieser Befehle können Sie sich wieder mit dem in den vorhergehenden Versuchen verwendeten Programm ansehen.

Versuch S90.1

Vom Vorzeichen bedingter Sprung

Sorgen Sie bitte dafür, daß Ihr System mit dem Programm aus den Bildern S87.1 und S87.2 geladen ist, und ersetzen Sie bei der Adresse 18ØA den Operanden des Befehls Nr. 7 durch das Byte FA für den Befehl JP M,adr:

ADDR, 1, 8, 0, A, DATA, F, A, (RS)

Starten Sie das Programm bei der Adresse 18ØØ und lassen Sie beliebige Subtraktionen mit sedezimal einstellig dargestellten Zahlen ausführen! Der Programmteil ALARM wird immer dann angesprungen, wenn eine größere Zahl von einer kleineren Zahl abgezogen wird. Wenn gleiche Zahlen voneinander subtrahiert werden, dann erkennen Sie ganz deutlich, daß die Zahl Null als positive Zahl betrachtet wird.

Sehen Sie sich das Vorzeichen-Bit S im Flag-Register nach der Ausführung von verschiedenen Subtraktionen an. Setzen Sie dazu diesmal den Breakpoint bei der Adresse 18ØA. Dieser Breakpoint läßt das Pro-

gramm grundsätzlich nach dem Einholen des Befehls JP M,ALARM anhalten.

Starten Sie das Programm bei der Adresse 18ØØ und lassen Sie zunächst eine Subtraktion mit positiver Differenz ausführen, z. B. die Subtraktion 9 − 5. Nach der Eingabe der Ziffer 5 erscheint die Anzeige 1 8 0 D 1.8. Der Befehl JP M,ALARM ist also nicht ausgeführt worden. Das war auch bei positiver Differenz nicht zu erwarten. − Betätigen Sie jetzt die Tasten

REG, SZ.H Anzeige: 0 0 0 0 F H

Das (hier stärker ausgedruckte) ganz links angezeigte Vorzeichen-Bit S (vgl. die Bilder S 82.1, S 88.1b und c) hat den Wert Ø und damit wird ein positives Ergebnis der Ausführung des Subtraktionsbefehls SUB D angemerkt. (Seite S 83.)

Wiederholen Sie den Versuch durch Betätigen der Taste RS und Start des Programms bei der Adresse 18ØØ mit einer Subtraktion, die ein negatives Rechenergebnis liefert! − Das System meldet sich mit der Adresse 183E, bei der die ALARM-Routine beginnt. Diesmal ist der Befehl JP M,ALARM ausgeführt worden. Nach Betätigung der Tasten REG und SZ.H finden Sie den Wert 1 des Vorzeichen-Bits S im Flag-Register.

Die nochmalige Wiederholung des Versuchs mit dem Subtraktions-Ergebnis null zeigt Ihnen, daß der Wert null als positive Zahl betrachtet wird (S = Ø). − Vor der Ausführung weiterer Versuche sollten Sie den Breakpoint mit der Taste CBR löschen.

Aufgabe S 91.1

Untersuchen Sie bitte, wie der vorstehend beschriebene Versuch abläuft, wenn Sie den Operationscode FA für den Befehl JP M,ALARM bei der Adresse 18ØA durch den Operationscode F2 für den Befehl JP P,ALARM ersetzen!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü 14.

Sprungbefehle, deren Ausführung vom Parity-Bit abhängig ist

Es wurde bereits erwähnt, daß das Parity-Bit in Programmen recht selten abgefragt wird (Seite S 83). Es wird hauptsächlich im Zusammenhang mit seriellem Daten-Transport verwendet. Wir können deshalb auch auf die Beschreibung eines Versuchs zu entsprechenden, bedingten Sprungbefehlen verzichten und wollen uns darauf beschränken, Ihnen diese Befehle vorzustellen.

> Das Parity-Bit im Flag-Register wird − abgesehen von den Befehlen INC rp, DEC rp, ADD HL,rp und CPL − von sämtlichen logischen und arithmetischen Verknüpfungsbefehlen beeinflußt. Alle anderen Befehle lassen den Inhalt des Parity-Bits unbeeinflußt.

Für den vom Wert des Parity-Bits abhängigen Sprungbefehl gibt es auch beim Z 80-Mikroprozessor nur absolut adressierende Varianten:

Mnemonischer Code	Operations-code	Operation	Erläuterung
JP PE,adr	EA	PC←adr wenn P=1	Das Programm wird bei der Adresse adr. fortgesetzt, wenn das Parity-Bit mit dem Wert 1 eine gerade Anzahl von 1-Bits in einem Register meldet. Andernfalls wird der Befehl ignoriert.
JP PO,adr	E2	PC←adr wenn P=Ø	Das Programm wird bei der Adresse adr. fortgesetzt, wenn das Parity-Bit mit dem Wert Ø eine ungerade Anzahl von 1-Bits in einem Register meldet. Andernfalls wird der Befehl ignoriert.

Sprungbefehle, deren Ausführung vom Übertrags-Bit abhängig ist

Das Übertrags-Bit gehört nach dem Zero-Bit zu den am häufigsten abgefragten Bits im Flag-Register. Aus diesem Grund enthält der Befehlssatz des Z 80-Mikroprozessors auch Übertrags-Bit-abhängige Zwei-Byte-Sprungbefehle, die Programmzähler-relativ adressieren.

Das Übertrags-Bit im Flag-Register wird von den Additions- und Subtraktions-Befehlen, vom Dezimal-Korrektur-Befehl, von speziellen Vergleichs-Befehlen und von den Rotations-Befehlen beeinflußt. – Die logischen Verknüpfungsbefehle löschen das Übertrags-Bit. – Alle anderen Befehle lassen das Übertrags-Bit unbeeinflußt.

Mnemonischer Code	Operations-code	Operation	Erläuterung
JP C,adr	DA	PC←adr wenn CY=1	Wenn das Übertrags-Bit den Wert 1 hat, wird das Programm bei der absolut angegebenen Adresse adr bzw. bei der um den Wert op vom aktuellen PC-Inhalt entfernten Adresse fortgesetzt. Andernfalls wird der Befehl ignoriert.
JR C,op	38	PC←(PC)+op wenn CY=1	
JP NC,adr	D2	PC←adr wenn CY=Ø	Wenn das Übertrags-Bit den Wert Ø hat, wird das Programm bei der absolut angegebenen Adresse adr bzw. bei der um den Wert op vom aktuellen PC-Inhalt entfernten Adresse fortgesetzt. Andernfalls wird der Befehl ignoriert.
JR NC,op	30	PC←(PC)+op wenn CY=Ø	

Ein Morse-Programm

Wir wollen Ihnen hier ein Programm vorstellen, in dem die im vorangegangenen Abschnitt verwendeten, bedingt auszuführenden Sprungbefehle, aber auch noch einige andere neue Befehle verwendet werden, die wir Ihnen im anschließenden Abschnitt vorstellen. Wir greifen dann jeweils auf das hier beschriebene Programm zurück.

Für den Zeitablauf von Morsezeichen gibt es sehr genaue Vereinbarungen: Ein langes Zeichen-Element soll die Länge von drei kurzen Zeichen-Elementen haben. Die Pause zwischen zwei Zeichen-Elementen soll so lang sein wie ein kurzes Zeichen-Element. Auch die Pausenlängen zwischen den einzelnen Morsezeichen und zwischen zwei Wörtern sind festgelegt. In der Praxis lassen sich diese Vereinbarungen jedoch nur ungefähr einhalten, weil Morsezeichen normalerweise von Hand getastet werden. Insbesondere das Längenverhältnis zwischen kurzen und langen Zeichen-Elementen wird individuell gehandhabt und ist oft abhängig von der Geschwindigkeit.

Die Übermittlung von Nachrichten durch Morsezeichen ist eine zwar noch oft praktizierte, grundsätzlich jedoch technisch überholte Methode. Der Vorteil ist, daß Morsezeichen auf einfachste Weise von Hand erzeugt werden können. Außerdem hat diese Art der Nachrichten-Übertragung fast etwas Sportliches an sich. Und so betrachtet ist das Programm, das wir Ihnen hier vorstellen, nahezu unanständig. Genau wie ein Programm, das Morsezeichen in normal lesbare Schrift umsetzt.

Im Bild S95.1 haben wir die wichtigsten Morsezeichen aufgelistet. Diese Morsezeichen können in gewünschter Reihenfolge im RAM-Speicher Ihres Systems abgelegt werden. Im Programm ist für jedes Zeichen ein Byte vorgesehen. In der Spalte „Bitmuster" erkennen Sie, in welcher Form die Elemente der Morsezeichen als Bitmuster dargestellt werden: Ein Null-Bit entspricht einem kurzen Zeichen-Element, ein Eins-Bit einem langen Zeichen-Element. Das Bit Nr. 7 stellt jeweils das erste Zeichen-Element dar.

Da Morsezeichen unterschiedliche Längen haben, zeigt das im Bitmuster rot eingetragene Eins-Bit das Ende des Zeichens an. Dieses Eins-Bit ist selbst kein Zeichen-Element mehr. Es verursacht eine längere Pause vor dem Aussenden des ersten Zeichen-Elements des folgenden Buchstabens. – Das grün eingetragene Bit Nr. 0 kennzeichnet, ob das jeweilige Zeichen das erste Zeichen eines neuen Worts ist. Das Programm fügt beim Wert 1 dieses Bits vor dem Aussenden des Zeichens eine längere Pause zur Kennzeichnung des Endes des vorangegangenen Wortes ein.

Im Bild S94.1 finden Sie den Ablaufplan unseres Morse-Programms, den wir hier nur kurz erläutern wollen. Einzelheiten dieses Ablaufplans und des zugehörigen Programms werden Sie verstehen, wenn Sie die anschließend vorgestellten Befehle kennengelernt haben.

Das Programm setzt voraus, daß ab der Adresse 187Ø im Speicher aufeinanderfolgend die Bitmuster für Morsezeichen abgelegt sind. Entsprechend diesen Bitmustern werden für den Lautsprecher Ihres Systems kurze und lange Morsezeichen-Elemente generiert.

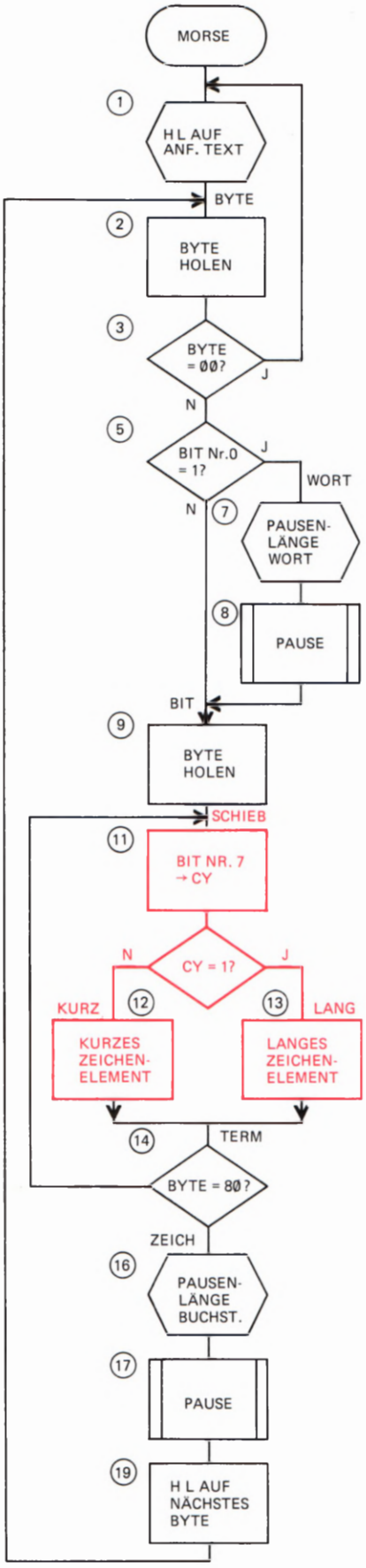


Bild S94.1
Programmablaufplan zur Generierung von Morsezeichen.

Mit der Anweisung Nr. 1 (Label MORSE) wird zunächst das HL-Registerpaar als Memory-Pointer auf das erste Morse-Byte im Speicher gesetzt. Anschließend wird das Morse-Byte, auf das der Memory-Pointer jeweils zeigt, in den Akkumulator geholt (2). Wenn es sich um das Byte ØØ handelt, das ja kein Morsezeichen darstellt, dann bedeutet das, daß der gespeicherte Text zu Ende ist. Das Programm wird mit dem Einholen des ersten Text-Zeichens fortgesetzt.

Anschließend (5) wird das (im Bild S95.1 grün markierte) Bit Nr. 0 maskiert (Seite S67). Wenn dieses Bit den Wert 1 hat, dann ergibt sich ein von ØØ unterschiedliches Byte, und das ist das Signal dafür, daß es sich um das erste Morsezeichen eines neuen Worts handelt. Es wird das Unterprogramm PAUSE aufgerufen (8). Die Länge dieser Pause zwischen zwei Wörtern wird vorher mit der Anweisung Nr. 7 eingestellt.

Weil jetzt das eingeholte Morse-Byte kaputt-geprüft worden ist, wird das Byte noch einmal neu aus dem Speicher geholt (9).

Den Kern des Programms enthält die Anweisung Nr. 11: Entsprechend der Darstellung im Bild S95.2 wird das Bit Nr. 7 mit einem später vorgestellten Schiebe-Befehl in das Carry-Bit des Flag-Registers gebracht (vgl. Bild S82.1). Dieses Bit repräsentiert jeweils – je nach seinem Wert – ein kurzes oder langes Morsezeichen-Element. Der anschließende Befehl JR NC,KURZ (12, vgl. Seite S92) verursacht bei einem Ø-Bit einen Sprung zum Programm-Teil KURZ; der Befehl JR C,LANG läßt das Programm zum Programm-Teil LANG springen. Diese beiden Programm-Teile haben wir als „Rucksäcke" (Seite S70) ausgelegt, damit das Hauptprogramm übersichtlich bleibt. Am Ende der beiden Rucksäcke – deren Funktion hier nicht weiter interessiert – steht jeweils der Befehl für einen unbedingten Sprung zum Label TERM (TERMinator = Ende-Zeichen).

Ab dem Label SCHIEB läuft das Programm in einer Schleife. Beim Label TERM wird jedesmal nach dem Schieben eines Bits in das Carry-Bit geprüft, ob das Rest-Byte im Akkumulator den Wert 8Ø mit dem Bitmuster 1ØØØ ØØØØ hat, ob also das im Bild S95.1 jeweils rot dargestellte 1-Bit bereits an der Bitstelle Nr. 7 angekommen ist. Das Bild S95.2 macht diesen Vorgang deutlich. Wenn an dieser Stelle des Programms im Akkumulator das Byte 8Ø steht, dann ist das ein Zeichen dafür, daß alle Bits des Bytes, die Morsezeichen-Elemente darstellen, von den Programm-Teilen KURZ oder LANG verarbeitet wurden. Das Programm arbeitet dann beim Label ZEICH weiter. Andernfalls wird die Schleife ab dem Label SCHIEB ein weiteres Mal durchlaufen.

Beim Label ZEICH ist die Ausgabe der Elemente des Morsezeichens für einen Buchstaben (Ziffer, Satzzeichen) beendet. Mit der Anweisung Nr. 16 wird die Länge der anschließend vom Unterprogramm PAUSE (17) generierten Pause nach einem Buchstaben programmiert.

Jetzt kann das nächstfolgende Morsezeichen ausgegeben werden. Vorbereitend wird dazu der Inhalt des HL-Registerpaares als Memory Pointer inkrementiert (18, Seite S53), so daß es auf die Speicherzelle mit dem nächstfolgenden Morse-Byte zeigt. Am Ende des Programms steht der Befehl für einen unbedingten Sprung zum Label Byte, bei dem ein Zeichen aus dem Speicher geholt wird.

Buchst.	Zeichen	Bitmuster	Byte	Neues Wort	Buchst.	Zeichen	Bitmuster	Byte	Neues Wort
A	.–	0110 0000	60	61	W	.––	0111 0000	70	71
B	–...	1000 1000	88	89	X	–..–	1001 1000	98	99
C	–.–.	1010 1000	A8	A9	Y	–.––	1011 1000	B8	B9
D	–..	1001 0000	90	91	Z	––..	1100 1000	C8	C9
E	.	0100 0000	40	41	0	–––––	1111 1100	FC	FD
F	..–.	0010 1000	28	29	1	.–––––	0111 1100	7C	7D
G	––.	1101 0000	D0	D1	2	..–––––	0011 1100	3C	3D
H	....	0000 1000	08	09	3	...–––	0001 1100	1C	1D
I	..	0010 0000	20	21	4	....–	0000 1100	0C	0D
J	.–––	0111 1000	78	79	5	.....	0000 0100	04	05
K	–.–	1011 0000	B0	B1	6	–....	1000 0100	84	85
L	.–..	0100 1000	48	49	7	––...	1100 0100	C4	C5
M	––	1110 0000	E0	E1	8	–––..	1110 0100	E4	E5
N	–.	1010 0000	A0	A1	9	––––.	1111 0100	F4	F5
O	–––	1111 0000	F0	F1	.	.–.–.–	0101 0110	56	57
P	.––.	0110 1000	68	69	,	––..––	1100 1110	CE	CF
Q	––.–	1101 1000	D8	D9	:	–––...	1110 0010	E2	E3
R	.–.	0101 0000	50	51	?	..––..	0011 0010	32	33
S	...	0001 0000	10	11	–	–....–	1000 0110	86	87
T	–	1100 0000	C0	C1	/	–..–.	1001 0100	94	95
U	..–	0011 0000	30	31	+	.–.–.	0101 0100	54	55
V	...–	0001 1000	18	19					

Bild S95.1
Diese Tabelle der wichtigsten Morsezeichen zeigt die Bitmuster und Bytes, aus denen unser System die Zeichen generiert.

Versuch S95.1

Generierung von Morsezeichen

Tasten Sie die Bytes des im Bild S96.1 aufgelisteten Programms in Ihr System ein und anschließend ab der Adresse 1826 bis einschließlich der Adresse 1868 die Bytes, die im Bild S96.2 als Hex-Dump (vgl. Seite S86) angegeben sind. (Ohne diese zusätzlichen Bytes ist das Programm nicht lauffähig.)

Damit das Morsezeichen-Programm Zeichen zum Ausgeben vorfindet, müssen ab der Adresse 1870 Morse-Bytes abgelegt werden. Folgend auf das letzte Morse-Byte muß ein Byte 00 eingetragen sein. Dadurch wird verhindert, daß die folgenden, zufällig im Speicher stehenden Bytes als (sinnlose) Morsezeichen interpretiert werden. Wir haben beschrieben, daß das Byte 00 die immer neue Ausgabe des gleichen Textes verursacht (Bild S94.1, Anweisung Nr. 3).

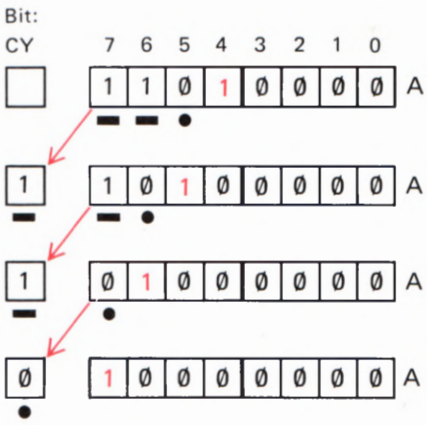


Bild S95.2
So erscheinen die Morsezeichen-Elemente des Buchstabens G nacheinander als CY-Bit. Der rot eingetragene Terminator 1 ist kein Morsezeichen-Element.

Aufgabe S95.1

Überlegen Sie, welche Änderung Sie im Programm vornehmen müssen, wenn der Morse-Text nur ein einziges Mal ausgegeben werden soll! Nach dieser einmaligen Ausgabe des Textes soll das System sich so verhalten, als würde die RS-Taste (Reset) betätigt.

Lesen Sie nocheinmal bei der Aufgabe S44.1 nach!

Die Lösung dieser Aufgabe finden Sie auf der Seite Ü14.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	MORSE	LD	HL,TEXT	18ØØ	21	7Ø	18	Zeiger → Text Anfang
2	BYTE	LD	A,(HL)	18Ø3	7E			Byte holen
3		CP	ØØ	18Ø4	FE	ØØ		Byte = ØØ für Ende?
4		JR	Z,MORSE	18Ø6	28	F8		Ja: Text von vorn
5		AND	Ø1	18Ø8	E6	Ø1		Bit Nr. 0 = 1?
6		JR	Z,BIT	18ØA	28	Ø5		
7	WORT	LD	B,Ø4	18ØC	Ø6	Ø4		Ja: Wort-Ende
8		CALL	PAUSE	18ØE	CD	3F	18	Pause
9	BIT	LD	A,(HL)	1811	7E			Byte holen
10		AND	FE	1812	E6	FE		Bit Nr. 0 wird Ø
11	SCHIEB	SLA	A	1814	CB	27		Bit Nr. 7 → CY
12		JR	NC,KURZ	1816	3Ø	ØE		CY = Ø: Kurz
13		JR	C,LANG	1818	38	1Ø		CY = 1: Lang
14	TERM	CP	8Ø	181A	FE	8Ø		Zeichen-Ende?
15		JR	NZ,SCHIEB	181C	2Ø	F6		Nein: Nächstes Bit
16	ZEICH	LD	B,Ø2	181E	Ø6	Ø2		Ja: Zeichen-Ende
17		CALL	PAUSE	182Ø	CD	3F	18	Pause
18		INC	HL	1823	23			Nächstes Zeichen-Byte
19		JR	BYTE	1824	18	DD		Schleife

Bild S96.1
Liste des Hauptprogramms zur Generierung von Morsezeichen. Zu diesem Programm gehören noch die Befehle, die im Bild S96.2 als Hex-Dump angegeben sind.

```
1826                              06 01
1828 18 02 06 03 E5 21 6A 18
1830 36 FF CD 4A 18 06 01 36
1838 7F CD 4A 18 E1 18 DB E5
1840 21 6A 18 36 7F CD 4A 18
1848 E1 C9 F5 2B 4E 23 7E D3
1850 02 CD 62 18 3E 7F D3 02
1858 CD 62 18 0D 20 F0 10 EB
1860 F1 C9 C5 06 80 10 FE C1
1868 C9

Geschwindigkeit:
1869    40
Text:
1870 C1 40 A8 08 A0 20 10 A8
1878 08 40 10 49 40 08 50 20
1880 A0 10 C0 20 C0 30 C0 91
1888 50 56 A9 08 50 20 10 C0
1890 20 60 A0 20 B1 F0 A0 10
1898 C0 60 A0 C8 95 89 F0 90
18A0 40 A0 10 40 40 55 90 78
18A8 7C 18 A8 55 00
```

Bild S96.2
Der Hex-Dump ab der Adresse 1826 enthält die „Rucksäcke" KURZ und LANG und das Unterprogramm PAUSE. Ab der Adresse 1870 ist ein Beispiel-Text aufgelistet.

Bei der Eingabe eines eigenen Textes müssen Sie außerdem beachten, daß jeweils das erste Zeichen eines neuen Wortes mit dem Morse-Byte aus der Spalte „Neues Wort" programmiert wird. – Im Bild S96.2 haben wir ab der Adresse 187Ø einen Beispiel-Text aufgelistet, den Sie zunächst der Einfachheit halber verwenden können.

Die Geschwindigkeit, mit der die Morsezeichen ausgegeben werden, wird durch das bei der Adresse 1869 abgelegte Byte bestimmt. Im Bild S96.2 haben wir für diese Adresse das Byte 4Ø eingetragen, das Sie zunächst für eine mittlere Morse-Geschwindigkeit einsetzen können.

Starten Sie jetzt das Programm bei der Adresse 18ØØ! Sie können versuchen, unseren Text zu entziffern. Gegebenenfalls können Sie versuchsweise bei der Adresse 1869 ein Byte mit einem größeren Wert (z. B. 6Ø) eintasten. – Wenn es Ihnen gelingt, auch noch bei Wert 2Ø oder bei einem noch kleineren Wert des Geschwindigkeits-Bytes den Text zu entziffern, dann müssen Sie schon ein Könner sein.

Wichtig ist es keineswegs, daß Sie unseren Morse-Text entziffern. (Notfalls können Sie unsere Bytes mit Hilfe der Tabelle im Bild S95.1 decodieren.) Wichtig ist allein, daß Sie die vorher beschriebene Arbeitsweise des Programms verstehen.

Der Ausgabe-Charakter des Morse-Textes läßt sich durch Verändern bestimmter Bytes im Programm beeinflussen:

Eigenschaft	Adr.	Byte	Bemerkungen
Geschwindigkeit	1869	1Ø – 7Ø	
Langes Zeichen-Element	182B	Ø2 – Ø4	Ø3 ist Norm
Zeichen-Pause	181F	Ø2 – Ø3	
Wort-Pause	18ØD	Ø4 – Ø6	

Der Befehlssatz unseres Mikroprozessors

(Fortsetzung)

Im Laufe unseres Lehrgangs haben wir recht häufig von **Unterprogrammen** Gebrauch gemacht, deren Technik wir Ihnen bereits im ersten Lehrbrief vorgestellt haben (Seite S5). Deren Handhabung wollen wir Ihnen hier zeigen und auch gleich noch die oft benutzten **Vergleichsbefehle** vorstellen.

Unterprogramm-Befehle

Unterprogramme werden mit dem Befehl CALL adr aufgerufen (Seite H32), und dieser CALL-Befehl hat grundsätzlich große Ähnlichkeit mit einem absolut adressierenden Sprungbefehl: Er bewirkt, daß das Programm bei der in den Operanden dieses Befehls angegebenen Adresse adr fortgesetzt wird. So gesehen besteht eigentlich überhaupt kein Unterschied zwischen einem JP- und einem CALL-Befehl. Der entscheidende Witz bei der Verwendung von Unterprogrammen ist jedoch, daß eine solche Routine (Seite S6) innerhalb eines Programms beliebig oft aufgerufen werden kann. Der Ansprung ein und desselben Unterprogramms kann also bei verschiedenen Adressen programmiert werden. Jedesmal werden dann die Befehle dieses Unterprogramms abgearbeitet und anschließend wird das Programm jeweils an der auf den CALL-Befehl folgenden Adresse fortgesetzt. Der CALL-Befehl mit seinen beiden Operanden steht also sozusagen stellvertretend für die beliebig vielen Befehle des Unterprogramms.

Das Bild S97.1 macht dieses Prinzip deutlich. Der erste Aufruf des Unterprogramms UPGRM lädt den Programmzähler mit der Anfangsadresse adr des Unterprogramms (vgl. ab Seite S71), so daß nach dem CALL-Befehl als nächster der erste Befehl des Unterprogramms und anschließend alle weiteren Befehle des Unterprogramms ausgeführt werden. Nach vollständiger Ausführung aller Befehle des Unterprogramms soll der auf den CALL-Befehl folgende Befehl NEXT1 (was immer das auch für ein Befehl sein mag) ausgeführt werden. Dazu muß am Ende des Unterprogramms ein Befehl stehen, der den Programmzähler mit der Adresse des Befehls NEXT1 lädt.

Die gleiche Überlegung gilt für den zweiten Aufruf des gleichen Unterprogramms. Nach dem Abarbeiten des Unterprogramms muß der letzte Befehl des Unterprogramms den Programmzähler mit der Adresse des hier auf den CALL-Befehl folgenden Befehls NEXT2 laden.

Die hier beschriebenen Anforderungen an die Möglichkeit, ein und dasselbe Unterprogramm wiederholt aufrufen zu können, scheint unmöglich erfüllbar zu sein: Wie soll das Unterprogramm wissen, vom wievielten CALL-Befehl innerhalb des Hauptprogramms es gerade aufgerufen wurde? Nur wenn es das wüßte, könnte es an seinem Ende den Programmzähler jeweils mit dem Wert laden, der die Fortsetzung des Hauptprogramms nach dem richtigen CALL-Befehl sicherstellt.

Offenbar funktioniert das aber bei unserem Mikroprozessor doch. Wir wollen die Erläuterung dieses Mechanismus' noch einen Augenblick zurückstellen und es zunächst bei der Feststellung bewenden lassen, daß hier der Unterschied zwischen den Call-Befehlen und den norma-

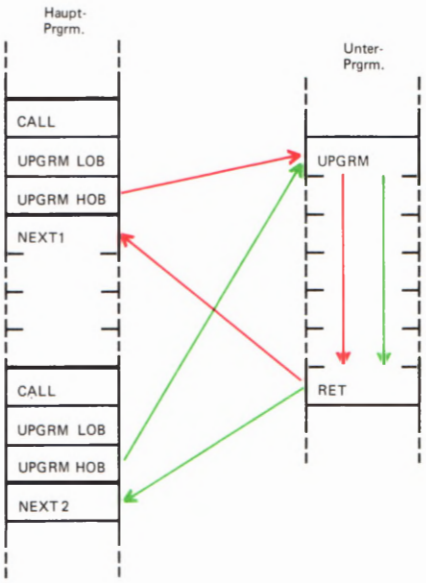


Bild S97.1
Nach dem Abarbeiten eines Unterprogramms muß das Hauptprogramm nach dem aufrufenden CALL-Befehl fortgesetzt werden.

len Sprung-Befehlen liegt. Dagegen besteht eine andere Gemeinsamkeit der CALL-Befehle und der JP-Befehle: Wie bei den JP-Befehlen gibt es auch bei den CALL-Befehlen die unbedingte und die von den Werten der Bits im Flag-Register bedingte Ausführung:

Mnemonischer Code	Operationscode	Operation	Erläuterung
CALL adr	CD	(PC) ← Stack PC ← adr	Das Programm wird bei der Anfangs-Adresse adr eines Unterprogramms fortgesetzt. Die auf den CALL-Befehl folgende Adresse wird im Stack abgelegt.
CALL c,adr	Tabelle	(PC) ← Stack PC ← adr wenn c erfüllt	Das Programm wird bei der Anfangsadresse adr eines Unterprogramms fortgesetzt, wenn die Bedingung c erfüllt ist. Die auf den CALL-Befehl folgende Adresse wird im Stack abgelegt.

Was es mit dem hier genannten Stack auf sich hat, werden wir in einem der folgenden Abschnitte zeigen. Hier zeigen wir Ihnen zunächst die Operationscodes der bedingten Unterprogramm-Aufrufe:

Flag	Z		CY		P/V		S	
c	Z	NZ	C	NC	PE	PO	M	P
CALL	CC	C4	DC	D4	EC	E4	FC	F4

Die Bedeutungen der Bedingungen c (Z, NZ, C usw.) brauchen wir nicht noch einmal zu erläutern; es sind die gleichen, die Sie bei den Befehlen für bedingte Sprünge kennengelernt haben.

Versuch S 98.1

Bedingte Unterprogramm-Aufrufe

Tasten Sie in Ihr System bitte die Bytes der Befehlsfolge im Bild S 99.1 ein und außerdem die Bytes aus dem Hex-Dump im Bild S 99.2!

Starten Sie das Programm bei der Adresse 18ØØ! In der Anzeige erscheinen wahrscheinlich undefinierte Zeichen, die Sie in diesem Versuch nicht zu interessieren brauchen. Betätigen Sie die Taste 0 – und anschließend die Taste 1. Betätigen Sie (außer den Tasten RS und MONI) beliebige andere Tasten! – Sie hören selbst, was passiert.

Interessanter ist die Befehlsfolge, die diese Wirkung hervorbringt. Mit der Anweisung Nr. 1 im Bild S 99.1 rufen wir eine Routine des Betriebsprogramms auf, die das Tastenfeld abfragt und bei Betätigung einer Zifferntaste das dem Aufdruck entsprechende, sedezimale Byte in den

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	TASTE	CALL	SCAN	18ØØ	CD	FE	Ø5	Tasten abfragen
2		CP	ØØ	18Ø3	FE	ØØ		Taste 0?
3		CALL	Z,SIRENE	18Ø5	CC	1Ø	18	Ja: Sirene
4		CP	Ø1	18Ø8	FE	Ø1		Taste 1?
5		CALL	Z,ALARM	18ØA	CC	29	18	Ja: Alarm
6		JR	TASTE	18ØD	18	F1		Nächste Taste

Akkumulator bringt. Den zur nächsten Anweisung gehörenden Befehl CP stellen wir Ihnen im folgenden Abschnitt vor. Er prüft, ob im Akkumulator das Byte ØØ steht. Ist das der Fall, dann erhält das Zero-Bit Z im Flag-Register (Bild S 82.1) den Wert 1. Steht im Akkumulator dagegen ein von ØØ abweichendes Byte, dann wird das Zero-Bit auf den Wert Ø gesetzt.

Die Wirkung des Befehls CALL Z,SIRENE (3) ist jetzt leicht zu durchschauen. Dieser Befehl ruft das Unterprogramm SIRENE nur dann auf, wenn das Zero-Bit Z im Flag-Register den Wert 1 hat, also dann, wenn die Taste 0 betätigt wurde. Bei Betätigung irgendeiner anderen Taste liefert der Befehl CP ØØ den Wert Ø an das Zero-Bit im Flag-Register und das Unterprogramm SIRENE wird nicht aufgerufen.

Ganz entsprechend arbeiten die folgenden Anweisungen Nr. 4 und Nr. 5: Der Befehl CP Ø1 prüft, ob im Akkumulator durch Betätigung der Taste 1 das Byte Ø1 steht. Ist das der Fall, dann nimmt das Zero-Bit im Flag-Register den Wert 1 an und der nachfolgende Aufruf des Unterprogramms ALARM wird ausgeführt. Nach Betätigung irgendeiner anderen Taste ist die Bedingung für den Aufruf des Unterprogramms ALARM nicht erfüllt.

Eins der beiden Unterprogramme wird nur dann angesprungen, wenn eine der Tasten 0 und 1 betätigt wurde. Wird eine beliebige andere Taste betätigt, dann ergeben die Prüfungen mit den beiden CP-Befehlen negative Ergebnisse (Z = Ø im Flag-Register) und der Befehl JR TASTE schickt das Programm wieder zur Tasten-Abfrage-Anweisung.

Bild S 99.1
Zu diesem Hauptprogramm des Versuchs S 98.1 gehören noch die Unterprogramme SIRENE und ALARM im Bild S 99.2.


```
SIRENE
1810 06 02 0E 00 21 C0 00 C5
1818 CD E4 05 C1 0E C0 21 00
1820 01 C5 CD E4 05 C1 10 EA
1828 C9

ALARM
1829    06 20 21 18 00 C5 CD
1830 DE 05 C1 21 10 00 C5 CD
1838 E2 05 C1 10 EE C9
```

Bild S 99.2
Die Unterprogramme SIRENE und ALARM werden von dem im Bild S 99.1 aufgelisteten Hauptprogramm aufgerufen.

Vergleichs-Operationen

Im vorangegangenen Versuch haben Sie eine Anweisung zum Vergleich zweier Bytes kennengelernt, in der offenbar nicht der auf der Seite S 58 für solche Vergleiche angepriesene Exklusiv-ODER-Befehl verwendet wurde. Selbstverständlich sind unsere dort getroffenen Feststellungen richtig: Im Programm im Bild S 99.1 können statt der Befehle CP ØØ auch die Befehle XOR ØØ verwendet werden. (Sie können das leicht nachprüfen, wenn Sie im Programm zum Versuch S 98.1 bei den Adressen 18Ø3 und 18Ø8 jeweils den Operationscode EE des Befehls XOR Konst einsetzen.) – Tatsächlich sind Mikroprozessoren gebaut worden, in deren Befehlssatz der Exklusiv-ODER-Befehl die einzige Möglichkeit zum Vergleich zweier Bytes bot.

Vornehmere Mikroprozessoren stellen im Befehlssatz spezielle Vergleichs-Befehle zur Verfügung. Diese Befehle beruhen auf einer vom internen Mikroprogramm der CPU vorgenommenen Subtraktion der beiden miteinander zu vergleichenden Bytes. Der Vorteil dieser Methode besteht darin, daß eine Subtraktion nicht nur das Byte ØØ

liefert und damit das Zero-Bit im Flag-Register auf den Wert 1 setzt, wenn die beiden miteinander zu vergleichenden Bytes gleich sind. Sie liefern darüber hinaus auch ein negatives Subtraktions-Ergebnis, wenn das zweite Byte größer ist als das erste und setzen dann das Vorzeichen-Bit S (Seite S 90) auf den Wert 1. Mit den Befehlen JP M und JP P (Seite S 82) lassen sich dadurch bedingte Sprungbefehle in Abhängigkeit davon ausführen, ob das eine der beiden Bytes größer oder kleiner ist als das andere. Entsprechend können auch unterschiedliche Unterprogramme aufgerufen werden, je nachdem, ob eins der miteinander zu vergleichenden Bytes größer oder kleiner ist als das andere, oder ob beide Bytes die gleichen Werte haben.

Der ganz besondere Vorteil der speziellen Vergleichsbefehle besteht darin, daß nach ihrer Ausführung der Inhalt des Akkumulators unverändert bleibt. Sie erinnern sich: Nach der Ausführung eines Exklusiv-ODER-Befehls steht im Akkumulator das Verknüpfungsergebnis; der ursprüngliche Akkumulator-Inhalt wird überschrieben. – Im Programm im Bild S 99.1 erkennen Sie, daß mit den CP-Befehlen das von der SCAN-Routine in den Akkumulator geholte Tasten-Byte zwei Vergleichs-Operationen nacheinander unterworfen wird. Auch bei der zweiten Vergleichs-Operation ist das Tasten-Byte im Akkumulator noch unverändert.

Wir wollen Ihnen hier die wichtigsten Vergleichs-Befehle des in Ihrem System verwendeten Mikroprozessors vorstellen:

Mnemonischer Code	Operation	Erläuterung
CP r	(A) – (r);S,Z H,P/V,N,C	Der Inhalt des Akkumulators wird mit dem Inhalt des Registers r oder mit dem Inhalt der vom HL-Registerpaar adressierten Speicherzelle verglichen. (A) bleibt unverändert. Das Vergleichs-Ergebnis ist aus dem Inhalt des Flag-Registers abzulesen.
CP (HL)	(A) – (HL);S, Z,H,P/V,N,C	
CP Konst	(A) – Konst;S, Z,H,P/V,N,C	Der Inhalt des Akkumulators wird mit dem konstanten Wert Konst verglichen. (A) bleibt unverändert. Das Vergleichs-Ergebnis ist aus dem Flag-Register abzulesen.

Die Operationscodes dieser Befehle können Sie aus folgender Tabelle ablesen:

	A	B	C	D	E	H	L	Speicherzelle	Konstante
CP	BF	B8	B9	BA	BB	BC	BD	BE	FE

In unserer Aufstellung bedeuten die Buchstaben S, Z, H, P/V, N und C, daß die entsprechenden Bits im Flag-Register (Bild S 82.1) bei der Ausführung dieser Befehle beeinflußt werden. Im Zusammenhang mit unseren Überlegungen interessieren dabei nur das Zero-Bit Z und das Vorzeichen-Bit S.

Im Morse-Programm haben wir von Vergleichs-Befehlen bei den Anweisungen Nr. 3 und Nr. 14 Gebrauch gemacht (Bilder S 94.1 und S 96.1).

Rotations- und Schiebe-Befehle

Im Zusammenhang mit den UND-Verknüpfungs-Befehlen haben wir bereits darauf hingewiesen, daß die Verwendung der Maschinensprache dem Programmierer die Möglichkeit bietet, sich liebevoll mit jedem einzelnen Bit eines Bytes zu befassen (Seite S61). Morse-Programme z.B., wie wir Ihnen eins ab der Seite S93 vorgestellt haben, können effektiv nur programmiert werden, wenn die Möglichkeit der Behandlung einzelner Bits eines Bitmusters besteht. Sie erkennen das bei der Betrachtung des Bildes S95.2.

Bei der in diesem Bild dargestellten Behandlung eines Bitmusters haben wir einen Befehl verwendet, der zu einer ganzen Reihe von Befehlen gehört, mit denen ein Bitmuster durch Verschieben in einem Register oder in einer Speicherzelle beeinflußt werden kann. Wir wollen Ihnen zum Abschluß unseres Lehrgangs einige dieser Befehle vorstellen.

Von den im Befehlssatz des Mikroprozessors Z80 verfügbaren **Schiebe-Befehlen** interessieren uns die beiden im Bild S101.2 aufgelisteten Befehle SLA und SRL. Bei der Ausführung eines dieser Befehle wird das Bitmuster im angesprochenen Register oder in der vom HL-Registerpaar adressierten Speicherzelle um eine Bit-Stelle nach links (SLA) bzw. um eine Bit-Stelle nach rechts (SRL) geschoben. Das Bild S101.1 macht deutlich, wie der Befehl SLA A aus dem Byte D6 im Akkumulator das Byte AC macht. Bei den hier vorgestellten Befehlen ist es wesentlich, daß jeweils das aus dem Register (oder aus der Speicherzelle) hinausgeschobene Bit in das CY-Bit des Flag-Registers hineingeschoben wird. Die durch das Schieben frei gewordene Bit-Stelle wird durch ein Ø-Bit aufgefüllt.

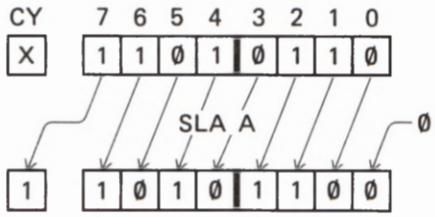


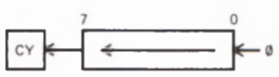
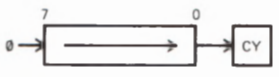
Bild S101.1
Durch die Funktion des hier dargestellten Schiebe-Befehls SLA A wird das Byte D6 im Akkumulator in das Byte AC verwandelt. Das Bit Nr.7 wird an die CY-Bit-Stelle geschoben.

Den Befehl SLA A haben wir als Anweisung Nr.11 im Morse-Programm verwendet (Bild S96.1). Wenn Sie die Bilder S95.2 und S101.1 miteinander vergleichen, dann erkennen Sie, wie im Morse-Programm ein Bit nach dem anderen als kurzes oder langes Morsezeichen-Element ausgewertet wird.

Die hier vorgestellten Schiebe-Befehle sind Zwei-Byte-Befehle und nur im Befehlssatz des Z80-Mikroprozessors verfügbar. Das zweite Byte ist ein Operationscode, mit dem das Register oder die vom HL-Registerpaar adressierte Speicherzelle zum Schieben des Bitmusters bestimmt wird.

Bild S 101.2
Mit den Zwei-Byte-Schiebe-Befehlen des Z80 wird ein Bitmuster durch das CY-Bit ausgeschoben.

Bei mehrmaliger Anwendung der soeben vorgestellten Schiebe-Befehle, wird ein Bitmuster nach und nach zerstört: Je nach Befehl

Mnem. Code	Opcode	A	B	C	D	E	H	L	(HL)	Funktion
					Opcode					
SLA	CB	27	20	21	22	23	24	25	26	
SRL	CB	3F	38	39	3A	3B	3C	3D	3E	

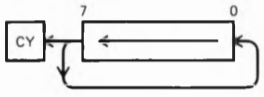
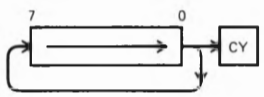
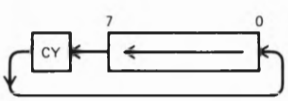
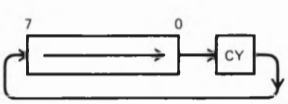
wird das Bit Nr. 7 oder das Bit Nr. 0 aus dem betreffenden Register hinausgeschoben; das Register wird mit Werten Ø aufgefüllt.

Neben diesen Schiebe-Befehlen gibt es noch zwei weitere Befehlsgruppen, mit denen ein Bitmuster auf ähnliche Weise beeinflußt werden kann. Die geschobenen Bits gehen dabei jedoch nicht verloren; sie werden innerhalb des Registers rotiert, d. h. letztlich kehren die hinausgeschobenen Bits immer wieder in das gleiche Register zurück. (Der Z80-Mikroprozessor verfügt noch über eine vierte Schiebe-Befehlsgruppe, mit der wir uns hier aber nicht beschäftigen können.)

Im Bild S102.1 haben wir die hier interessierenden **Rotations-Befehle** zusammengestellt. Bei den Befehlen RLCA und RLC r bzw. RLC (HL) sowie bei den Befehlen RRCA und RRC r bzw. RRC (HL) wird das aus dem Register bzw. aus der vom HL-Registerpaar adressierten Speicherzelle an der einen Seite hinausgeschobene Bit an der anderen Seite wieder hineingeschoben. Gleichzeitig wird dieses Bit an die CY-Bit-Stelle des Flag-Registers kopiert. Dabei bewirken die Befehle RLCA und RLA das Links-Rotieren des Bitmusters; die Befehle RRCA und RRC bewirken das Rechts-Rotieren. Die Ein-Byte-Befehle RLCA und RRCA wirken nur auf das Bitmuster im Akkumulator. Die universelleren Zwei-Byte-Befehle RLC und RRC sind nur im Befehlssatz des Z80-Mikroprozessors verfügbar.

Genau entsprechende Überlegungen gelten für die Befehle RLA und RL r bzw. RL (HL), die ein Links-Rotieren bewirken, und für die Befehle RRA und RR r bzw. RR (HL), die ein Rechts-Rotieren bewirken. Den Unterschied zu den vorhergenannten beiden Rotations-Befehlen zeigt das Bild S102.1 in der Spalte Funktion: Das hinausgeschobene Bit wird nicht mit dem gleichen Befehl wieder eingeschoben, sondern durch das Flag-Register transportiert und erst bei der nächsten Anwendung des Befehls wieder eingeschoben.

Bild S 102.1
Die Rotations-Befehle lassen kein Bit verloren gehen. Das ausgeschobene Bit wird direkt oder auf dem Wege über die CY-Bit-Stelle wieder eingeschoben. Das ausgeschobene Bit erscheint immer an der CY-Bit-Stelle.

Mnem. Code	Opcode	A	B	C	D	E	H	L	(HL)	Funktion
					Opcode					
RLCA	--	Ø7	--	--	--	--	--	--	--	
RLC	CB	Ø7	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	
RRCA	--	ØF	--	--	--	--	--	--	--	
RRC	CB	ØF	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	
RLA	--	17	--	--	--	--	--	--	--	
RL	CB	17	1Ø	11	12	13	14	15	16	
RRA	--	1F	--	--	--	--	--	--	--	
RR	CB	1F	18	19	1A	1B	1C	1D	1E	

Prüfungsaufgaben

Was ist beim Anfertigen und Einsenden der Lösungen der Prüfungsaufgaben zu beachten?

Bitte senden Sie die Lösungen der Prüfungsaufgaben nur dann an das Lehrinstitut ein, wenn Sie den Lehrgang „Mikroprozessor-Technik" als Fernunterrichtswerk erworben haben. Wenn Ihnen dieser Lehrgang als Selbstunterrichtswerk zur Verfügung steht, dann können Sie die Prüfungsaufgaben an das Lehrinstitut zur Korrektur einsenden, wenn Sie zusätzlich ein entsprechendes Betreuungspaket erwerben.

Zur Einsendung sind nur die Lösungen der Prüfungsaufgaben bestimmt, die Sie auf den Seiten mit der Griffmarke P – in diesem Lehrbrief ab der Seite P 3 – vorfinden. Die in den Abschnitten gestellten Aufgaben, zu denen wir Ihnen jeweils im selben Lehrbrief auf den Ü-Seiten die Lösungen angeben, dienen nur Ihrer Selbstkontrolle. Ihre Lösungen dieser Übungsaufgaben sollen Sie nicht an uns einschicken.

Beschreiben Sie bei Ihren Aufgaben-Lösungen bitte grundsätzlich jedes Blatt nur auf einer Seite! Der Papierverbrauch wird dadurch zwar größer, aber Sie erleichtern damit dem Korrektor die Arbeit.

Ihre Studiennummer und Ihr Name sollen auf jedes Blatt geschrieben werden, damit es keine Verzögerungen oder Falschsendungen gibt. Ihre Anschrift braucht nur auf dem ersten Lösungsblatt zu stehen.

Einen Gutschein für weitere Aufgabenlösungsblätter erhalten Sie mit den ersten Lösungsblättern. Füllen Sie den Gutschein bitte aus und senden Sie ihn mit den ersten Aufgabenlösungen an uns ein. Sie erhalten die bestellten Blätter kostenlos und portofrei. Der Sendung liegt wieder ein Gutschein für Ihre nächste Bestellung bei, so daß Sie keine Lösungsblätter hinzukaufen müssen.

Ferner legen wir Briefumschläge verschiedener Größe bei. Verwenden Sie zum Einsenden Ihrer Prüfungsarbeiten an uns einen dieser bereits mit unserer Anschrift versehenen Briefumschläge.

Die großen Umschläge werden auf dem Postweg leicht beschädigt, wenn sich nur wenige Blätter darin befinden. Wenige Blätter falten Sie besser und stecken sie in einen kleineren Umschlag.

Legen Sie Ihrer Sendung bitte einen gleichgroßen Umschlag bei und vergessen Sie nicht, auf diesen Ihre Anschrift zu schreiben. In diesem Umschlag erhalten Sie Ihre Arbeiten zurück. Wenn Sie mit Ihren Arbeiten einen Gutschein zum Bezug von Lösungsblättern einsenden, dann legen Sie bitte auf jeden Fall den mit Ihrer Anschrift versehenen großen Umschlag bei.

Die für Ihre erste Korrektursendung nicht benötigten Briefumschläge bewahren Sie bitte auf, falls Sie zum Einsenden der Aufgabenlösungen späterer Lehrbriefe andersformatige Umschläge benötigen.

Wir legen Ihnen bei jeder Rücksendung wieder einen gleichgroßen, an uns adressierten Umschlag und einen weiteren bei, auf den Sie bitte wieder Ihre Anschrift schreiben. Auf diese Weise haben Sie bei jeder

Sendung an uns die Möglichkeit, das Umschlagformat der jeweiligen Anzahl der Blätter anzupassen.

Frankieren Sie bitte den an uns gerichteten Brief. Das Porto für die Rücksendung tragen wir. Sie brauchen also den Umschlag, den Sie – mit Ihrer Adresse versehen – Ihrer Sendung an uns beilegen, nicht zu frankieren.

Sie können die Lösung mehrerer Lehrbriefe zusammen an uns einsenden. Es ist jedoch zweckmäßig, wenn wir Sie recht bald auf Denkfehler aufmerksam machen können. Wir empfehlen Ihnen deshalb, jeweils nur die Lösungen zu einem Lehrbrief an uns einzusenden.

Die Lösungen sind stets in der Reihenfolge zu bringen, in der die Aufgaben im Lehrbrief stehen. Bei Rechenaufgaben soll der ganze Rechnungsgang gebracht und das Endergebnis unterstrichen werden. Benutzen Sie dabei aber bitte keinen Rotstift, weil Rot die Korrekturfarbe ist. Lassen Sie bitte zwischen den einzelnen Lösungen mindestens 1 cm Platz frei.

Achten Sie bitte auf die vollständige Lösung und Einsendung aller Aufgaben! Lückenhafte Arbeiten müssen unkorrigiert zurückgesandt werden.

Sie können nur dann ein Abschlußzeugnis über die erfolgreiche Teilnahme am Lehrgang erhalten, wenn sämtliche Aufgabenlösungen zur Korrektur vorgelegt wurden.

Die einzelnen Blätter sind mit Hilfe von Briefklammern zusammenzuheften, wobei – bei der Einsendung mehrerer Lösungen – die Lösungen zur niedrigsten Lehrbriefnummer oben liegen sollen. Wenn Sie die Lösungen mehrerer Lehrbriefe zugleich einsenden, dann stecken Sie bitte an das oberste Blatt der Sendung einen kleinen Zettel, auf den Sie z. B. schreiben: „Lösungen zu den Lehrbriefen 2 bis 4".

Im übrigen vermeiden Sie bitte, kleine Zettel beizulegen. Auch Mitteilungen auf Postabschnitten usw. sowie Fragen, die zwischen die Aufgabenlösungen eingestreut sind, werden leicht übersehen. Für Fragen zum Lehrstoff legen Sie bitte ein besonderes A4-Blatt bei, das Sie mit Namen und Studiennummer versehen.

Die Rücksendung der korrigierten Lösungen mit dem nächstfolgenden Lehrbrief ist mit Rücksicht auf die Arbeitseinteilung am Institut nicht möglich; der Versand der Lehrbriefe erfolgt unabhängig von den korrigierten Lösungen.

Sehen Sie die korrigierten Lösungen genau durch! Finden Sie eine unklare Korrektur, dann senden Sie diese mit einem entsprechenden Vermerk und sämtlichen zu diesem Lehrbrief gehörenden Lösungsblättern mit den folgenden Lösungen nochmals ein! Wir sind Ihnen dankbar, wenn Sie uns auf ein Versehen aufmerksam machen.

Die Einsendung der Aufgabenlösungen muß innerhalb von 5 Jahren erfolgen, vom Beginn des Studiums an gerechnet.

Wenn Sie diese Hinweise beachten, ist es uns möglich, Ihre Arbeiten in kürzester Zeit durchzusehen und zurückzusenden.

Prüfungsaufgaben

Prüfungsaufgaben

Lehrbrief 1

Bitte beachten Sie beim Anfertigen und Einsenden der Lösungen zu diesen Prüfungsaufgaben die allgemeinen Hinweise auf den Seiten P1 und P2.

1. In dem im Bild S6.1 dargestellten Programmablaufplan wurde die Kämm-Routine ohne Rücksicht darauf eingeführt, ob das Kämmen jeweils notwendig ist oder nicht. Diese Verbeugung vor der Eitelkeit haben wir bereits in der Aufgabe S6.1 kritisiert und dementsprechend nach der Anweisung „Verabschieden" eine Verzweigung in das Hauptprogramm eingeführt, die in Abhängigkeit vom Ergebnis des Blicks in den Spiegel die Kämm-Routine aufruft oder den Kamm unbenutzt läßt (vgl. Bild Ü1.1).

   Sicher wäre es sinnvoll, die Anweisung „Spiegel gucken" und nachfolgend die Verzweigung „Frisur in Ordnung" vor jeder Kämm-Routine in das Hauptprogramm einzufügen.

   Programmtechnisch eleganter ist es, die **jedesmal** vor der Kämm-Routine erscheinende Anweisung „Spiegel gucken" zu einem Teil des Unterprogramms zu machen. Das Unterprogramm heißt dann nicht mehr „Kämmen", sondern „Kämmen wenn nötig".

   Skizzieren Sie das Hauptprogramm entsprechend dem Bild S6.1 mit dieser Änderung! – Beachten Sie, daß eine Anweisung aus dem Bild S6.1 jetzt wegfallen kann!

   Skizzieren Sie – ähnlich wie im Bild S10.1 – das neue Unterprogramm neben dem Hauptprogramm! Gehen Sie dabei von der Darstellung im Bild S7.1 aus und ergänzen Sie diese Darstellung durch die Anweisung „Spiegel gucken" und die nachfolgende Verzweigung „Frisur in Ordnung?" – Fügen Sie vor der Grenzstelle „Ende Kämmen" eine Anweisung „Zurück zum Hauptprogramm" ein! (Vgl. Seite S10.) – Überlegen Sie genau, wohin die Wege nach der Verzweigung mit „Frisur in Ordnung?" führen!

   Skizzieren Sie die Sinnbilder für die Operationen in der Form und mit dem Seitenverhältnis, wie es die Bilder im Abschnitt „Der Programmablaufplan" zeigen!

2. Beim Entwickeln von Programmen ergibt sich sehr häufig die Notwendigkeit, daß bei bestimmten Adressen Ziffernpaare geändert werden müssen. Das ist immer dann der Fall, wenn man einzelne Anweisungen durch andere Anweisungen ersetzen will. Ein Beispiel dafür ist der Ersatz des Ziffernpaares A∅, das die UND-Verknüpfung bewirkt, durch das Ziffernpaar B∅, das für die ODER-Verknüpfung zuständig ist, im Versuch S29.1.

   Zur Lösung der hier gestellten Aufgabe sollen Sie in dem im Bild S25.1 aufgelisteten Programm folgende Ziffernpaare ändern. (Wir gehen davon aus, daß in der Anzeige Ihres Systems die System-Meldung „µPF– –1" steht.

Adresse	1. Eingabe	2. Eingabe
180E	4F	
180F	2F	
1810	E6	02
1812	CB	C9
1814	A1	
1815	A0	
1816	CB	20
1818	B0	
1819	F6	C0
181B	D3	02
181D	3E	02
181F	D3	01
1821	06	C9
1823	10	FE
1825	AF	
1826	D3	01
1828	18	D6

Bild P4.1
Zu Prüfungsaufgabe 3: Ersetzen Sie die Ziffernpaare aus der Programm-Liste im Bild S25.1 ab der Adresse 180E durch die hier aufgelisteten Ziffernpaare.

Adresse	Ziffernpaar	
1819	02 wird 40	(Starten Sie das Programm nach dieser Änderung und sehen Sie sich an, was passiert!)
181C	06 wird 00	(Diese vier Änderungen machen die Anweisung Nr. 6 unwirksam. Wenn Sie das Programm nach diesen Änderungen starten, dann sehen Sie, wozu das Warten gut ist.)
181D	C9 wird 00	
181E	10 wird 00	
181F	FE wird 00	

Geben Sie – ähnlich, wie wir es auf der Seite S26 getan haben – der Reihe nach sämtliche Tasten an, die Sie zur Änderung der Ziffernpaare betätigen müssen! Wenn die gleiche Taste zweimal nacheinander betätigt werden muß, dann schreiben Sie bitte die Bezeichnung dieser Taste auch zweimal an.

Anmerkungen oder Begründungen zu der jeweiligen Tasten-Betätigung brauchen Sie nicht zu geben.

3. Zur Lösung dieser Aufgabe müssen Sie ein Programm eintasten:

Versuch P4.1

Eine besondere Verknüpfung von Eingangssignalen

Tasten Sie das im Bild S25.1 aufgelistete Programm bis zum Ziffernpaar 20 bei der Adresse 180D einschließlich ein.

Tasten Sie ab der Adresse 180E die im Bild P4.1 aufgelisteten Ziffernpaare ein und starten Sie anschließend das Programm bei der Adresse 1800. (Sie wissen inzwischen, wie das gemacht wird.)

Betätigen Sie versuchsweise einzeln und gemeinsam die Tasten 7 und 3 und beobachten Sie die Anzeige! Sicher fällt Ihnen auf, daß sich das Ausgangssignal *a* anders verhält als bei den vorherigen Verknüpfungs-Versuchen. Sie haben jetzt eine besondere Art von UND-Verknüpfung programmiert.

Skizzieren Sie bitte die Funktionstabelle aus dem Bild P4.2 ab und ergänzen Sie diese Tabelle mit den Werten für das Ausgangssignal *a* bei den verschiedenen Kombinationen der Eingangssignale e_1 und e_2. Für das Eingangssignal e_1 ist die Taste 7 zuständig, für das Eingangssignal e_2 die Taste 3.

e_1	e_2	a
0	0	0
0	1	1
1	0	0
1	1	0

Bild P4.2
Zu Prüfungsaufgabe 3: Ergänzen Sie bitte diese Funktionstabelle entsprechend dem Ergebnis des Versuchs P4.1! – Sie erhalten eine UND-Verknüpfung mit einem negierten Eingangssignal.

4. Wandeln Sie die folgenden Zeichenfolgen in die Dezimaldarstellung um:

a) 23H; b) ABH; c) 3C7H; d) F2FH; e) 345H; f) C43BH

5. Stellen Sie die folgenden, dezimal dargestellten Zahlen sedezimal dar:

a) 17 b) 35 c) 463 d) 1024 e) 8467 f) 38752

Prüfungsaufgaben

Lehrbrief 2

Bitte beachten Sie beim Anfertigen und Einsenden der Lösungen zu diesen Prüfungsaufgaben die allgemeinen Hinweise auf den Seiten P1 und P2.

1. Im Laufe irgendeines Programms sind die Register Ihres Mikroprozessors mit folgenden Bitmustern geladen:

   (A) = ØlØØ ØlØl

   (BC) = 1ØØl ØØll Øll0 ll0l

   (DE) = ØØØl Øll0 ØØl0 lØØØ

   (HL) = llll ØØØl lll0 Ø1ØØ

   Tragen Sie die Register-Inhalte in sedezimaler und in dezimaler Schreibweise auf dem anschließend folgenden Lösungsblatt ein!

2. Tragen Sie das Bitmuster für die Zahl in das Schema eines Sechzehn-Bit-Registers auf dem Lösungsblatt ein, die dezimal durch 26325 dargestellt ist.

3. Lösen Sie die folgenden Aufgaben schriftlich und tragen Sie auf dem Lösungsblatt auch die Überträge bei den Additionsaufgaben und die „geborgten" Werte bei den Subtraktionsaufgaben ein (vgl. die Seiten S33 und S35):

   a) 6BØDH + 4C8H

   b) DD52H + 3BFDH

   c) B45CFH + 5C4DEH

   d) F36EH − AFBH

   e) 458AH − 2BFDH

4. Im Bild P5.1 haben wir eine Folge von drei Befehlen aufgelistet, deren dritten Sie noch nicht kennen. Er bewirkt die Addition der im Akkumulator und im C-Register stehenden Werte:

   (A) + (C) → A

   Das Additionsergebnis steht nach Ausführung des Befehls zweistellig im Akkumulator.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand
1		LD	A, F3	18ØØ	3E	F3
2		LD	C, CB	18Ø2	ØE	CB
3		ADD	A, C	18Ø4	81	

Bild P5.1
Zu Prüfungsaufgabe 4: Welche Inhalte haben das Flag-Register und der Akkumulator nach dem Ablauf dieser Befehle?

Versuch P6.1

Addition der Inhalte des Akkumulators und des C-Registers

Löschen Sie die Register A bis L so, wie es beim Versuch H15.1 beschrieben ist. Tasten Sie anschließend ab der Adresse 18ØØ die drei Befehle entsprechend dem Bild P5.1 ein.

Lassen Sie das Programm in Einzelschritten ablaufen und tragen Sie auf dem Lösungsblatt in die Darstellung des Flag-Registers und des Akkumulators die Bitmuster ein, die nach Ablauf des Befehls ADD A,C vorliegen. In der Register-Anzeige hat das Flag-Register die Abkürzung F.

5. Die Register des Mikroprozessors sollen per Programm wie folgt geladen werden:

   (A) = 45**H**; (B) = A2**H**; (C) = Ø1**H** (E) = Ø1**H**
   (H) = A2**H** (L) = 7F**H**

   Bei geschickter Anwendung der Befehle LD r,Konst, LD r,r' sowie LD rp,Konst brauchen Sie in einem Programm zur Lösung dieser Aufgabe vier Befehle (rp ist die Abkürzung für Registerpaar). Tragen Sie diese Befehle bitte in den Vordruck auf dem Lösungsblatt ein.

6. Aus wievielen Bytes besteht das Uhrenprogramm im Bild H9.1?

   Sie können die Bytes abzählen: 1 Byte = 8 Bit = zwei Sedezimalziffern. Es geht viel eleganter und einfacher.

   Geben Sie bitte die Anzahl der Bytes an und beschreiben Sie, wie Sie zu dieser Anzahl gekommen sind!

Prüfungsaufgaben

Bitte beachten Sie beim Anfertigen und Einsenden der Lösungen zu diesen Prüfungsaufgaben die allgemeinen Hinweise auf den Seiten P1 und P2.

1. Für den Datentransport zwischen CPU und Speicher gibt es die Möglichkeit der direkten Adressierung bestimmter Speicherzellen mit Drei-Byte-Befehlen [LD A,(adr), LD (adr),A] und der indirekten Adressierung über den Inhalt eines Registerpaares mit Ein-Byte-Befehlen [z. B. LD r,(HL) und LD (HL),r]. (Vgl. Seite S 47.) Vorzuziehen ist nach Möglichkeit die indirekte Adressierung. Nur wenn sich diese Möglichkeit verbietet, greift man zur direkten Adressierung.

   Das Bild P7.1 zeigt den Speicherbereich zwischen den Adressen 1934 und 1939. Die eingetragenen Speicher-Inhalte sollen in die rechts rot eingetragenen CPU-Register übertragen werden.

   Ergänzen Sie die auf dem Lösungsblatt angegebene Programmliste so, daß die gestellte Aufgabe erfüllt wird. Es stehen Ihnen die Befehle LD HL,Konst, LD r,(HL), INC HL (Seiten S 53 und S 54), LD A,(adr) und LD r,r' und eventuell LD A,(DE) (Seite S 51) zur Verfügung.

   Schließen Sie die Befehlsfolge mit dem Befehl RST 30 (Operationscode F7, vgl. Seite S 61) ab.

   Bei normaler Programmierung brauchen Sie (das Byte F7 des Befehls RST 30 eingeschlossen) für das Programm 19 Bytes. Wenn Sie es geschickt anstellen, brauchen Sie vier Bytes weniger. (Sehen Sie sich den Speicher-Inhalt im Bild P7.1 **genau** an!)

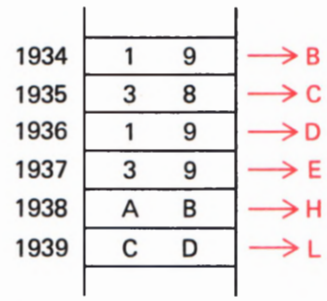


Bild P7.1
Zu Prüfungsaufgabe 1: Die Inhalte dieser Speicherzellen sollen in die rot eingetragenen Register übertragen werden.

2. Geben Sie bitte die Ergebnisse folgender Exklusiv-ODER-Verknüpfungen in Sedezimal-Schreibweise an:

   a) FC ⩢ 2A b) AB ⩢ AC c) 7B ⩢ FB d) 99 ⩢ 88

3. In irgendeinem Zusammenhang sind von einem Bitmuster im B-Register nur die Bits Nr. 0 und Nr. 7 von Interesse. Nur diese beiden Bits sollen ihre Werte behalten; alle anderen Bits (Nr. 1 bis Nr. 6) sollen die Werte 0 erhalten.

   Ergänzen Sie die Programm-Liste auf dem Lösungsblatt mit Befehlen, die diese Aufgabe erfüllen. – Schließen Sie die Befehlsfolge wie bei der Lösung der Aufgabe 1 mit einem RST 30-Befehl ab.

4. Zur Lösung dieser Aufgabe müssen Sie einen Versuch ausführen, in dem der auf der Seite S 53 vorgestellte Befehl INC HL verwendet wird.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	INKR	LD	HL,ØFFE	18ØØ	21	FE	ØF	(HL) = ØFFE
2	LOOP	INC	HL	18Ø3	23			HL ← (HL) + 1
3		JR	LOOP	18Ø4	18	FD		Sprung → LOOP

Bild P8.1
Zu Prüfungsaufgabe 4: Untersuchen Sie bitte im Versuch P8.1 den Inhalt des HL-Registerpaares jeweils nach der Ausführung des INC HL-Befehls!

Versuch P8.1

HL-Registerpaar inkrementieren

Tasten Sie bitte die im Bild P8.1 aufgelisteten drei Befehle in Ihr System ein.

Die Bemerkungen neben den Befehlen erläutern zusätzlich zum Programmablaufplan im Bild S8.2, was die Befehle bewirken: Das HL-Registerpaar wird vorab mit dem Wert ØFFE geladen und dann wird in einer Schleife bei jedem Schleifen-Durchlauf der Inhalt des HL-Registerpaares inkrementiert.

Lassen Sie die Befehlsfolge in Einzelschritten ablaufen und kontrollieren Sie jeweils nach Ausführung des Befehls Nr. 3 mit den Tasten REG und HL den Inhalt des HL-Registerpaares. – Vergessen Sie nicht, nach der Kontrolle des HL-Registerpaar-Inhalts mit der Taste PC den Inhalt des Programmzählers in die Anzeige zurückzuholen, damit Sie anschließend den nächsten Einzelschritt ausführen lassen können!

Tragen Sie in das Bild auf dem Lösungsblatt den Inhalt des HL-Registerpaares nach der ersten und der zweiten Ausführung des Befehls INC HL ein!

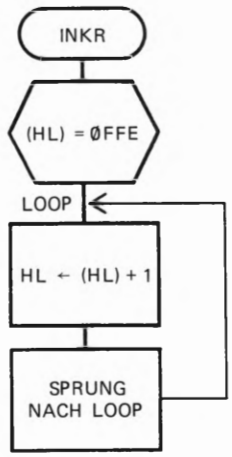


Bild P8.2
Zu Prüfungsaufgabe 4: Programmablaufplan der im Bild P8.1 aufgelisteten Befehlsfolge.

5. Bei der Lösung der Aufgabe H56.1 haben Sie den im Bild H56.1b markierten Inhalt des Speichers in den Bereich der Adressen 1814 bis 1817 verschoben.

   Tragen Sie bitte auf dem Lösungsblatt die Tasten-Betätigungen und die zugehörigen Anzeigen ein, die bei Ihrem System zu einer Verschiebung der in das Bild H56.1b eingetragenen Bytes von der Adresse 1814 bis einschließlich der Adresse 181A in einen Speicherbereich gehören, der mit der Adresse 181C beginnt.

Prüfungsaufgaben

Bitte beachten Sie beim Anfertigen und Einsenden der Lösungen zu diesen Prüfungsaufgaben die allgemeinen Hinweise auf den Seiten P1 und P2.

1. Bei der Adresse 18AC soll in irgendeinem Zusammenhang ein Programmzähler-relativer Sprung programmiert werden, der das Programm dann beim Label NEXT mit der Adresse 18E7 fortsetzt, wenn das CY-Bit im Flag-Register den Wert Ø hat (Bild P9.1).

   Die Art der Berechnung des Operanden ist bei allen Programmzähler-relativen Sprungbefehlen die gleiche. – Berechnen Sie bitte zuerst den Operanden für den benötigten Sprungbefehl, ohne daß Sie dazu Ihr System zu Hilfe nehmen, und geben Sie den Rechengang an.

   Geben Sie bitte außerdem den mnemonischen Code des bedingt auszuführenden Sprungbefehls mit dem Operanden NEXT sowie den sedezimal dargestellten Operationscode und den Operanden an.

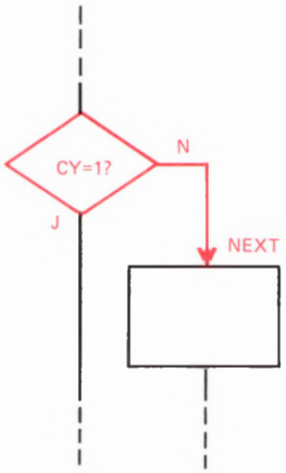


Bild P9.1
Zu Prüfungsaufgabe 1: Zu der hier rot eingetragenen Anweisung soll der Operationscode und der Operand des Programmzähler-relativen Befehls angegeben werden.

2. Geben Sie bitte an, welche Änderung Sie in der im Bild H69.2 aufgelisteten Befehlsfolge vornehmen müssen, wenn im Versuch H69.1 mit dem CALL-Befehl statt des Unterprogramms ALARM das Unterprogramm SIRENE mit den im Bild S99.2 aufgelisteten Bytes aufgerufen werden soll, das jedoch – abweichend vom Bild S99.2 – bei der Adresse 1843H beginnt.

   Tragen Sie den geänderten Befehl bitte in die Tabelle auf dem Lösungsblatt ein.

   Wir gehen davon aus, das die zum Unterprogramm SIRENE gehörenden Bytes im Speicher Ihres Systems ab der Adresse 1843 geladen sind.

3. Im Laufe eines beliebigen Programms werden die im Bild P9.2 aufgelisteten Befehle abgearbeitet, die sämtlich den Inhalt des Stack-Pointers beeinflussen. Zwischen dem ersten und dem letzten angeschriebenen Befehl enthält das Programm außer dem (im Bild nicht aufgeführten) Befehl RET am Ende des Unterprogramms UPRGRM keine anderen als die aufgelisteten Befehle, die den Inhalt des Stack-Pointers beeinflussen.

   Geben Sie bitte den Inhalt des Stack-Pointers nach der Ausführung des Befehls POP AF an.

Operation	Operand
LD	SP,1A78
..	..
CALL	UPRGRM
..	..
PUSH	BC
PUSH	DE
PUSH	HL
PUSH	AF
..	..
POP	AF

Bild P9.2
Zu Prüfungsaufgabe 3: Welchen Inhalt hat der Stackpointer nach der Ausführung des Befehls POP AF?

4. Entwerfen Sie bitte eine Befehlsfolge, die bei Verwendung der auf der Seite S87 als Hex-Dump angegebenen Programm-Teile EINGAB und ALARM bewirkt, daß nach der Eingabe von zwei Ziffern über das Tastenfeld diese beiden Ziffern miteinander ODER-verknüpft werden. Das Verknüpfungs-Ergebnis soll im Akkumulator stehen. Außerdem sollen die zu verknüpfenden Ziffern und das Verknüpfungs-Ergebnis in der Anzeige erscheinen.

   Sorgen Sie dafür, daß der Programm-Teil ALARM dann angesprungen wird, wenn im Verknüpfungs-Ergebnis eine **ungerade**

Anzahl von 1-Bits steht. Bei einer geraden Anzahl von 1-Bits im Verknüpfungs-Ergebnis soll das Programm sofort – ohne Ansprung des Programm-Teils ALARM – die Eingabe von zwei neuen Bytes erwarten.

Sie können bei der Lösung dieser Aufgabe von der im Bild S 87.1 aufgelisteten Befehlsfolge ausgehen. Ändern müssen Sie die Anweisung Nr. 3, mit der die Verknüpfung der ersten mit der zweiten Zahl bewirkt wird. (Ein Hinweis: Lesen Sie auf der Seite S 69 nach!) Außerdem muß die im Bild S 87.1 rot markierte Anweisung Nr. 7 geändert werden, mit der bestimmt wird, unter welcher Bedingung das Unterprogramm ALARM angesprungen wird. (Vgl. Seite S 92.)

Ändern Sie vor dem Ausprobieren der von Ihnen entworfenen Befehlsfolge noch das Byte Ø2 bei der Adresse 181F in das Byte A1! Sie erreichen damit, daß statt des Verknüpfungszeichens „–" ein dem Verknüpfungszeichen „v" für die ODER-Verknüpfung entsprechendes Zeichen in der Anzeige erscheint.

a) Tragen Sie die Folge der zur Lösung der Aufgabe notwendigen Befehle in die Tabelle auf dem Lösungsblatt ein!

b) Geben Sie bitte ein beliebiges Paar eingegebener Zahlen an, bei dem Ihre Befehlsfolge den Programmteil ALARM anspringt.

Übungen

Übungen

Lösungen der im Text gestellten Aufgaben

Bitte sehen Sie sich die folgenden Lösungen erst dann an, wenn Sie die im Text gestellten Aufgaben selbständig durchgearbeitet haben.

Zu Aufgabe H2.1

Die Schaltung kann durch das Leuchten der Lumineszenzdiode melden, daß in einem Raum mit zwei Türen die Gefahr des Durchzugs besteht.

Am Ausgang des NAND-Glieds erscheint dann – und nur dann – ein Ø-Signal, wenn am einen Eingang des NAND-Glieds UND (gleichzeitig) am anderen Eingang 1-Signale liegen. – Dieser Sachverhalt läßt sich auch so ausdrücken:

Am Ausgang des NAND-Glieds erscheint immer dann ein 1-Signal, wenn am einen Eingang ODER am anderen Eingang ein Ø-Signal liegt.

Das 1-Signal am Ausgang wird von der gegen L-Potential geschalteten Lumineszenzdiode durch Leuchten gemeldet.

Zu Aufgabe S6.1

Das Bild Ü1.1 zeigt den geänderten Teil des Programms entsprechend dem Bild S6.1.

Nach der Anweisung „Spiegel gucken" folgt eine Verzweigung in Abhängigkeit von der Entscheidung „Frisur in Ordnung?". Wenn die Frisur in Ordnung ist (Ja), dann kann das Kämmen unterbleiben. Es folgt sofort die Anweisung „Mantel anziehen".

Fällt die Entscheidung jedoch so aus, daß die Frisur nicht in Ordnung ist (Nein), dann wird der Programmzweig mit dem Aufruf des Unterprogramms „Kämmen" durchlaufen. Danach folgt die Zusammenführung mit dem Hauptprogramm, in dem die nächste Anweisung „Mantel anziehen" lautet.

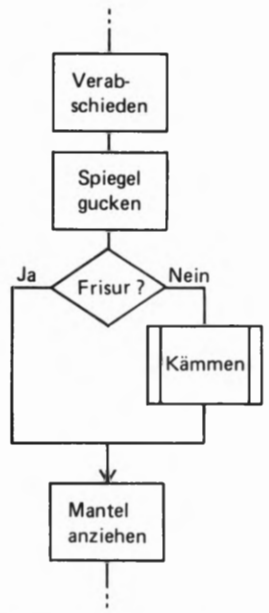


Bild Ü1.1
Zu Aufgabe S6.1. Das Unterprogramm „Kämmen" wird nur dann aufgerufen, wenn bei der Verzweigung die Entscheidung „Frisur in Ordnung? Nein!" gefällt wird.

Zu Aufgabe S8.1

Wenn nach dem Verlassen des Ankleidezimmers das Fähnchen nicht entfernt wird, geschieht zunächst gar nichts: Unser Professor marschiert programmgemäß zur Haustür.

Schlimm wird es am nächsten Morgen. Der Professor betritt das Ankleidezimmer, ohne sich um das Fähnchen zu kümmern. Beim Verlassen des Ankleidezimmers sieht er jedoch das Fähnchen vom vergangenen Morgen und entschreitet (wie programmiert) durch die Haustür. (Vorausgesetzt, daß er diesmal korrekt gekleidet ist. Wenn nicht, dann geht er wieder ins Ankleidezimmer, nimmt das Fähnchen aus der Öse und steckt es – laut Anweisung – wieder hinein. Er wird sich nicht mal wundern, denn er ist ja ein zerstreuter Professor.)

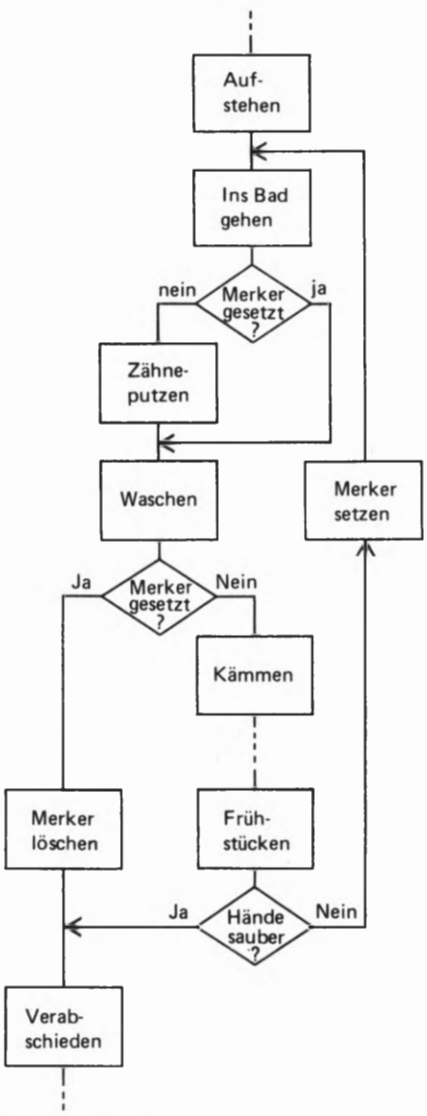


Bild Ü2.1
Zu Aufgabe S9.1. Der gleiche Merker wird zweimal abgefragt.

Zu Aufgabe S9.1

Der Programmablaufplan im Bild Ü2.1 spricht für sich selbst. Beachten Sie, daß der Merker zweimal abgefragt wird: Zuerst zum Umgehen der Anweisung „Zähneputzen", und noch einmal nach der Anweisung „Waschen". Erst danach darf der Merker gelöscht werden.

Zu Aufgabe H8.1

Erinnern Sie sich: Unabhängig von vorher eingegebenen oder **ausgeführten** Anweisungen bewirkt die im Tastenfeld rot markierte Taste RS (RESET) immer das Erscheinen der System-Meldung (Seite H7).

Es ist also ganz einfach: Betätigen Sie die Taste RS!

Zu Aufgabe H12.1

Weil Ihre Uhr sofort nach dem Start – ohne zu warten – um eine Sekunde weiterläuft, muß sie auf 18 Uhr, 7 Minuten und 59 Sekunden voreingestellt werden.

Das Bild Ü2.2 zeigt in der dem Bild H11.1 entsprechenden Darstellung, bei welchen Adressen welche Daten (Ziffernpaare) eingegeben werden müssen. Dabei haben wir bereits berücksichtigt, daß immer Ziffern**paare** einzugeben sind. Die Minuten müssen also in der Form Ø7 eingetastet werden.

Vor der Eingabe der Daten muß dem System mitgeteilt werden, bei welcher Adresse das erste, nämlich das Stunden-Ziffernpaar, stehen soll. Diese Mitteilung setzt die Betätigung der Taste ADDR voraus. Nach der dann folgenden Eingabe der Adresse 1AØØ können – wie bei der Eingabe der Daten aus dem Bild H9.1 – Ziffernpaare nacheinander eingetastet werden, wenn vor dem zweiten und dritten Ziffernpaar jeweils zur Erhöhung der Adresse die Taste + betätigt wird.

Vor dem Eingeben des ersten Ziffernpaares muß allerdings durch die Betätigung der Taste DATA angegeben werden, daß die Adreß-Eingabe beendet ist und nun Daten folgen.

Für die Eingabe der Start-Zeit ergibt sich also diese Tastenfolge:

ADDR, 1, A, 0, 0, DATA, 1, 8, +, 0, 7, +, 5, 9 (RS)

Die abschließende Betätigung der Taste RS ist nicht unbedingt notwendig.

	Daten		Adressen
Stunde	1	8	1AØØ
Minute	Ø	7	1AØ1
Sekunde	5	9	1AØ2

Bild Ü2.2
Zu Aufgabe H12.1. Zum Voreinstellen der Uhr auf 18 Uhr, 8 Minuten, Ø Sekunden werden die hier eingetragenen Daten bei den angegebenen Adressen abgelegt.

Zu Aufgabe S17.1

a) 4 mal zweihundertsechsundfünfzig + 2 mal sechzehn + 0 mal eins
oder
$4H \times 16^2 + 2H \times 16^1 + ØH \times 16^0$
oder
$4 \times 256 + 2 \times 16 + Ø \times 1$
= 420H = eintausendsechsundfünfzig

b) 5 mal zweihundersechsundfünfzig + A mal sechzehn + B mal eins
oder
$5H \times 16^2 + AH \times 16^1 + BH \times 16^0$
oder
$5 \times 256 + 10 \times 16 + 11 \times 1$
= 5ABH = eintausendvierhunderteinundfünfzig

c) F mal sechzehn + 4 mal eins oder:
$FH \times 16 + 4H \times 16^0$ oder
$15 \times 16 + 4 \times 1 = 244$
F4H = zweihundertvierundvierzig

d) Ø mal zweihundersechsundfünfzig + 9 mal sechzehn + D mal eins
oder
$ØH \times 16^2 + 9H \times 16^1 + DH \times 16^0$
oder
$Ø \times 256 + 9 \times 16 + 13 \times 1$
Ø9DH = einhundertsiebenundfünfzig

e) F mal viertausendsechsundneunzig + A mal zweihundertsechsundfünfzig + B mal sechzehn + C mal eins
oder
$FH \times 16^3 + AH \times 16^2 + BH \times 16^1 + CH \times 16^0$
oder
$15 \times 4Ø96 + 10 \times 256 + 11 \times 16 + 12 \times 1 = 64188$
FABCH = vierundsechzigtausendeinhundertachtundachtzig

Zu Aufgabe S20.1

a) In 166 ist $16^2 = 256$ nicht enthalten.

$166 : 16^1 = 166 : 16 = 1Ø = AH$ (Rest 6) **A.**
$6 : 16^0 = 6 : 1 = 6 = 6H$ **A6**

Einhundertsechsundsechzig = A6H

b) In 1147 ist $16^3 = 4Ø96$ nicht enthalten

$1147 : 16^2 = 1147 : 256 = 4 = 4H$ (Rest 123) **4..**
$123 : 16^1 = 123 : 16 = 7 = 7H$ (Rest 11) **47.**
$11 : 16^0 = 11 : 1 = 11 = BH$ **47B**

Eintausendeinhundertsiebenundvierzig = 47BH

c) In 3Ø21 ist $16^3 = 4Ø96$ nicht enthalten.

$3Ø21 : 16^2_1 = 3Ø21 : 256 = 11 = \mathbf{BH}$ (Rest 205) **B..**
$2Ø5 : 16^1_0 = 205 : 16 = 12 = \mathbf{CH}$ (Rest 13) **BC.**
$13 : 16^0 = 13 : 1 = 13 = \mathbf{DH}$ **BCD**

Dreitausendeinundzwanzig = BCDH

d) In 385Ø ist $16^3 = 4Ø96$ nicht enthalten.

$385Ø : 16^2_1 = 385Ø : 256 = 15 = \mathbf{FH}$ (Rest 1Ø) **F..**
$1Ø : 16^1_0 = 1Ø : 16 = Ø = Ø\,\mathbf{H}$ (Rest 10) **FØ.**
$1Ø : 16^0 = 1Ø : 1 = 1Ø = \mathbf{AH}$ **FØ.**

Dreitausendachthundertundfünfzig = FØAH **FØA**

e) In 51783 ist $16^4 = 65536$ nicht enthalten.

$51783 : 16^3_2 = 51783 : 4096 = 12 = \mathbf{CH}$ (Rest 2631) **C...**
$2631 : 16^2_1 = 2631 : 256 = 1Ø = \mathbf{AH}$ (Rest 71) **CA..**
$71 : 16^1_0 = 71 : 16 = 4 = 4\,\mathbf{H}$ (Rest 7) **CA4.**
$7 : 16^0 = 7 : 1 = 7 = 7\,\mathbf{H}$ **CA47**

Einundfünfzigtausend siebenhundertdreiundachtzig = CA47H

Zu Aufgabe S28.1

Das Bild Ü4.1a zeigt die Funktionstabelle einer UND-Verknüpfung.

Zu Aufgabe S29.1

Das Bild Ü4.1b zeigt die Funktionstabelle einer ODER-Verknüpfung.

Zu Aufgabe S30.1

Das Bild Ü4.1c zeigt die Funktionstabelle einer Exklusiv-ODER-Verknüpfung.

a)

e_1	e_2	a
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

b)

e_1	e_2	a
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

c)

e_1	e_2	a
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

Bild Ü4.1
Zu den Aufgaben S28.1, S29.1 und S30.1: a) Funktionstabelle einer UND-Verknüpfung;
b) Funktionstabelle einer ODER-Verknüpfung;
c) Funktionstabelle einer Exklusiv-ODER-Verknüpfung.

Lösungen der im Text gestellten Aufgaben

Bitte sehen Sie sich die folgenden Lösungen erst an, wenn Sie die im Text gestellten Aufgaben durchgearbeitet haben.

Zu Aufgabe H14.1

a) Es können zweihundertsechsundfünfzig unterschiedliche Zahlen (einschließlich der Zahl Null) abgelegt werden.

b) FF**H** = Zweihundertfünfundfünfzig.

Zu Aufgabe H17.1

Die einfachste Methode besteht darin, nacheinander die Tasten REG und AF zu betätigen.

Sie können aber auch zweimal nacheinander die Taste – betätigen. Nach der ersten Betätigung erscheinen die Dezimalpunkte an den beiden mittleren Anzeigestellen und melden damit die Bereitschaft des Systems, eine Eingabe in das C-Register entgegenzunehmen. Das ist hier natürlich uninteressant.

Nach der zweiten Betätigung der Taste – erfolgt automatisch die Inhalts-Anzeige der Register A (Akkumulator) und F (Flag-Register) und es besteht gleichzeitig eine Eingabe-Möglichkeit in den Akkumulator.

Zu Aufgabe H20.1

Initialisieren Sie mit der Taste REG die Register-Anzeige und anschließend mit der Taste AF die Anzeige der Inhalte von Akkumulator und Flag-Register! Mit der Taste DATA schaffen Sie sich die Eingabe-Möglichkeit von Daten, zunächst in das Flag-Register. Die Taste + verschiebt dann die Dezimalpunkte auf die für den Akkumulator-Inhalt zuständigen, linken Anzeigestellen. Jetzt können Sie mit den weißen Daten-Tasten 1 und A den Wert 1A**H** in den Akkumulator eingeben.

Nach nochmaligem Betätigen der Taste + erscheinen automatisch die Inhalte der B- und C-Register: X X X.X. B C, wobei die Dezimalpunkte an den mittleren Anzeigestellen die Eingabe-Möglichkeit von Daten in das C-Register signalisieren. – Durch einmaliges Betätigen der weißen Daten-Taste 0 wird das C-Register gelöscht.

Mit der Taste + ergibt sich dann die Eingabe-Möglichkeit in das B-Register: X.X.0 0 B C. Auch dieses Register läßt sich durch einmaliges Betätigen der Taste 0 löschen.

Zu Aufgabe H22.1

Im Versuch H21.1 wird zuerst das B-Register mit dem Inhalt 1A**H** des Akkumulators und anschließend das C-Register mit dem Inhalt des Akkumulators geladen.

Das Bild H20.1 zeigt mit einem roten Pfeil, daß das C-Register auch mit dem Inhalt des B-Registers geladen werden kann. Da nach dem Befehl LD B,A die Inhalte (B) = (A) = 1AH sind, läßt sich im zweiten Programmschritt der Wert 1AH für das C-Register auch aus dem B-Register holen. Entsprechend dem Formalismus der mnemonischen Codes für die Lade-Befehle muß dieser Befehl lauten:

LD C,B

Zu Aufgabe S33.1

1. Summand:

$DH \times 16^2 + AH \times 16^1 + 9H \times 16^0 = 13 \times 256 + 10 \times 16 + 9 \times 1 = 3497$

2. Summand:

$BH \times 16^2 + 5H \times 16^1 + 8H \times 16^0 = 11 \times 256 + 5 \times 16 + 8 \times 1 = 2904$

Summe: 6401

In 6401 ist $16^4 = 65536$ nicht enthalten.

$6401 : 16^3 = 6401 : 4096 = 1 = 1H$	(Rest 2305)	1...
$2305 : 16^2 = 2305 : 256 = 9 = 9H$	(Rest 1)	19..
$1 : 16^1 = 1 : 16 = 0 = 0H$	(Rest 1)	190.
$1 : 16^0 = 1 : 1 = 1 = 1H$		1901

Ergebnis: 6401 = 1901H

Diese Zeichenfolge haben wir auch bei der Addition der sedezimal dargestellten Zahlen erhalten.

Zu Aufgabe S34.1

a) 7CH + 3BH = B7H

b) 135H + 427H = 55CH

c) ABCH + DEFH = 18ABH

d) 1E4AH + 2AC1H = 490BH

Zu Aufgabe S36.1

a)

	F	C	H
−	3	2	H
	C	A	H

b)

	4	6'	7	H
		5	17	H
−	4	5	A	H
	0	0	D	H

Ü

6

c)
```
    5'  A'  6   H
    4 ↳19 ↳16   H
-   2   C   7   H
-------------------
    2   D   F   H
```

d)
```
    F   F   7'  C   H
            6 ↳1C   H
-   A   F   5   F   H
-----------------------
    5   Ø   1   D   H
```

Zu Aufgabe H25.1

Der vorhergehende Versuch hat gezeigt, daß mit dem Befehl LD A,Konst der Akkumulator mit dem im Operanden des Befehls stehenden Wert geladen wird. Der erste Befehl des Programms sorgt also dafür, daß im Akkumulator die mit AB sedezimal dargestellte Zahl abgelegt wird.

Die beiden folgenden Befehle LD B,A und LD C,A kennen Sie bereits vom Versuch H21.1 her. Sie bewirken hier, daß der Wert AB jeweils vom Akkumulator in die Register B und C kopiert wird.

Die mnemonischen Codes der folgenden vier Befehle Nr. 4 bis Nr. 7 sind den Befehlen Nr. 2 und Nr. 3 ganz ähnlich. Es handelt sich um Kopier-Befehle, bei denen jeweils das Herkunfts-Register der Akkumulator ist. Da vor dem Komma das Zielregister steht, ist leicht zu erkennen, daß nacheinander der im Akkumulator abgelegte Wert AB in die Register D, E, H und L kopiert wird.

Das Programm bewirkt das Laden sämtlicher, im Bild H23.1 dargesteller Register mit dem Wert AB.

Zu Aufgabe H29.1

Das Bild Ü7.1 zeigt die gewünschte Befehlsfolge. Das Programm ist fast doppelt so lang wie das Programm im Bild H26.1, das das gleiche bewirkt, das aber überwiegend aus Ein-Byte-Befehlen besteht. Selbstverständlich ist das kürzere Programm zu bevorzugen.

Ein Programm mit Kopier-Befehlen wird bei einer Aufgabe wie dieser nur dann funktionieren, wenn in die Register durchweg gleiche Werte eingetragen werden sollen. Sollen in die Register unterschiedliche Werte eingetragen werden, dann muß man ein Programm der Art verwenden, wie es hier angegeben ist. Wir werden Ihnen jedoch in einem der nächsten Abschnitte Befehle zeigen, die das Laden von Registern mit unterschiedlichen Werten unter bestimmten Bedingungen wesentlich eleganter vornehmen.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand
1		LD	A, AB	18Ø0	3E	AB
2		LD	B, AB	18Ø2	Ø6	AB
3		LD	C, AB	18Ø4	ØE	AB
4		LD	D, AB	18Ø6	16	AB
5		LD	E, AB	18Ø8	1E	AB
6		LD	H, AB	18ØA	26	AB
7		LD	L, AB	18ØC	2E	AB

Bild Ü7.1
Zu Aufgabe H29.1. Diese Befehlsfolge bewirkt das Laden der Register A bis L mit dem konstanten Wert AB. Sie ist fast doppelt so lang wie die Befehlsfolge im Bild H26.1, die das gleiche bewirkt.

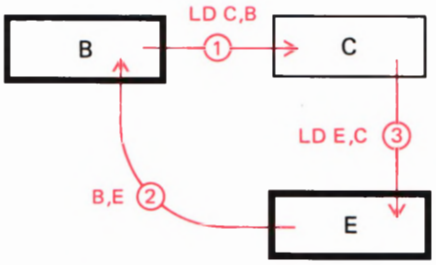


Bild Ü 8.1
Zu Aufgabe H 30.1. Für den Austausch von zwei Register-Inhalten mit LD-Befehlen muß der Inhalt des einen der Register zwischenzeitlich in ein anderes Register gerettet werden.

Zu Aufgabe H 30.1

Das Bild Ü 8.1 zeigt das Lösungsschema, bei dem das Register C zur Zwischenablage des ursprünglichen Inhalts des B-Registers benutzt wird. Natürlich kann man auch eins der Register A, D, H oder L zur Zwischenablage benutzen.

Mit dem ersten Befehl (LD C,B) wird der ursprüngliche Inhalt des B-Registers „gerettet", damit er beim nachfolgenden zweiten Befehl (LD B,E) nicht vom ursprünglichen Inhalt des E-Registers überschrieben wird und dann verloren ist. – Mit dem dritten Befehl (LD E,C) gelangt dann der zwischendurch gerettete, ursprüngliche Inhalt des B-Registers in das C-Register.

Machen Sie einen entsprechenden Versuch!

Zu Aufgabe S 39.1

Mit fünf Stellen ergibt sich die Zeichenkombination 1 1 1 1 1 als Darstellung der größten Zahl im Dualsystem. Diese Zeichenkombination bedeutet:

Einmal 2^4 + einmal 2^3 + einmal 2^2 + einmal 2^1 + einmal 2^0 = 31

Die größte Zahl, die mit fünf Stellen im Dualsystem dargestellt werden kann, ist also Einunddreißig.

Zu Aufgabe S 39.2

Wir geben nur für die Aufgabe a den Lösungsweg noch einmal an; er ist so einfach, daß wir ihn uns für die übrigen Aufgaben sparen können.

a) Die am weitesten links stehende Ziffer hat den Stellenwert 2^3. Es ergibt sich also:

$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 0 + 2 + 0 = 10$

b) 21 c) 49 d) 92 e) 197

Zu Aufgabe S 40.1

a) 111**B** b) 1101**B** c) 100100**B** d) 110110100**B**

e) 11110111001**B**

Zu Aufgabe H 36.1

Die größte Zahl enthält ein Sechzehn Bit „breites" Register dann, wenn sämtliche Bits die Werte 1 haben:

1111 1111 1111 1111 **B** = FFFF**H** = 65535

Wir haben das Bitmuster so angeschrieben, daß die Zusammensetzung aus vier Halbbytes deutlich wird. Ein Sechzehn-Bit-Register kann demnach auch als Zwei-Byte-Register bezeichnet werden.

Lösungen der im Text gestellten Aufgaben

Bitte sehen Sie sich die folgenden Lösungen erst dann an, wenn Sie die im Text gestellten Aufgaben selbständig durchgearbeitet haben.

Zu Aufgabe S44.1

Die einzelnen Befehle haben folgende Wirkungen:

Nr. 1: Der Inhalt 12 der Speicherzelle mit der Adresse 183B wird in den Akkumulator kopiert und

Nr. 2: im B-Register zwischenzeitlich aufbewahrt. Das ist notwendig, weil beim anschließenden Transport des Bytes AB von der Speicherzelle 1923 über den Akkumulator in die Speicherzelle mit der Adresse 183B das Byte 12 sowohl im Akkumulator als auch in der Speicherzelle mit der Adresse 183B überschrieben wird und damit verschwindet.

Nr. 3: Transport des Bytes AB von der Speicherzelle 1923 über den
Nr. 4: Akkumulator in die Speicherzelle 183B.

Nr. 5: Das Byte 12, das ursprünglich in der Speicherzelle 183B
Nr. 6: stand, wird aus dem B-Register in den Akkumulator und von dort in die Speicherzelle 1923 kopiert.

Statt der umständlichen Bezeichnung „Speicherzelle mit der Adresse „XXXX" sagt man oft auch vereinfachend: „Speicherzelle XXXX".

Zu Aufgabe S54.1

Unter Löschen wird das Ablegen von Bytes 00 in einer Speicherzelle verstanden. Im Programm im Bild S53.1 wird dieses Byte mit dem Operanden des Befehls Nr. 3, LD (HL),00 in die vom Inhalt des HL-Registerpaares adressierte Speicherzelle gebracht.

Als erste soll die Speicherzelle mit der Adresse 1900 gelöscht werden. Auf diese Speicherzelle muß also der Inhalt des HL-Registerpaares am Anfang des Programms zeigen. Das wird mit dem Befehl Nr. 1, LD HL,1900 erreicht.

Die Anzahl einhundert der zu löschenden Speicherzellen wird in sedezimaler Form mit dem Befehl Nr. 2, LD B,64**H** im B-Register abgelegt.

Im Programm müssen also folgende Änderungen vorgenommen werden:

Adresse 1F92	statt Byte 18	jetzt Byte 19
Adresse 1F94	statt Byte 10	jetzt Byte 64
Adresse 1F96	statt Byte AB	jetzt Byte 00

Machen Sie einen entsprechenden Versuch!

Zu Aufgabe H48.1

Die Berechnung der Übertragungs-Dauer des Uhren-Programms aus dem Speicher auf das Band ist entsprechend der schematischen Darstellung im Bild H48.1 recht einfach. – Die Programm-Daten des Uhren-Programms sind ab der Adresse 18ØØ bis zur Adresse 1847 abgelegt. Einschließlich des Bytes bei der Adresse 18ØØ besteht das Programm also aus 48**H** = 72 Bytes.

Für die Übertragungs-Dauer gilt also:

1-KHz-Synchronisation	4,00 s
7 Informations-Bytes x 60 ms	0,42 s
2-KHz-Synchronisation	2,00 s
72 Bytes Programm x 60 ms	4,32 s
End-Synchronisation	2,00 s
Übertragungs-Dauer	12,74 s

Sie können diese Zeit leicht kontrollieren, wenn Sie die Ausgabe der Daten – auch ohne einen Cassetten-Recorder anzuschließen – über den Lautsprecher des Systems verfolgen und die Übertragungs-Zeit mit einer Stoppuhr messen. Wahrscheinlich werden Sie eine etwas längere als die berechnete Zeit messen, weil Ihre eigene Reaktionszeit nicht zu vernachlässigen ist.

Zu Aufgabe S60.1

a) 2B ⩛ EA = C1 b) 75 ⩛ 1Ø = 65
c) AB ⩛ AB = ØØ d) 3C ⩛ FF = C3

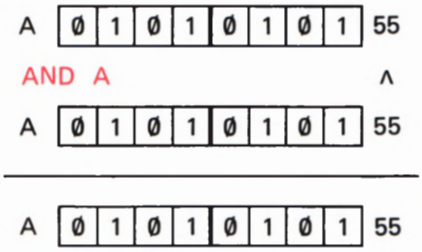


Bild Ü10.1
Zu Aufgabe S63.1: Am Beispiel des Akkumulator-Inhalts 55 wird deutlich, daß der Befehl AND A den Akkumulator-Inhalt nicht beeinflußt.

Zu Aufgabe S63.1

Das Bild Ü10.1 zeigt am Beispiel des Akkumulator-Inhalts 55, daß der Befehl AND A den Akkumulator-Inhalt unverändert läßt.

Es sei darauf hingewiesen, daß alle Befehle für logische Verknüpfungen, also auch der AND A-Befehl, den Inhalt des Flag-Registers, das wir später kurz vorstellen werden, beeinflussen. Der Verknüpfungsbefehl AND A, der den Inhalt des Akkumulators unverändert läßt, wird in Programmen manchmal nur zur Beeinflussung des Flag-Register-Inhalts verwendet.

Ü

10

Bild Ü10.2
Zu Versuch Ü11.1: Befehlsfolge zur UND-Verknüpfung des Akkumulator-Inhalts mit sich selbst.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
1	START	LD	A,55	18ØØ	3E	55		A ← 55
2		AND	A	18Ø2	A7			A ← (A) ∧ (A)
3		RST	3Ø	18Ø3	F7			Breakpoint

Versuch Ü11.1

Der Befehl AND A

Tasten Sie bitte die im Bild Ü10.2 aufgelisteten Befehle ein und starten Sie den Befehlsablauf bei der Adresse 18ØØ! Sehen Sie sich nach der System-Meldung den Inhalt des Akkumulators an (REG, AF)! – Wiederholen Sie den Versuch nach Änderung des Operanden des Befehls bei der Adresse 18Ø1! – Es ist immer

$$(A) \wedge (A) = (A)$$

Zu Aufgabe H53.1

Es müssen folgende Tasten betätigt werden:

Tasten	Anzeige	Bemerkungen
ADDR, 1, 8, 1, 0	1.8.1.0. 0 0	Einstellen der Adresse des ersten NOP-Befehls.
DATA, 2, 1	1 8 1 0 2.1.	Eingabe des Operationscodes des Befehls LD HL,19ØØ.
+, (0, 0)	1 8 1 1 0.0.	Eingabe des ersten Operanden dieses Befehls. Die folgenden beiden Bytes müssen zusätzlich eingefügt werden. – Nach dem Eintasten des Bytes ØØ wird für die Anwendung der INS-Anweisung jetzt automatisch die richtige Adresse angezeigt, die um eins kleiner ist als die Adresse der Speicherzelle, bei der das folgende Byte eingefügt werden muß. (Vgl. Seite H53.)
INS	1 8 1 2 0.0.	Daten-Eingabe-Modus bei der Adresse, bei der ein Byte eingefügt werden soll. Zunächst wurde das Byte ØØ eingetragen.
1, 9	1 8 1 2 1.9	Eingabe des zweiten Operanden des Befehls LD HL,19ØØ. – Es wird wieder automatisch die richtige Adresse für die nachfolgende Anwendung der INS-Anweisung angezeigt.
INS	1 8 1 3 0.0.	Daten-Eingabe-Modus für das nächste, einzufügende Byte.
B, 6	1 8 1 3 B.6.	Eingabe des Operationscodes des Befehls OR (HL).

Wenn Sie die Tasten-Eingaben im Zusammenhang mit der INS-Anweisung aufmerksam betrachten, dann stellen Sie fest, daß sich die INS-Taste beim Eingeben von Daten in den Speicher ganz ähnlich verwenden läßt wie die Taste +. Mit einem Unterschied: Die Taste INS sorgt automatisch dafür, daß alle Bytes bei den folgenden Adressen um eine Speicherzelle nach oben verschoben werden.

Es folgen noch die notwendigen Programm-Korrekturen:

ADDR, 1, 8, 1, 6	1.8.1.6. 1.8	
DATA, 1, A	1 8 1 6 1.A.	
+, +, +, E, 6	1 8 1 9 E.6.	
ADDR, 1, 8, 2, 2	1.8.2.2. 3 7	
DATA, 3, 9	1 8 2 2 3.9.	
+, +, +, 3, E	1 8 2 5 3.E.	
ADDR, 1, 9, 0, 0	1.9.0.0 XX	Bei dieser Adresse wird das einzu-ODERnde Bitmuster abgelegt, z.B. Ø2:
DATA, 0, 2	1 9 0 0 0.2	

Nach diesen Eingaben können Sie das Programm starten. Sie erhalten das gleiche Ergebnis wie beim Versuch S69.1.

Zu Aufgabe H56.1

Es müssen folgende Tasten betätigt werden:

Tasten	Anzeige		Bemerkungen
MOVE			Das System erwartet die Eingabe der Anfangs- (Start-) Adresse des Datenblocks.
1, 8, 1, 8	1 .8 .1 .8 .	−S	Anfangs-Adresse
+	X.X.X.X.	−E	Das System erwartet die End-Adresse des Datenblocks.
1, 8, 1, B	1 .8 .1 .B .	−E	End-Adresse
+	X.X.X.X.	−D	Das System erwartet die Adresse, bei der das erste Byte des Datenblocks stehen soll.
1, 8, 1, 4	1 .8 .1 .4 .	−D	Ziel-Adresse (**D**estination)
GO	1 8 1 7	0.7.	Letztes Byte des verschobenen Blocks.

Im Gegensatz zur Blockverschiebung im Versuch H56.1 wird hier nach der Ausführung der Verschiebung die Adresse mit dem letzten Byte des verschobenen Blocks angezeigt. Der Grund liegt darin, daß beim Schieben eines Blocks zu höheren Adressen die Verschiebung bei der höchsten Adresse des Blocks beginnt, beim Schieben eines Blocks zu niedrigeren Adressen die Verschiebung bei der niedrigsten Adresse beginnt.

Lösungen der im Text gestellten Aufgaben

Bitte sehen Sie sich die folgenden Lösungen erst dann an, wenn Sie die im Text gestellten Aufgaben selbständig durchgearbeitet haben.

Zu Aufgabe S79.1

Wir nehmen an, die Adresse 18F5 sei mit dem Label NEXT gekennzeichnet. – Das Bild Ü13.1 zeigt, wie sich der zum Inhalt des Programmzählers zu addierende Wert, also der Operand des JR-Befehls, errechnen läßt. Wenn x der Operand des JR-Befehls ist, dann gilt:

(PC) nach Fetch von JR NEXT		Operand		Adresse NEXT	
18AD	+	x	= 18F5		x = 18F5 − 18AD = 48

Der JR-Befehl muß also den Operanden 48 haben.

Dieses Ergebnis liefert auch Ihr System, wenn Sie folgende Tasten betätigen:

ADDR, 1, 8, A, B	Adresse des Operationscodes 18 des JR-Befehls
RELA, +	Anforderung der Sprung-Ziel-Adresse
1, 8, F, 5	Sprung-Ziel-Adresse
GO	Anzeige der Operanden-Adresse und des Operanden.

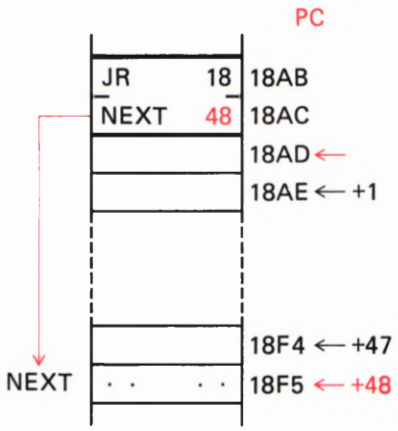


Bild Ü13.1
Zu Aufgabe S79.1: Nach diesem Schema läßt sich der Operand des Programmzähler-relativen Sprungs berechnen.

Zu Aufgabe S87.1

Zum Berechnen des Operanden für einen Programmzähler-relativen Sprung bei der Adresse 185Ø zur Ziel-Adresse 18ØØ müssen folgende Tasten betätigt werden:

ADDR, 1, 8, 5, 0	Adresse des Operationscodes 18 des JR-Befehls
RELA, +	Anforderung der Sprung-Ziel-Adresse
1, 8, 0, 0	Sprung-Ziel-Adresse Steht da schon
GO	Anzeige der Operanden-Adresse und des Operanden.

Machen Sie einen Versuch! Führen Sie zunächst noch einmal den Versuch S86.1 durch und ersetzen Sie dann die drei Bytes für den Befehl JP RECHN ab der Adresse 18ØØ durch die zwei Bytes für den JR-Befehl.

Zu Aufgabe S89.1

Sie müssen zunächst den Operanden des bei der Adresse 18ØA programmierten, Programmzähler-relativen Sprungbefehls zur Adresse 183E des Labels ALARM bestimmen. Nach dem Schema, das Sie bei den Lösungen der vorhergehenden Aufgaben benutzt haben, liefert Ihnen Ihr System den Operanden 32.

Jetzt können Sie die drei Bytes C2, 3E, 18 ab der Adresse 180A durch die Bytes 20 32 00 für den Befehl JR NZ,ALARM und einen nachfolgenden NOP-Befehl (NOP = **N**o **OP**eration, keine Operation) ersetzen. – Machen Sie einen Versuch!

Zu Aufgabe S91.1

Das Ergebnis ist ganz einfach: Beim Versuch S90.1 wurde der ALARM-Teil des Programms immer dann angesprungen, wenn das Rechenergebnis negativ (**M**inus) war. Nach dem Ersatz des Befehls JP **M**,ALARM durch den Befehl JP **P**,ALARM wird der Programm-Teil ALARM immer dann angesprungen, wenn das Rechenergebnis positiv (**P**lus) ist.

Interessant ist, daß auch das Rechenergebnis null (00) als positives Ergebnis gewertet wird.

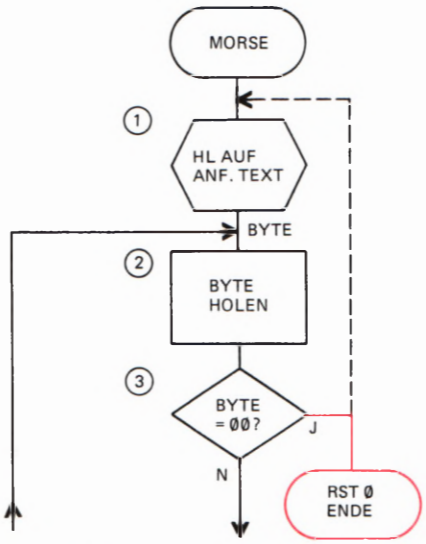


Bild Ü14.1
Zu Aufgabe S95.1: Beim Byte 0 kann statt eines Neu-Starts des Programms die System-Meldung programmiert werden.

Zu Aufgabe S95.1

Die Lösung dieser Aufgabe ergibt sich aus der Betrachtung des Programmablaufplans im Bild S94.1. – Das Ende des Morse-Textes erkennt das Programm am Byte 00, das im Speicher nach dem letzten Morse-Byte abgelegt ist (Seiten S94 und S95). Die vom Programm aus dem Speicher geholten Morse-Bytes werden mit der Anweisung Nr. 3 auf diesen Wert 00 geprüft. Wenn das Byte 00 erkannt wird, dann springt das Programm normalerweise zurück zur Anweisung Nr. 1, mit der das HL-Registerpaar so gesetzt wird, daß es auf das erste Morse-Byte im Speicher zeigt.

Wenn sich das Programm nach dem Entdecken des Bytes 00 so verhalten soll, als würde die Reset-Taste betätigt, dann muß statt des Sprunges zur Anweisung Nr. 1 ein Sprung zu einer Anweisung programmiert werden, welche die gleiche Wirkung wie das Betätigen der RS-Taste hat. Eine solche Anweisung ist der Befehl RST 0 mit dem Operationscode C7 (vgl. Aufgabe S44.1). In das Bild Ü14.1 haben wir eine entsprechende Änderung des Programmablaufs rot eingetragen.

Da der Befehl RST 0 im Programm nicht vorgesehen ist, muß er als „Rucksack" angehängt werden. Platz dafür findet sich bei der Adresse 186C (Bild S96.2). – Zum Anspringen dieses Befehls muß der Operand des Befehls Nr. 4, JR Z,adr, bei der Adresse 1806 so geändert werden, daß die Adresse 186C erreicht wird. – Ihr System berechnet Ihnen nach der in den vorangegangenen Lösungen beschriebenen Methode für den Operanden dieses Befehls den Wert 64.

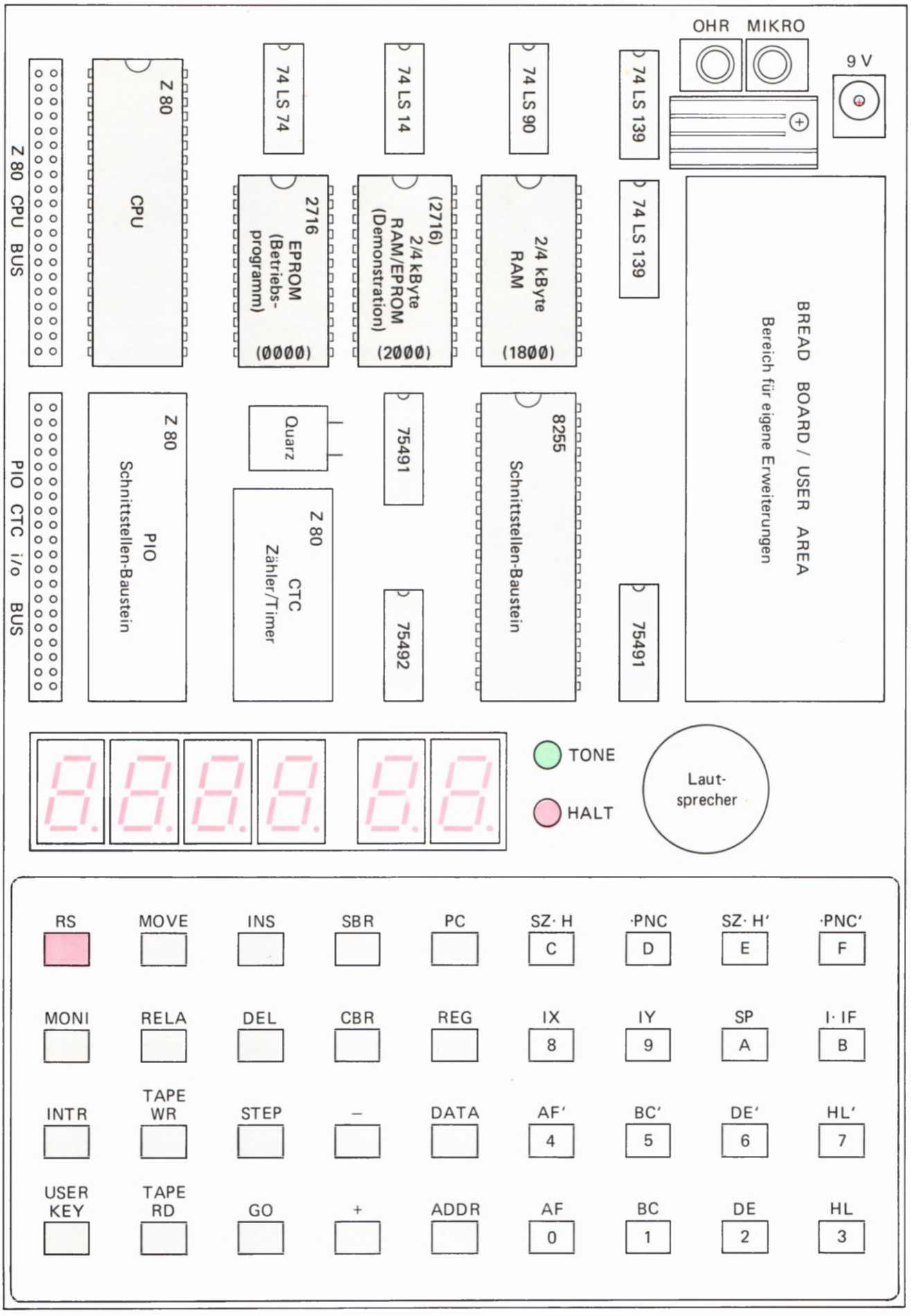
Das Bild Ü14.2 zeigt die Änderung und Ergänzung, die im Programm vorgenommen werden muß.

Bild Ü14.2
Zu Aufgabe S95.1: Für das in den Bildern S96.1 und S96.2 aufgelistete Morse-Programm ergeben sich diese Änderungen, wenn der Morse-Text nur einmal ausgegeben werden soll.

Nr.	Label	Operation	Operand	Adresse	Opcode	Operand	Operand	Bemerkungen
4		JR	Z,RUCKS	1806	28	64		Ja: System-Meldung
21	RUCKS	RST	0	186C	C7			System-Meldung

Tabellen

Der Mikro-Professor



Register und Anschlüsse des Z80-Mikroprozessors

Die dem 8085-Mikroprozessor entsprechenden Register sind grau unterlegt.