EINFÜHRUNG

Inhalt	Seite
Über diesen Kurs	0 - 2
Zeitplan und Tests	0 - 3
Kursbetreuung und Hilfen	0 - 4
Liste der Kursbetreuer	0 - 5
Zum Arbeitsablauf	0 - 6
Das Arbeitsmaterial	0 - 7

Über diesen Kurs

Unser "EINFÜHRUNGSKURS MIKROPROZESSOREN" beginnt im März 1981 und soll ein Jahr dauern.

Sie werden mit den Lehrbüchern "Z–80 Einführung und Programmierung" und "Z–80 Interface–Technik und Anwendung" arbeiten, die Bestandteile des Nanocomputer Trainingskonzeptes der Fa. SGS-Ates sind.

Das Informationsangebot, das diese Bücher enthalten, ist weit umfangreicher, als für einen Einführungskurs nötig wäre. Wir haben Ihnen deshalb mit der Arbeitsmappe einen Leitfaden gegeben, der Ihnen den Einstieg und die Arbeit mit dem Material erleichtern soll. Sie werden diese mehr als 700 Seiten nicht Blatt für Blatt durcharbeiten müssen, nur die für den Einstieg wesentlichen Passagen.

Dies gilt ganz besonders für den Band "Interface–Technik und Anwendung", der zunächst nur wenig verwendet wird, Ihnen aber später viele Möglichkeiten der Arbeit mit dem Mikroprozessor eröffnet.

Zeitplan und Tests

Der Kurs ist in sechs Teile gegliedert. Dadurch wird die Fülle an Informationen in überschaubare, einzeln abgeschlossene Abschnitte aufgeteilt. Es ist sinnvoll, jeweils ein Kursteil in einem Zuge durchzuarbeiten. Die Arbeitszeit teilen Sie sich selbst ein.

Versuchen Sie, sich einen ganzen Tag innerhalb der zwei Monate, die für jeden Teil vorgesehen sind, für diese Arbeit freizuhalten. Erfahrungsgemäß kommen Sie so besser zum Ziel, als mit kurzen Abschnitten, die sich über Tage und Wochen hinziehen.

Die Kursteile schließen mit einem Test ab. Diese Tests sollen Ihnen eine Bestätigung für den Erfolg Ihrer Arbeit geben im Sinne eines Rückblickes oder eines Schlußstriches unter einen Arbeitsabschnitt, nicht im Sinne einer Leistungskontrolle.
Deshalb werde ich Ihre Tests nicht benoten, sondern nur korrigieren und - falls erforderlich- auch Erklärungen und Hilfen geben.

Kursbetreuung und Hilfen

Falls Sie bei Ihrer Arbeit auf Fragen und Probleme stoßen, die Sie allein nicht lösen können, bieten wir Ihnen folgende Hilfen an:

Während der üblichen Bürostunden können Sie mich telefonisch in der SRT erreichen:

Rolf Dräger Tel. 0911/61 20 45.

Für schriftliche Anfragen verwenden Sie einfach einen der beiliegenden EM-Umschläge. Schreiben Sie Ihre Fragen nur auf einen Zettel, formvollendete Briefe sind nicht nötig. Nur bitte ich Sie, alle Zuschriften mit Namen und Rundfunkanstalt zu kennzeichnen, um Verwechselungen auszuschließen. Bitte beschriften Sie auch einen Adressenaufkleber mit Ihrer Anschrift für den Antwortbrief.

Oder wenden Sie sich an Ihren Fernkursbetreuer. In jeder Rundfunkanstalt haben sich Kollegen bereiterklärt, Sie im Rahmen ihrer Möglichkeiten zu unterstützen. Die Liste der Fernkursbetreuer finden Sie auf der Seite 0 – 5.

Ich hoffe sehr, daß Sie eine dieser Kontaktmöglichkeiten nutzen und wir so zu einer guten Zusammenarbeit kommen werden.

Betreuer für den Fernkurs EM 01

BR Rainer Bachmann, Sender Dillberg, 8431 Postbauer
Tel. 0918/87 73

Lothar Sack, HA Techn. Betrieb FS, Floriansmühl-
str. 60, 8000 München 45, Tel. 089/38 06 22 81

HR Herr Zappe, Bildmeßtechnik, Bertramstr. 8,
6000 Frankfurt/Main, Tel. 0611/15 52 42 9

Walter Träger, ARD-Sternpunkt, Bertramstr. 8,
6000 Frankfurt/Main, Tel. 0611/59 05 56

NDR Rudolf Janz, Meßtechnik Bild, Gazellenkamp 54,
2000 Hamburg-Lockstedt, Tel. 040/41 34 36 0

RB Gerhard Brich, HF Meßtechnik, Heinrich-Hertz-Str. 21
2800 Bremen

SR Werner Brill, Fernsehmeßtechnik, Am Halberg,
6600 Saarbrücken, Tel. 0681/60 23 91

SFB Dieter Alfer, Hochfrequenztechnik, Masurenallee 8 - 14,
1000 Berlin 19, Tel. 030/30 82 11 8 (22 77)

SDR Siegfried Bokelmann, Zentraltechnik, Neckarstr. 230,
7000 Stuttgart, Tel. 0711/28 82 19 4

SWF Helmut Kleine, Meßtechnik, Landesstudio Mainz,
Postfach 37 40, 6500 Mainz, Tel. 06131/30 23 05

Werner Blum, FS/PT-BT, Hans-Bredow-Straße,
7570 Baden-Baden, Tel 07221/27 62 68 2

ZDF Engelbert Günter, Studio Bonn, Langer-Graben-
Weg 45 - 47, 5300 Bonn-Bad Godesberg, Tel. 0228/88 61

Zum Arbeitsablauf

Sie finden genaue Angaben über den Arbeitsablauf zu jedem Abschnitt hier in der Arbeitsmappe. Die von den Lehrbüchern vorgegebene Aufteilung in theoretische und praktisch-experimentelle Arbeiten, gibt Ihnen große Freizügigkeit in der Wahl des Arbeitsplatzes und der Arbeitszeit. Da Sie den Nanocomputer nur für die praktischen Arbeiten brauchen, genügt oft auch eines der Bücher, um bei einer günstigen Gelegenheit für den Fernkurs zu arbeiten.
Die vorgegebenen Bearbeitungszeiträume von zwei Monaten pro Kursteil sind Richtwerte, Sie können Ihre Tests gern früher abschicken.

Das Arbeitsmaterial

Jeder Kursteilnehmer erhält als Studienmaterial:

1. Lehrbuch Band 1
   Z–80 Einführung und Programmierung
2. Lehrbuch Band 3
   Z–80 Interface–Technik und Anwendung
3. SRT–Arbeitsmappe

Im Laufe des Kurses ergänzen wir die Arbeitsmappe und schicken Ihnen Programmformulare und eine Befehlsliste.

Für die praktischen Arbeiten stellt Ihnen die Rundfunkanstalt einen Nanocomputer. Diese Geräte sind mit eigenem Begleitmaterial versehen, das Sie nach Abschluß des Kurses zusammen mit dem Gerät wieder abgeben müssen. Es besteht aus folgenden Teilen:

1. Technisches Handbuch (englisch) mit Schaltbildern
2. Nanocomputer Bedienungsanleitung (deutsch)
3. Nanocomputer NC–Z Software DN 314 (englisch)
4. Nanocomputer NE–Z Software DN 340 (englisch)

Bitte überprüfen Sie das Material auf Vollständigkeit.

1. Teil

Bearbeitungszeitraum:
April, Mai 1981

Inhalt	Seite
Übersicht	1 - 2
Sprungmöglichkeiten	1 - 3
Kapitel 1 und 2, Anleitung	1 - 4
Weitere Übungsaufgaben	1 - 5
Lösungen dazu	1 - 6
Kapitel 3, Anleitung	1 - 7
Kapitel 4, Anleitung	1 - 9
Kapitel 5, Anleitung	1 - 11
Zur Schreibweise der Adressen	1 - 12
1. Test	1 - 13

1. Teil - Übersicht

Im ersten Teil bearbeiten Sie die Kapitel 1 – 5 im Lehrbuch Z–80 Einführung und Programmierung. Es geht hierin zunächst einmal um die wichtigsten Grundbegriffe der Computertechnik und ein erstes Kennenlernen des Nanocomputers.

Die Kapitel 1 – 3 stellen die Einführung für den Anfänger dar. Vieles davon wird Ihnen bereits aus der Digitaltechnik bekannt sein, Sie werden zügig vorankommen.

Die Kapitel 4 und 5 bringen Ihnen erste Versuche mit dem Übungsgerät, Sie lernen auch schon einige Befehle des Z–80 kennen.

Jedes Kapitel ist eine in sich geschlossene Einheit mit Vorschau und Zusammenfassung in Form einiger Fragen. Dadurch ergibt sich eine Gliederung des Lernstoffes und die Möglichkeit nach Unterbrechungen an das vorangegangene Kapitel anzuschließen. Bitte lesen Sie zunächst weiter in der Arbeitsmappe.

Sprungmöglichkeiten

Wenn Sie bereits über die Grundbegriffe der Computertechnik Bescheid wissen, sollten Sie folgende Möglichkeiten zur Arbeitserleichterung nutzen:

Jedes Kapitel schließt mit einem Rückblick in Form von Fragen über den Inhalt und den Antworten dazu. Diese Fragen geben nach meiner Auffassung ein wirklich ehrliches Bild des Inhalts wieder und gestatten Ihnen deshalb selbst zu beurteilen, ob Sie ein Kapitel durcharbeiten müssen, oder ob Sie gleich zum nächsten übergehen können. Möglicherweise können Sie die ersten beiden Kapitel auf diese Weise überspringen.

Lesen Sie diese Kapitel gründlich durch und versuchen Sie, die Fragen im Rückblick selbst zu beantworten. Im Hexadezimal-Code sollten Sie keine besonderen Geheimnisse suchen. Es ist lediglich eine vereinfachte Schreibweise für die Bitmuster der Binärzahlen. Die Kunstziffern A bis F hat man nur deshalb eingeführt, weil man damit alle 16 Kombinationen einer Gruppe von 4 Bit mit einstelligen Ziffern kennzeichnen kann. Auf der nächsten Seite (1 – 4) finden Sie noch weitere Übungsaufgaben zur Zahlenumwandlung.

Weitere Übungsaufgaben zur Zahlenumwandlung

(Bearbeitung freiwillig)

Wandeln Sie mit Hilfe der Tabelle auf S. 13 folgende Bitmuster in Hexzahlen um:

a. 0 0 0 1 1 0 0 0 — 18 H
b. 1 1 1 0 1 0 0 1 — E9 H
c. 0 1 0 0 1 1 0 1 — 4D H
d. 1 0 1 0 0 1 1 1 — A7 H

Finden Sie zu folgenden Hexzahlen das zugehörige Bitmuster:

e. F 9 H — 1111 1001
f. 1 A H — 0001 1010
g. 0 3 H — 0000 0011
h. B B H — 1011 1011

Geben Sie zu den folgenden Hexzahlen jeweils die vorangehende und die nachfolgende Hexzahl an:

i. B F H — BEH B0H
j. 2 C H — 2BH 2DH

Lösungen zu den Übungsaufgaben

a. 18H

b. E9H

c. 4DH

d. A7H

e. 11111001

f. 00011010

g. 00000011

h. 10111011

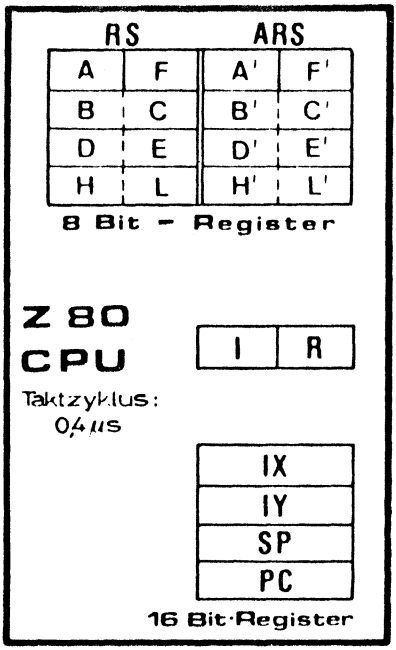
i. BEH, C0H

j. 2BH, 2DH

Nun geht es ganz konkret um den Z-80 Mikroprozessor. Lesen Sie dieses Kapitel zunächst nur bis zum Abschnitt "Einige Z-80 Befehle" auf S. 37.

Hier nun noch einige Ergänzungen zu den Registern der CPU:

Die Allzweck-Register der CPU sind nur eine - wenn auch sehr nützliche - Zugabe des Z-80 Mikroprozessors. Ihre Funktion könnte auch ein bestimmter Bereich des Arbeitsspeichers (RAM) übernehmen, nur ergeben sich durch die Anordnung der Register in der CPU besonders kurze Befehle und Ausführungszeiten für den Datenaustausch. Diese CPU-Register müssen beispielsweise nicht über 16-Bit Adressen angesprochen werden. Sie werden diese Register sehr häufig verwenden. Auf der nächsten Seite finden Sie ein Schema der Z-80 CPU, das Ihnen die Orientierung erleichtert. Sie arbeiten praktisch nur mit dem Registersatz RS (A, B, C, D, E, H, L), der alternative Registersatz ARS wird im Zusammenhang mit Programmunterbrechungen interessant.


RS
ARS
A
F
A'
F'
B
C
B'
C'
D
E
D'
E'
H
L
H'
L'
8 Bit - Register
Z 80
CPU
Taktzyklus:
0,4 µs
I
R
IX
IY
SP
PC
16 Bit-Register

Arbeiten Sie nun auf S. 37 weiter. Die Darstellung "(addr.)" wird Ihnen sofort klarer, wenn Sie davon ausgehen, daß eine Adresse schließlich nur die "Hausnummer" eines Speicherplatzes ist. Einmal wird die "Hausnummer" verändert, im anderen Fall mit "(addr.)" wird der Inhalt dieser Speicherzelle, also das, was unter der "Hausnummer" zu finden ist, verändert.

Der Abschnitt "Befehls-Byte-Terminologie" auf S. 38 enthält einen Druckfehler:

Es muß heißen L D < B 2 > < B 3 > , A.

Die Fragen 2, 3 und 5 im Rückblick S. 41 erfordern ein Nachschlagen im Text, niemand wird so etwas aus dem Kopf beantworten können.

An diesem Kapitel gibt es einiges zu streichen:
Bitte streichen Sie folgende Abschnitte:

1. Der Nanocomputer S. 44, 45
2. LD und DP S. 52, 53, 54, 55 (wird später behandelt)
3. Regeln zum Versuchsaufbau S. 58, 59, 60 oben (wird ebenfalls später behandelt)
4. 1. Schritt, 2. Schritt, S. 68, 69 (die Gefahr einer Beschädigung der Speicherbausteine ist zu groß)

Beginnen Sie nun mit der Bearbeitung dieses Kapitels. Die Erklärung der Tastatur wird am Schluß des Kapitels in den Versuchen noch durch Beispiele ergänzt, außerdem sollten Sie auch im Anschluß daran die Beispiele aus der Bedienungsanleitung ab S. 17 einmal durchspielen.
Aber nun zunächst einmal zum Text dieses Kapitels. Die Seite 57 enthält zwei Fehler: Im oberen Teil heißt es: Seine Übertragungsgeschwindigkeit beträgt 110 Baud. Dieser Wert stimmt nicht, das Betriebssystem legt 600 Baud fest, die Baudrate kann jedoch beliebig verändert werden (siehe Bedienungsanleitung Seite 15).
Der zweite Fehler betrifft die Experimentier–Platine. Im letzten Satz der Seite 57 heißt es, die äußeren Gruppen der Anschlüsse wären mit +5 Volt und Masse verbunden. Das stimmt nicht, die Anschlüsse sind frei.

Die beiden Adressen für die Testprogramme auf Seite 70 lauten:

MEMTUT	FADC
CONTST	FB43

Bitte denken Sie auch an die Beispiele in der Bedienungsanleitung, dort werden die Tastatureingaben noch einmal Schritt für Schritt angegeben.

Wenn Sie die ersten Kapitel übersprungen haben, finden Sie hier noch einmal eine Zusammenfassung der bisher angesprochenen Z-80 Befehle und einige Wiederholungen aus dem Kapitel 2.

Den Text auf S. 75 ab "Ist ein Programm ..." sollten Sie mit Vorbehalt lesen, die "Handassemblierung" eines Programms ist bei weitem nicht so schwierig, wie es hier dargestellt wird. Sie werden diese Erfahrung selbst noch machen, denn schließlich ist das eines der Ziele unseres Lehrgangs.

Die auf S. 76 dargestellte Form der Programmaufzeichnung ist heute fast überall eingeführt, nehmen Sie diese Dinge hier zunächst einmal zur Kenntnis, später werden Sie die Bedeutung noch kennenlernen.
Bitte nehmen Sie sich Zeit für die Versuche zum Kapitel 5. Hier wird noch einmal schön langsam und ausführlich die Eingabe in den Nanocomputer erläutert. Ganz nebenbei lassen Sie auch schon erste Programme ablaufen und lernen dabei weitere Z-80 Befehle kennen. Blättern Sie in Zweifelsfällen zurück oder lesen Sie die Bedienungsanleitung, falls Sie über die Bedeutung der Eingabetasten zusätzliche Informationen brauchen. Arbeiten Sie möglichst nicht mit der RESET-Taste. Diese löscht die CPU-Register und damit die Ergebnisse. Zur Unterbrechung eines Programms verwenden Sie besser die BREAK-Taste, die Sie ohne Veränderung der CPU in das Betriebssystem, also auf die Anzeige, zurückbringt.

Zur Schreibweise der Adressen

Sie haben bereits gelernt, daß der Mikroprozessor manche Befehle nur ausführen kann, wenn er in den nachfolgenden Befehlsbytes eine Adresse vorfindet. Da nur jeweils ein Byte in einer Speicherzelle Platz findet, müssen Adressen, die aus 2 Bytes bestehen, geteilt und nacheinander eingegeben werden. Welche Hälfte kommt zuerst? Für den Z–80 gilt, daß stets zuerst das LO–Adreßbyte, dann das HI–Adreßbyte im Speicher stehen.

Z. B.: Es lautet ein Befehl: Springe zur Adresse 0 1 2 3. Der Objekt–Code heißt: C 3, 2 3, 0 1.

Für den Z–80 müssen Sie also die Schreibweise der Adresse umkehren. Nicht jedoch für den Mnemonik–Code. Dort finden Sie: C 3, 2 3, 0 1 entspricht J P 0 1 2 3 H.

Falls im Moment der Programmerstellung die Sprungadresse noch nicht festgelegt ist, schreibt man häufig auch:

C 3, < B 2> , <B 3> J P < B 3> ,< B 2>

Dieses Verwirrspiel soll Ihnen nicht etwa das Leben schwer machen, hier wird lediglich darauf hingewiesen, daß im Befehlscode die Reihenfolge für Adressenangaben mit L O vor H I vorgeschrieben ist.

1. Test

Karl Blocking SDR
Name und Rundfunkanstalt

1.1 Bitte analysieren Sie das folgende Programm.
Welche Wirkung hat es?

Speicheradresse	Objekt-Code	Quelle-Code
0100	3E	LDA, 00H
0101	00	<00>
0102	3C	INC A
0103	32	LD (0109H), A
0104	09	<L0>
0105	01	<HI>
0106	C3	JP 0102H
0107	02	<L0>
0108	01	<HI>

0101 gehört zu 0100!

Lade AKKu mit 00, 1001 Kein Abbruch

NOP Befehl, inkrementiere AKKu +1

Speichere Inhalt des AKKus nach Speicheradresse

109H, Speicherstelle LO, HI, Unbedingter Sprung nach 0102. Schleife beginnt von vorn.

1. Test

Karl Bloching SDR
Name der Rundfunkanstalt

1.2 Wie lautet die Hexzahl für folgendes Bitmuster?

1 1 0 1 / 0 1 0 1 / 1 0 1 0 / 0 0 1 1

D5A3 H

1.3 Nennen Sie die allgemein verwendbaren 8-Bit-Register des Z-80 Mikroprozessors.

A, B, C, D, E, H, L

1.4 Die Anzeige des Nanocomputers zeigt 0 4 2 5 0 0 und es leuchtet die M E M - L E D. Was müssen Sie eingeben, um den Inhalt der Speicherzelle 0 4 0 1 zur Anzeige zu bringen?

0401 mit LA Taste eingeben

1.5 Ist es möglich, die Speicherzelle F B 4 0 H Ihres Nanocomputers mit F 7 H zu laden?

~~ja~~/nein - Begründung: Rombereich

Bitte schicken Sie uns einen Adressenaufkleber mit Ihrer Anschrift für den Antwortbrief mit.

mit

1. Test – Lösungsblatt

1.1 Dieses Programm besteht aus einer Zählschleife, in der der Akkuinhalt bei jedem Durchlauf um eins erhöht wird. Der dabei erreichte Zählerstand wird bei jedem Durchlauf in der Speicherzelle 0 1 0 9 abgespeichert. Der erste Befehl setzt den Akku auf den Wert Null, er sorgt für eindeutige Anfangsbedingungen.

1.2 Die Hexzahl lautet D 5 A 3 H.

1.3 Im Z–80 gibt es folgende 8–Bit–Register zur allgemeinen Verwendung:
B, C, D, E, H, L, B', C', D', E', H', L'.

1.4 Es ist nacheinander einzugeben:
0, 4 0 1, L A.

1.5 Nein, denn diese Adresse gehört zum ROM-Bereich.

2. Teil

Bearbeitungszeitraum:
Juni, Juli 1981

Inhalt	Seite
Übersicht	2 - 2
Sprungmöglichkeiten	2 - 3
Kapitel 6, Anleitung	2 - 4
Kapitel 6, Hinweise zu den Experimenten	2 - 5
Korrekturblatt	2 - 6
Kapitel 7, Anleitung	2 - 7
Ergänzungen	2 - 8
Die Bedeutung des Stack	2 - 10
Tabelle der Adressierungsarten	2 - 12
2. Test	2 - 14

2. Teil - Übersicht

Im zweiten Teil bearbeiten Sie die Kapitel 6 und 7 in Ihrem Lehrbuch Band I. Hauptthema sind die verschiedenen Adressierungsarten des Z-80 Mikroprozessors. Sie werden dabei viele neue Begriffe kennenlernen, die ganz grundsätzliche Bedeutung für alle Mikroprozessor-Systeme und Programmiervorhaben in Maschinensprache haben.

Wir werden auch wieder einige Passagen im Buch aussparen, die für uns unbedeutend sind, bitte halten Sie sich deshalb an die Vorschläge in der Arbeitsmappe.

Sprungmöglichkeiten

Folgende Erleichterungen möchte ich Ihnen vorschlagen, falls Sie besonders wenig Zeit zur Verfügung haben, oder bereits genügend Erfahrung im Umgang mit Mikroprozessoren besitzen:

1. Beginnen Sie mit den Experimenten zu Kapitel 6 auf Seite 108. Der entsprechende Abschnitt in der Arbeitsmappe beginnt auf Seite 2 - 5.

2. Bearbeiten Sie anschließend das Kapitel 7 ab Seite 137 (Arbeitsmappe ab Seite 2 - 7).

3. Überspringen Sie den Rückblick (Seite 150 – 154) und gehen Sie gleich weiter zu den Experimenten auf Seite 154 (Arbeitsmappe Seite 2 - 11).

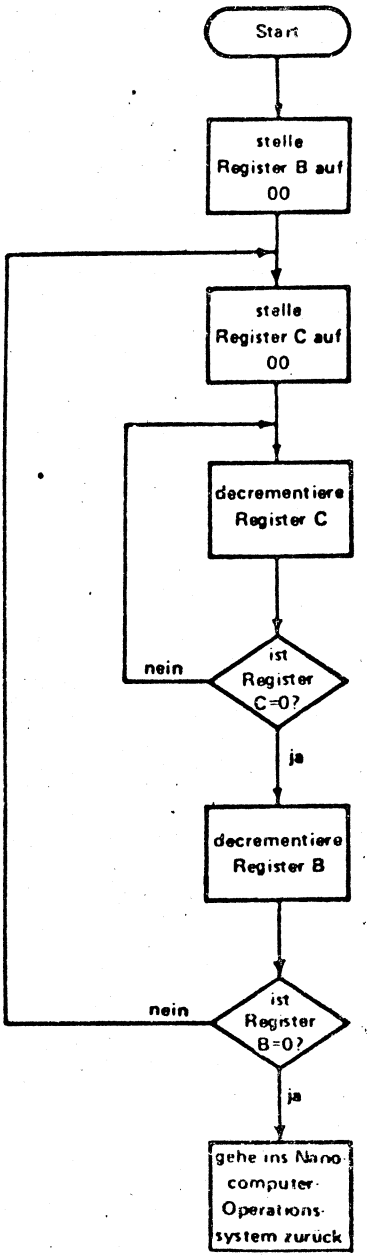
Kapitel 6 beginnt mit einer Erläuterung zum Aufbau des Befehls–Codes und der Bedeutung der Bits innerhalb eines Befehls–Bytes. Diese Betrachtungen haben für uns nur geringen Nutzen, da wir im Hexcode arbeiten, die innere Struktur des Z–80 Befehls aber auf dem Octalcode aufbaut und deshalb der logische Zusammenhang innerhalb der 3–Bit–Registercodierung teilweise verlorengeht. Deshalb möchte ich Ihnen die Bearbeitung der Seiten 90 bis 101 freistellen.
Auf Seite 91 unten bei "Anmerkung": sind Klammern vergessen worden. Es muß heißen: (HL)

Nun zu dem sehr wichtigen Abschnitt über die Adressierungsarten ab Seite 102:
Viele Mikroprozessoren verwenden mehrere Adressierungsarten. Damit wächst die Flexibilität für den Programmierer. Besonders vielseitig ist hierin der Z–80, nahezu alle Adressierungsarten, die bei anderen Mikroprozessoren vorkommen, sind in diesem System enthalten. So kommt Ihnen die Arbeitszeit, die Sie hier investieren, auch zugute, wenn Sie einmal ein anderes System kennenlernen wollen.
Bitte arbeiten Sie diesen Abschnitt durch. Zu den Experimenten folgen Hinweise auf der nächsten Seite.

Kapitel 6, Hinweise zu den Experimenten

Die Experimente ab Seite 108 sind wichtig für das Verständnis der Adressierungsarten, sie geben Beispiele für die verschiedenen Möglichkeiten. Erwarten Sie aber bitte noch keine praxisbezogenen Anwendungsfälle, im Moment stehen wir noch am Anfang und müssen uns deshalb mit diesen sehr theoretischen Betrachtungen auseinandersetzen. Die ersten Ansätze für die praktische Verwendung liegen in den Zeitschleifenprogrammen in Versuch 3. Da der Mikroprozessor quarzgesteuert ist, lassen sich mit Hilfe von Zeitschleifen sehr exakte Zeitverzögerungen einstellen.

Bild 6 – 6 auf Seite 118 enthält einen Fehler. Die richtige Darstellung finden Sie auf dem nächsten Blatt (S. 2 – 6).



Korrekturblatt zu Bild 6.– 6 auf Seite 118, Band I.

Kapitel 7, Anleitung

Kapitel 7 beginnt mit der Erklärung der Zweierkomplement–Darstellung, einem Thema, das ich Ihnen im Rahmen dieses Einführungskurses ersparen möchte. Im Prinzip geht es dabei um eine Möglichkeit, negative Zahlen darzustellen. Wir kommen später noch darauf zurück.

Gehen Sie gleich bis zur Seite 137. Bearbeiten Sie den Abschnitt: "Die Z–80 Adressierungsarten" zunächst nur bis zur Seite 140. Sie bekommen für die darauf folgenden Abschnitte noch Ergänzungen auf Seite 2 – 8 in der Arbeitsmappe.

Erläuterungen zur indizierten und relativen Adressierung:
Diese beiden Adressierungsarten geben die Zieladresse in der Form eines Abstandswertes an, im Computerdeutsch heißt das "Distanzbyte". Unterschiedlich ist bei diesen Verfahren nur der Ausgangspunkt, von dem aus der Abstand gerechnet wird: Bei der indizierten Adressierung steht der Ausgangspunkt als "echte" Adresse in einem der 16-Bit Indexregister IX oder IY. Bei der relativen Adressierung ist der Ausgangspunkt der Wert des Programm-Counters, der nach dem betreffenden Befehl erreicht ist.

Die Bestimmung des Abstandswertes ist für beide Adressierungen gleich:
In Vorwärtsrichtung, also in Richtung aufsteigender Adressen, wird der Abstand ausgezählt und ganz normal im Hex-Code angegeben.
Beispiel:

Zieladresse	0427
Ausgangspunkt	- 041B
Distanzbyte d	0C

In Rückwärtsrichtung, in Richtung fallender Adressen, muß "rückwärts" gezählt werden, denken Sie hierbei an einen Kilometerzähler, den Sie vom Stand 00 an, rückwärts drehen: 00, 99, 98, 97 usw.

In der Hex–Schreibweise sieht das folgendermaßen aus: 0 0, F F, F E, F D, F C, F B, F A, F 9 usw.

Diese Zahlen sind nun die Zweierkomplement–Darstellung der entsprechenden negativen Zahlen, die sich als Abstandswerte in "Rückwärtsrichtung" ergeben. An dem Wert des höchstwertigen Bits (Bit 7) im Distanzbyte erkennt der Prozessor, ob das Ziel in Vorwärts– oder Rückwärtsrichtung liegt. Dieses Bit wird deshalb bei Zahlendarstellungen auch als Vorzeichen-Bit bezeichnet:

Bit 7 = 0 – positive Zahl

Bit 7 = 1 – negative Zahl

Dieses Bit geht dabei aber für die eigentliche Zahlendarstellung verloren. Vorzeichenbehaftete Zahlen in einem 8–Bit Prozessor erstrecken sich deshalb nur über einen verringerten Zahlenbereich von:

	Hex	Dezimal
positiv	0 0 bis 7 F	0 0 bis 1 2 7
negativ	F F bis 8 0	- 1 bis - 1 2 8

Lesen Sie nun die Seiten 140 bis 146 im Lehrb uch durch. Anschließend gibt ihnen die Arbeitmappe (Seite 2 – 10) Ergänzungen zum Abschnitt "Stack–Pointer".

Die Bedeutung des Stack

Moderne Mikroprozessoren besitzen neben den Universalregistern der CPU auch die Fähigkeit, in einem bestimmten Bereich des RAM einen speziellen Speicher aufzubauen, den Stack. Dies ist eine Art von Notizbuch oder Merkzettelsammlung für die CPU. Der Stack dient also primär als Hilfsregister während eines Programmlaufes. Vorzugsweise werden dort Rücksprungadressen während der Bearbeitung von Unterprogrammmen aufbewahrt oder es werden bei Verzweigungen die Inhalte von CPU-Registerpaaren zwischengespeichert (auf den Stack gerettet).
Die CPU verwaltet ihren Stack selbst. Ein besonderes 16-Bit Register, der Stack-Pointer, zeigt der CPU den nächsten freien Stackplatz an.
Im Stack wird immer mit 16-Bit (vollständige Adressen oder Inhalte von Registerpaaren) gearbeitet, der Stack-Pointer bewegt sich also immer um zwei RAM-Adressen weiter. Der Stack-Pointer in der CPU ist nach einem Reset leer, das Betriebssystem muß den Stack-Pointer zunächst einmal mit einer vernünftigen Anfangsadresse im RAM laden. Aus praktischen Gründen legt man den Stack an die höchsten RAM-Adressen, der Stack-Pointer bewegt sich üblicherweise "rückwärts" im Adress-Bereich, er wächst also von der oberen RAM-Grenze nach unten. Ist nur wenig RAM-Speicher in einem System vorgesehen, so kann der Stack bei Fehlern im Programm durchaus auch einmal in den Bereich der Arbeitsdaten am Anfang des RAM wachsen und dort vorhandene Daten überschreiben, bei uns kommt so etwas selbstverständlich nicht vor!

Soweit die zusätzlichen Erläuterungen zum Stack. Lesen Sie nun im Lehrbuch ab S. 146 weiter.
Da wir die Zweierkomplement–Darstellung weggelassen haben, sollten Sie im Rückblick (S. 150) die Fragen 1 bis 4 auslassen. Die Frage 5 können Sie leicht bearbeiten, indem Sie sich eine Tabelle für die Distanzbytes in Rückwärtsrichtung anlegen. (0 0, F F, F E, F D . . . usw)
Die abschließenden Versuche zu diesem Kapitel sollten Sie mit viel Sorgfalt durchdenken. Hier wird Ihnen der Sinn und Nutzen der verschiedenen Adressierungsarten durch die Gegenüberstellung unterschiedlicher Programme verdeutlicht.

Übrigens setzt unser Nanocomputer–Betriebssystem den Stack–Pointer auf die Adresse 0 F 0 0 und nicht auf 0 F C E, wie auf Seite 162 unten angegeben ist.

Die Adressierungsarten, Kurzdarstellung

1. Registeradressierung. Einbytebefehl
   Datenbewegung zwischen CPU–Registern und über (HL) in das RAM.

2. Unmittelbare Adressierung. Zweibytebefehl
   Eines der CPU–Register oder eine RAM–Zelle (über (HL)) werden mit den Daten aus dem zweiten Byte geladen.

3. Indirekte Register–Adressierung. Einbytebefehl
   Datenbewegung zwischen dem Akku und einer RAM–Zelle, deren Adresse in einem CPU–Registerpaar B C, D E, H L steht.

4. Unmittelbare erweiterte Adressierung. Dreibytebefehl
   Die zwei letzten Bytes des Befehls werden direkt in ein CPU–Registerpaar geladen.

5. Erweiterte Adressierung. Dreibytebefehle
   Datenbewegung zwischen einem CPU–Registerpaar und zwei benachbarten RAM–Zellen. Die Zelle mit dem LOW–Byte wird im Befehl adressiert.

6. Modifikation der Seite Null. Einbytebefehl
   Verkürzte Sprungadressierung für acht Restartadressen im Bereich HI–Byte 0 0.

7. Relative Adressierung. Zweibytebefehl
   Bestimmung einer Zieladresse über ein Distanzbyte relativ zum Stand des Programmzählers nach Ausführung des Befehls.

8. Indizierte Adressierung. Dreibytebefehl
   Adressierung einer Zieladresse über ein Distanzbyte relativ zu einer Adresse in einem der Indexregister I X oder I Y.

9. Implizierte Adressierung. Einbytebefehl
   Einige Befehle erfordern keine besondere Adressierung, da sie ausschließlich den Akku betreffen.

10. Einzelbit–Adressierung. Zwei/Vierbytebefehl
    In den CPU–Registern und über (HL) und die Index–Register lassen sich einzelne Bits direkt beeinflussen.

Speicheradressierung für 64 K Byte

	16 Bit Adressen Dezimal	Hex	8 Bit Daten
64 K	65535	FFFF	
	61440	F000	
	61439	EFFF	
	57344	E000	
	57343	DFFF	
	53248	D000	
	53247	CFFF	
48 K	49152	C000	
	49151	BFFF	
	45056	B000	
	45055	AFFF	
	40960	A000	
	40959	9FFF	
	36864	9000	
	36863	8FFF	
32 K	32768	8000	
	32767	7FFF	
	28672	7000	
	28671	6FFF	
	24576	6000	
	24575	5FFF	
	20480	5000	
	20479	4FFF	
16 K	16384	4000	
	16383	3FFF	
	12288	3000	
	12287	2FFF	
8 K	8192	2000	
	8191	1FFF	
4 K	4096	1000	
	4095	0FFF	
	0000	0000	

2. Test – Lösungsblatt

2.1 Es sind die ersten beiden Programmzeilen in Programm 10 auf Seite 114. Nach dem ursprünglichen Bild 6 – 6 käme dieses Programm nie aus der äußeren Schleife heraus.

2.2 Die Programme werden frei verschiebbar.

2.3 Der Sprung führt auf die Adresse 0 1 0 D.

2.4 Durch diese Befehlsfolge wird der Inhalt des BC–Registerpaares in das DE–Registerpaar geladen.

2. Test

Name und Rundfunkanstalt

2.1 Welche Programmzeilen werden durch den Fehler in Bild 6 – 6 betroffen?

2.2 Welchen Vorteil bringt eine relocative Programmierung?

2.3 Auf welche Adresse führt folgender Sprungbefehl?

Adresse	OP-Code	Mnemonic
0110	18	JR,d
0111	FB	"

2.4 Was bewirkt folgende Befehlsfolge in einem Programm?

```
PUSH BC
POP  DE
```

Register BC wird über SP in Register DE verschoben.
Push BC → SP, Pop SP → DE

richtig

3. Teil

Bearbeitungszeitraum:
August, September 1981

Inhalt

	Seite
Übersicht	3 - 2
Kapitel 8, Anleitung	3 - 3
Die Flags als Bitmuster	3 - 5
Programmbeispiel 3.1	3 - 6
Unterprogrammtechnik	3 - 7
Programmbeispiel 3.2	3 - 9
Programmbeispiel 3.3	3 - 14
3. Test	3 - 20

3. Teil - Übersicht

Sie lernen im dritten Teil zunächst noch einige spezielle Sprungbefehle kennen. Dann werden wir uns ausführlich mit der Handhabung von Unterprogrammen beschäftigen. Die Unterprogrammtechnik eröffnet uns den Zugang zum Nanocomputer-Betriebssystem, wir werden Teile daraus in unseren Programmbeispielen verwenden. Sie beginnen im Lehrbuch Band I mit dem Kapitel 8, später kommen Abschnitte aus dem Band III dazu. Die Versuche werden Sie weitgehend mit Beispielen aus der Arbeitsmappe durchführen. Sprungmöglichkeiten sind nicht eingeplant.

Lesen Sie zunächst einmal im Kapitel 8 bis zur Seite 169 unten. Sie finden dort Erklärungen zur Funktion des Programmzählers bei Sprungbefehlen. Die Beispiele im Text liegen außerhalb unseres RAM–Bereiches, Sie können sie deshalb mit dem Übungsgerät nicht nachvollziehen.

Auf S. 169 unten beginnt ein Abschnitt über die Flags und die bedingten Sprünge. Hierzu folgende Ergänzung:

Flags sind die Zustandssignale der CPU. Über die Flags erfahren Sie, welches Ergebnis eine gerade abgeschlossene Operation erzielt hat. Sie bekommen die Möglichkeit, weitere Schritte von diesem Ergebnis abhängig zu machen. Bildlich gesprochen, erlauben die Flags, Weichen im Programmablauf zu stellen. Es werden Programmverzweigungen in Abhängigkeit von Zwischenergebnissen ermöglicht.

Flags verhalten sich wie Flip-Flops. Einmal gesetzt, behalten sie ihren Wert (0 oder 1) bei, bis zu einem Reset des Computers oder bis sie eine andere Operation gegenteilig beeinflußt. Im Z–80 stehen für jeden Registersatz sechs Flags zur Verfügung. Vier davon werden normalerweise in Programmen als Entscheidungskriterium verwendet, die übrigen erfüllen Sonderaufgaben.

Einzelne Flags besitzen unterschiedliche Bedeutung, abhängig davon, welcher Befehl gerade bearbeitet wird. Hauptsächlich arbeitet man aber mit drei eindeutig bestimmten Flags:

C-Flag:	Carry	=	Übertrag
Z-Flag:	Zero	=	Null
S-Flag:	Sign	=	Vorzeichen

Die Flags des Z–80 sind reine Software–Steuersignale, keine Steuerspannungen für die Hardware. Das Betriebssystem des Nanocomputers erlaubt uns jedoch, die Flags anzuschauen. Die Flags, eine Gruppe von untereinander unabhängigen Einzelbits, wird hierbei als geschlossenes Byte in einer zweistelligen Hexziffer präsentiert. Zur Beurteilung einzelner Flags muß die Hexziffer in die 8–Bit Binärstellung umgewandelt werden. Die Stellung der Flags innerhalb des 8–Bit Rahmens gibt Ihnen die Darstellung auf Seite 3 – 5.

Lesen Sie nun erst einmal im Buch die Seiten 169 bis 172. Die Seiten 173 und 174 lassen Sie aus, ein Beispiel finden Sie hier in der Arbeitsmappe auf Seite 3 – 6.

Die FLAGS

S	Z	X	H	X	P/V	N	C

S = 1: Bit 7 = 1

Z = 1: Ergebnis = 0

X = keine Bedeutung

H = 1: Übertrag v. Bit 3 n. 4

P = 1: gerade Parität

V = 1: Übergang bei 0

N = 1: Subtraktion

C = 1: Übertrag v. Bit 7

Arbeitsweise der Flags, Programmbeispiel 3.1

Adresse	Op-Code	Label	Mnemonic
0100	06 02	ZEIT	LD B,02
0102	0E 02	SCHL.2	LD C,02
0104	0D	SCHL.1	DEC C
0105	20 FD		JR NZ,-3
0107	05		DEC B
0108	20 F8		JR NZ,-8
010A	FF		RST 38

Dieses Programm entspricht der auf den Seiten 114 bis 119 besprochenen Zeitschleife.

Entscheidend ist für uns hier die Beobachtung der Flags. Bitte laden Sie dieses Programm und führen Sie es im Einzelschritt aus. Notieren Sie dabei für jeden Schritt die Anzeige der Flags (AF anwählen). Setzen Sie anschließend die Hexzahlen in Bitmuster um und verfolgen Sie die Veränderungen des Zero-Flags während des Programmablaufs. An diesem Beispiel wird die Beeinflussung des Zero-Flags durch den gerade eben bearbeiteten Decrement-Befehl besonders deutlich. Es zeigt sich auch, daß Ladebefehle keine Auswirkungen auf Flags haben.

Unterprogrammtechnik

Unterprogramme sind abgeschlossene Programmteile, die Teilaufgaben in größeren Programmiervorhaben erfüllen. Sie können von einem beliebigen Hauptprogramm aus angerufen werden. Am Ende des Unterprogramms erfolgt automatisch ein Rücksprung in das Hauptprogramm.

Die Verwendung von Unterprogrammen bringt für den Programmierer folgende Vorteile:

1. Mehrfach verwendete Programmteile belegen als Unterprogramm häufig weniger Speicherraum.
2. Unterprogramme erlauben eine übersichtliche Programmstruktur. Eine Modultechnik ist möglich, sofern genügend lauffähige Unterprogramme vorhanden sind.
3. Fremde Software, die in Form von Unterprogrammen vorliegt, kann in eigenen Programmen leicht verwendet werden. (Programmbibliotheken)

Diese Vorteile überzeugen, es ist heute unmöglich, ohne Unterprogramme sinnvoll zu programmieren. Besonders rückt Punkt 2 immer stärker in den Vordergrund. Speicherplätze werden durch den Fortschritt der Halbleitertechnik immer billiger, dagegen wird es immer schwieriger sich in unübersichtlichen, unstrukturierten Programmen zurechtzufinden. Ein besonderes Anliegen dieses Kurses ist es deshalb, Ihnen die Regeln zur Verwendung von Unterprogrammen zu vermitteln.

Die speziellen Befehle, die ein Unterprogramm möglich machen, werden Ihnen im Lehrbuch von Seite 175 bis 178 vorgestellt. Lesen Sie diesen Text durch. Es wird dort viel mit dem Stack gearbeitet, ohne noch einmal ausführlich die Zusammenhänge zu erläutern. Grundsätzliches dazu können Sie in der Arbeitsmappe (Seite 2 – 10) nachlesen.

Die Versuche zum 8. Kapitel werden wir nicht durchführen, Sie finden andere Versuche hier in der Arbeitsmappe.

Umwandlung eines Programms in ein Unterprogramm

Das Programm 3.2 hat die Aufgabe, zwei 8-Bit Zahlen zu addieren. Die beiden Summanden stehen in den RAM-Zellen S1 und S2 bereit, das Ergebnis soll in der Zelle ES erscheinen. Da diese Zellen direkt hintereinander liegen, wird die indirekte Register-Adressierung mit dem HL-Register verwendet.

Programmbeispiel 3.2: A D D I T 1

Adresse	Op-Code	Label	Mnemonic
0100	00 bis FF	S 1	
0101	00 bis FF	S 2	
0102	00 bis FF	ES	
0103	00	ADDIT 1	NOP
0104	00		NOP
0105	21 00 01		LD HL,0100
0108	7E		LD A,(HL)
0109	2C		INC L
010A	86		ADD A,(HL)
010B	2C		INC L
010C	77		LD (HL),A
010D	FF		RST 38

Das Programm beginnt mit zwei NOP-Befehlen, hiermit halten wir uns zwei Bytes für die spätere Erweiterung des Programms frei.

In Zeile 0105 wird das HL-RP für die Adressierung der Zelle S1 vorbereitet.
Der Befehl 7E lädt den ersten Summanden aus S1 in den Akku. LD A, (HL) heißt wörtlich: Lade den Inhalt der Speicherzelle, deren Adresse im HL-RP steht, in den Akku (siehe auch S. 104). Mit INC L wird nun die Zelle S2 adressiert. Jetzt kann der Inhalt des Akkus, das ist der Wert S1, mit dem Inhalt der Speicherzelle S2 (über HL-Adressierung) addiert werden. Mit INC L adressieren wir nun die Ergebniszelle ES und bringen mit dem Befehl 77 den Inhalt des Akkus dorthin. Der Schlußbefehl FF führt uns wieder in das Betriebssystem zurück. Dieses Programm zeigt Ihnen deutlich die Vorteile dieser indirekten Adressierungsart. Genausogut hätten wir über das IX oder IY Register mit Distanzangaben adressieren können. Soweit unser Programmbeispiel. Sie können es eingeben und ausprobieren, vergessen Sie nicht, vorher irgendwelche Werte für S1 und S2 zu laden.

Dieses Additionsprogramm soll nun in ein Unterprogamm umgewandelt werden:

Die Umwandlung ist für dieses Beispiel sehr einfach. Ersetzen Sie den letzten Befehl RST 38 durch einen Return-Befehl C9 – RET.
Um die Wirkung dieser Umwandlung zu studieren, fehlt uns nun ein Hauptprogramm, das dieses Unterprogramm aufruft, es könnte folgendermaßen aussehen:

Adresse	Op-Code	Label	Mnemonic
0000	CD 03 01		CALL ADDIT 1
0003	FF		RST 38

Geben Sie dieses Hauptprogramm ein und lassen Sie es im Einzelschritt ablaufen. Beobachten Sie dabei den PC. Vom Hauptprogramm wird in das Unterprogramm gesprungen, dort die Addition ausgeführt und in das Hauptprogramm zurückgekehrt.

Von anderen Programmen aus könnte nach diesem Verfahren ebenfalls eine Addition durchgeführt werden. Es muß nur dafür gesorgt werden, daß die neuen Summanden in den Zellen S1 und S2 bereitstehen.
Die eindeutige Festlegung der drei Zellen S1, S2 und ES erlaubt uns, von beliebiger Stelle aus Daten an das Unterprogramm A D D I T 1 zu übergeben oder, was für das Ergebnis der Addition in ES gilt, vom Unterprogramm an das Hauptprogramm weiterzureichen. Eine Datenübergabe dieser Art bezeichnet man als "Parameterübergabe in definierten RAM-Zellen".

Es ist dies eine der sichersten und im Sinne einer übersichtlichen Programmierung, eine der "saubersten" Formen der Datenübergabe in Programmen. Andere Verfahren werden Sie noch kennenlernen. Unser Unterprogramm A D D I T 1 besitzt noch einen Schönheitsfehler. Nach Rückkehr in das Hauptprogramm ist das HL – RP mit 0102 und der Akku mit dem Wert aus ES geladen. Ein wirklich gutes Unterprogramm darf die CPU–Register, die eventuell im Hauptprogramm mit Daten belegt sind, in keiner Weise verändern, nur dann ist ein Unterprogramm wirklich universell einsetzbar. Wir sorgen nun dafür, daß die Inhalte der Register A, H, L während des Ablaufs unseres Unterprogramms auf dem Stack zwischengespeichert (gerettet) und nachher wieder eingesetzt werden. Hierzu dienen die PUSH– und POP–Befehle.

Das Unterprogramm bekommt nun seine endgültige Form. Zur besseren Unterscheidung erhält es auch einen neuen Namen: A D D I T 2. Im Hauptprogramm ändert sich, bis auf diesen Namen, nichts.

Adresse	Op-Code	Label	Mnemonic
0100	00 bis FF	S 1	
0101	00 bis FF	S 2	
0102	00 bis FF	ES	
	.		
0103	F5	ADDIT 2	PUSH AF
0104	E5		PUSH HL
0105	21 00 01		LD HL,0100
0108	7E		LD A,(HL)
0109	2C		INC L
010A	86		ADD A,(HL)
010B	2C		INC L
010C	77		LD (HL),A
010D	E1		POP HL
010E	F1		POP AF
010F	C9		RET

Bitte überprüfen Sie dieses Programm, stellen Sie fest, ob es in der gewünschten Weise funktioniert. Im nächsten Schritt werden wir versuchen ein Unterprogramm zu erstellen, das uns erlaubt, die Summanden S1 und S2 über die Tastatur direkt in die RAM-Zellen einzugeben.

Programmbeispiel 3.3 Tasteneingabe

Die Summanden haben Sie bisher recht umständlich über MEM und ST eingeben müssen. Nun soll die Eingabe über die Tastatur direkt in das Programm erfolgen. Das Betriebssystem des Nanocomputers enthält ein Unterprogramm K B S C A N, das die Tastatur abliest. Dieses Programm ist in Band III ab S. 223 und in der Software-Beschreibung DN 314 ab Seite 3 beschrieben. (DN 314 finden Sie im Technical Manual des Nanocomputers.) Bitte lesen Sie sich diese Beschreibungen durch, damit Sie ungefähr wissen, wie eine Tastatur-Eingabe verarbeitet wird.

Folgende Punkte sind für die Benutzung des Unterprogramms K B S C A N von Bedeutung:

1. Es werden die CPU-Register AF, BC verwendet, deren ursprüngliche Inhalte gehen verloren.
2. Die Parameterübergabe (Tastenwert) erfolgt über die CPU-Register A und C.
3. Die Mitteilung "Taste gedrückt – Taste nicht gedrückt" gibt K B S C A N über das Carry-Flag ab.

K B S C A N prüft die Tastatur bei jedem Anruf nur ein einziges Mal innerhalb weniger Millisekunden. Es ist also vom Anwender eine Programmschleife aufzubauen, die das Unterprogramm wiederholt aufruft.

Vorversuch: Programmschleife

"Abwarten auf Tastendruck"

Adresse	Op-Code	Label	Mnemonic
0200	CD DB F8		CALL KBSCAN
0203	30 FB		JR NC,-5
0205	CD DB F8		CALL KBSCAN
0208	38 FB		JR C,-5
020A	FF		RST 38

Hier sehen Sie, wie die Abfrageschleifen für die Tastatur aufgebaut sind. Da auch dieses Programm mit der GO-Taste gestartet wird, warten wir zunächst ab, daß diese Taste losgelassen wird. Die ersten beiden Programmzeilen bilden eine Schleife, in der das Unterprogramm K B S C A N solange aufgerufen wird, bis die GO-Taste nicht mehr gedrückt ist. Diese Entscheidung führen wir über den bedingten, relativen Sprungbefehl

JR NC,-5

durch. Solange das Carry-Flag den Wert 0 besitzt (C = 0 d. h. Taste gedrückt), springt das Programm um 5 Adressen zurück und ruft K B S C A N erneut an. Ist die Taste losgelassen, wird Zeile 0205 ausgeführt. Das Programm läuft in eine neue Schleife, in der abgewartet wird, daß endlich jemand eine Taste drückt. Sie erkennen das daran, daß in Zeile 0208 der Befehl

JR C,-5

steht. Übersetzt heißt dieser Befehl:
Springe relativ um 5 Adressen zurück, wenn das Carry-Flag den Wert 1 besitzt. Wird jetzt eine Taste gedrückt, verläßt das Programm die Schleife und führt uns zurück in das Betriebssystem. Den Tastenwert finden Sie in A oder C. Bitte geben Sie dieses Programm ein und überzeugen Sie sich von dessen Funktion. Überprüfen Sie die Tabelle in DN 314, Seite 4. Es ist dort ein Druckfehler enthalten, den Sie ausbessern sollten.

Im nächsten Schritt werden wir nun die Tasteneingabe für unser Additionsprogramm erarbeiten.

Da wir für die Addition zwei Zahlenwerte brauchen, sollten wir die Tasteneingabe auch wieder in Form eines Unterprogramms schreiben. Wenn dieses Unterprogramm mehrfach verwendet werden soll, darf es nicht die Abwarteschleife für das Loslassen der GO-Taste enthalten. Durch die Vorgabe von K B S C A N sind wir auf die Parameterübergabe über CPU-Register festgelegt. S1 und S2, die Zellen, die die Tastenwerte aufnehmen, adressieren wir diesmal über das DE-RP.

Adresse	Op-Code	Label	Mnemonic
0120	CD DB F8	TASTE 1	CALL KBSCAN
0123	38 FB		JR C,-5
0125	12		LD (DE),A
0126	CD DB F8		CALL KBSCAN
0129	30 FB		JR NC,-5
012B	C9		RET

Dieses Unterprogramm "TASTE 1" wartet auf einen Tastendruck, bringt dann den Tastenwert in die Speicherzelle, deren Adresse im DE-RP steht und wartet anschließend darauf, daß die Taste wieder losgelassen wird. Es verwendet AF, BC, die Datenadresse muß in DE stehen. Darauf sollte ein noch zu erstellendes Hauptprogramm Rücksicht nehmen.

Fassen wir noch einmal zusammen:

Es sind zwei Zahlen zu addieren, die in RAM-Zellen bereitstehen.

Diese Zahlen sollen durch die Tastatur eingegeben werden.

Zwei Unterprogramme stehen bereits fest: ADDIT 2, TASTE 1. Bleibt nur noch die Aufgabe, diese Unterprogramme sinnvoll zu kombinieren.

Ablauf des Hauptprogramms:

1. Loslassen der GO-Taste abwarten
2. DE-RP mit Adresse S1 laden
3. TASTE 1 aufrufen
4. S2 adressieren
5. TASTE 1 aufrufen
6. ADDIT 2 aufrufen
7. Rückkehr in das Betriebssystem

Programmname: HAUPT 1

Adresse	Op-Code	Label	Mnemonic
0000	CD DB F8	HAUPT 1	CALL KBSCAN
0003	30 FB		JR NC,-5
0005	11 00 01		LD DE,0100
0008	CD 20 01		CALL TASTE 1
000B	1C		INC E
000C	CD 20 01		CALL TASTE 1
000F	CD 03 01		CALL ADDIT 2
0012	FF		RST 38

Dieses Hauptprogramm sollten Sie nun ausprobieren. Überzeugen Sie sich, ob tatsächlich die gewünschten Zahlen addiert werden.

Dabei wird Ihnen auffallen, daß die Zahlen, die die Tastatur abgibt, 4 Bit–Zahlen mit führender Null sind. Hier ist unser schönes Programm irgendwie noch nicht vollkommen oder ausbaufähig. Wir werden im 4. Teil noch einmal darauf zurückkommen und gemeinsam die Eingabe auf 2–stellige Hexziffern erweitern. Ebenso werden wir versuchen, auch die Anzeige des Nanocomputers für unser Additionsvorhaben zu verwenden.

3. Test

Schreiben Sie ein Programm, mit dem Sie 4 Zahlenwerte über die Tastatur in die RAM-Zellen 0300 bis 0303 eingeben können. Als Zahlenwerte gelten die 4-Bit Zahlen mit führender Null (00 bis 0F), die KBSCAN abgibt. Die Wahl der Adressen für Ihr Programm steht Ihnen frei.

Versuchen Sie, das viermalige Eingeben in Form einer Programmschleife über das Herunterzählen eines Schleifenzählers und bedingten Sprung auszuführen. Ihr Programm sollte mit dem RST 38-Befehl schließen. An eine Addition der Zahlenwerte ist nicht gedacht.

Verwenden Sie beiliegende Programmformulare. Bitte geben Sie Namen und Rundfunkanstalt an, um Verwechselungen auszuschließen.

4. Teil

Bearbeitungszeitraum:
Oktober, November 1981

Inhalt

	Seite
Übersicht	4 - 2
Kapitel 9 und 10, Anleitung	4 - 3
Zusammenfassung der neuen Befehle	4 - 4
Ein- und Ausgänge für Mikroprozessoren	4 - 6
Die Z-80-PIO	4 - 9
Versuche zu den Kapiteln 9 und 10	4 - 15
Die Verwendung der Nanocomputer-Anzeige	4 - 20
Erweiterung des Additionsprogramms	4 - 27
4. Test	4 - 29

4. Teil - Übersicht

Sie lernen wieder weitere Befehle kennen. Befehle, die eine Verbindung herstellen zwischen der digitalen Schaltungstechnik und dem Mikroprozessor. Mit einem weiteren Abschnitt über Ein- und Ausgänge in Mikroprozessoren werden Sie dann auf dem Experimentierfeld erste Versuche durchführen und erfahren, wie Sie Steuersignale erzeugen oder verarbeiten können.

Die angekündigte Erweiterung des Additionsprogramms aus dem Teil 3 zeigt Ihnen Möglichkeiten zur Verwendung der Nanocomputer–Anzeige. Sie erhalten damit weitere Beispiele für den Einbau von Teilen des Betriebssystems in eigene Programme.

Lesen Sie bitte im Kapitel 9 die Seiten 196 bis 205. Die dort beschriebenen Logik-Befehle sind Ihnen aus der Digitaltechnik geläufig. Neu daran ist nur die parallele Verarbeitung aller 8 Bits eines Datenwortes. Der Mikroprozessor benimmt sich sehr eigenartig. Betrachten Sie den Akku: Vor der Operation muß er mit einer der "Eingangsvariablen" geladen sein, nachher enthält er das Ergebnis der Verknüpfung. Ein Ergebnis, das aus acht einzelnen, bitweise durchgeführten Logikverknüpfungen hervorgeht. Im Gegensatz dazu wirkt das Zero–Flag wie der Ausgang eines dem Akku nachgeschalteten NAND–Gatters, das alle 8 Bit gleichzeitig miteinander verknüpft.

Gehen Sie von der Seite 205 aus gleich zum 10. Kapitel weiter. Lesen Sie bitte die Seiten 216 bis 228. Dort sind weitere Befehle recht anschaulich erläutert. Die Fortsetzung finden Sie hier in der Arbeitsmappe.

1. Die Gruppe der Logikbefehle

Logische Verknüpfungen führt der Mikroprozessor Bit für Bit durch. Der Akku besitzt eine Sonderfunktion. Vor der Operation muß er mit einer Eingangsvariablen geladen sein. Diese Variable geht verloren, der Akku enthält anschließend das Ergebnis der Verknüpfung. Die zweite Eingangsvariable kann in einem der CPU-Register stehen; mit (HL), (IX + dd), (IY + dd) aus einer RAM- oder ROM-Zelle stammen oder direkt angegeben werden. Folgende Operationen sind möglich:

AND - UND-Verknüpfung
OR - ODER-Verknüpfung
XOR - Exclusiv-ODER
NEG - Zweierkomplement des Akkuinhalts
CPL - Bitweises Invertieren des Akkuinhalts

Die Logikbefehle verwenden wir selten im eigentlichen Sinne der Digitaltechnik.
AND dient zum Maskieren (Ausblenden) bestimmter Bits aus einem 8-Bit Datenwort.
Mit dem Befehl AND A, einer UND-Verknüpfung des Akkuinhalts mit sich selbst, können Sie die Flags nach Ladebefehlen setzen, ohne den Akkuinhalt zu verändern.
OR dient zum Mischen oder Zusammensetzen von zwei Bitmustern. Bei 16-Bit Decrementbefehlen erlaubt der OR-Befehl eine Überprüfung auf das Ergebnis Null.

XOR ist im Hilfsmittel, nach Unterschieden in zwei Datenworten zu suchen, Auswertung über das Zero–Flag.
NEG und CPL Diese Befehle besitzen leider irreführende Mnemonics. Die Negation der Digitaltechnik (Umkehr der Logikpegel 1 in 0 und 0 in 1) heißt CPL. Der Befehl NEG dagegen gehört nicht zu den Logikoperationen, er bildet das Zweierkomplement des Akkuinhalts!

2. Die Einzelbit–Operationen

In den CPU–Registern und im Speicherraum über die indirekte Adressierung, können Sie einzelne Bits setzen, rücksetzen oder ohne Beeinflussung prüfen. Die Bits werden von 0 bis 7 gezählt, von rechts beginnend (0 bis 7! Eine beliebte Fehlerquelle).
Diese Befehle ersetzen oft umständliche Maskierungen, sie sind außerordentlich nützlich.

3. Rotate– und Shift–Befehle

Diese Befehle ersetzen die Schieberegister der Digitaltechnik. Die einzelnen Varianten sind leicht auseinanderzuhalten. SHIFT–Befehle entsprechen genau dem klassischen Schieberegister, die Befehle unterscheiden sich bezüglich der Schieberichtung und der Einbeziehung des Carry–Bits.

ROTATE–Befehle bewirken unterschiedliche Bitmuster, sie sind dem zyklischen Speicher, der aus einem rückgekoppelten Schieberegister entsteht, ähnlich.
Shift– und Rotate–Befehle lassen sich am leichtesten anhand der bildlichen Darstellungen der Bitverschiebung auseinanderhalten.

Eingänge und Ausgänge für Mikroprozessoren, die I/O - Technik

Vorbemerkungen

Mikroprozessoren besitzen keine Eingänge und Ausgänge im üblichen Sinne. Die Bus-Leitungen der CPU, auf denen der gesamte Datenverkehr des Mikroprozessors mit seinen angeschlossenen Systembausteinen stattfindet, verlangen besondere Schaltungen für den Kontakt mit der Aussenwelt, sogenannte Ports. Ports beobachten den Adreß-, Steuer- und Datenbus. Sie werden aktiviert, sobald sie ihre Adresse zusammen mit einem I/O - Steuersignal erkennen.
Eingangsports legen dabei die ankommenden Daten kurzzeitig auf den Datenbus; es werden hierzu sogenannte Three-State-Puffer verwendet. Diese speziellen Bausteine sind im Lehrbuch Band III, Seite 131 bis 134 erklärt.
Ausgangsports übernehmen Daten vom Datenbus der CPU und speichern sie für die Weiterverarbeitung ausserhalb des Mikroprozessors. Sie formen die impulsförmigen Bussignale in statische Logiksignale um. Hierfür werden beispielsweise getaktete D-Flip-Flops verwendet.

Memory - mapped - I/O

Die vorher beschriebenen Ports können einfach an freien Plätzen innerhalb des 64 KByte-Adreßraumes der CPU angeordnet sein. Der Hersteller des Systems baut anstelle der RAM- oder ROM-Zellen dort Input- oder Output-Ports ein. Die CPU behandelt solche Ports wie Speicherzellen. Daten werden wie üblich über Ladebefehle aus den Adressen, die den Eingangsports zugeordnet sind, in das Mikroprozessorsystem eingelesen (z.B. LD A, NN). Ebenso wird über Ladebefehle an solche Adressen ausgegeben, die mit den Ausgangsports belegt sind (z.B. LD NN,A). Der Prozessor kann solche Eingangs- und Ausgangsports selbst nicht auseinanderhalten, für die CPU handelt es sich um gewöhnliche Speicherzellen, dies ist allein Aufgabe des Programmierers.
Ports dieser Art werden mit "Memory - Mapped - I/O" bezeichnet. Der deutsche Ausdruck lautet "Speicherplan-I/O". Man spricht auch von "transparenten RAM-Zellen".
Ports dieser Art sind grundsätzlich bei jedem Mikroprozessorsystem möglich. Sie sind jederzeit erweiterungsfähig und nachrüstbar, solange der Adreßraum nicht vollständig belegt ist. Bei kleinen Systemen erlauben sie

sparsame Hardwarekonzepte, dadurch daß einzelne Adreßleitungen direkt, ohne Verwendung von Adreßdekodern, zum Aktivieren der Ports benutzt werden.

Mikroprozessoren mit speziellen Input - Output - Befehlen

Moderne Mikroprozessoren bieten zusätzlich zur Memory-Mapped-I/O-Technik besondere Input-Output-Befehle, die für den Benutzer einige Vorteile bringen:
Einmal reduzieren sie den Hardware-Aufwand. Es ergeben sich einfachere Schaltungen für die Steuerung der Ports. Der größere Vorteil liegt aber im Software-Bereich. Besondere I/O-Befehle sorgen dafür, daß bei der Programmierung eine Trennung zwischen Speicheroperationen und Ein- und Ausgaben möglich wird. Die Programme werden dadurch übersichtlicher und von Geräten unabhängig, schließlich soll die Software vielseitig verwendbar werden.

Die Input / Output - Befehle der Z - 80
(siehe auch Band III ab Seite 57)

Im Befehlssatz des Z - 80 finden Sie eine Reihe von Input/Output - Befehlen, die von der Speicheradressierung unabhängig sind. Der Z - 80 erzeugt spezielle Steuersignale für Ports, verbunden mit einer 8 - Bit Portadresse auch Geräteadresse genannt. Es sind damit 256 Ports ansprechbar. Jede Portadresse kann mit Input- und Output-Port doppelt belegt werden, die Auswahl erfolgt über die Steuersignale. Die einfachsten I/O-Befehle erlauben den Datenaustausch zwischen dem Akku und einem Port.
Die Befehle lauten:

IN A,N = DB, N Input vom Port N zum Akku
OUT N,A = D3, N Output vom Akku zum Port N

Es sind Zwei-Byte-Befehle mit direkter Adressierung des Ports.
Bei beiden Befehlen ist der Akku Zwischenstation für die Daten. Dies kann zu Verzögerungen im Datenfluß führen, der Akku wird zum "Flaschenhals" des Systems.

Andere I/O - Befehle des Z - 80 erlauben Dateneingabe- oder Ausgabe mit den CPU - Registern. Dabei muß das C - Register die Portadresse angeben. (Vergleichbar der indirekten Registeradressierung.)

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

Darüber hinaus kann der Z - 80 noch Blockeingaben und -Ausgaben ausführen. Diese Befehle sind den Block - Transfer - Befehlen (Band I, Seite 148) sehr ähnlich, sie sind in Band III ab Seite 59 beschrieben.

Die Z - 80 PIO (siehe auch Band III, Seite 306)

Die festverdrahteten Ports mit fest zugeordneten 8 - Bit Portadressen finden Sie überall in der Mikroprozessortechnik. Im Zuge der Weiterentwicklung der IC - Technik entstanden darüber hinaus spezielle Portbausteine, die genau auf einen Mikroprozessortyp zugeschnitten sind. Sie bieten viele zusätzliche Möglichkeiten gegenüber den einfachen Ports, sind aber oft aufgrund ihres komplexen Innenlebens schwer zu verstehen. Hierzu gehört auch die Z - 80 PIO, ein programmierbarer I/O - Baustein mit 2 x 8 Bit Datenleitungen. Sie haben richtig gelesen, die PIO ist programmierbar! Das Programm bestimmt, ob eine der 16 Datenleitungen gerade Eingang oder Ausgang sein soll. Ein 8 - Bit Port der PIO muß nicht zwangsläufig Eingang- oder Ausgangsport sein, innerhalb der 8 - Bit lassen sich einzelne Leitungen beliebig zuordnen. Weiterhin läßt die Z - 80 PIO sehr schöne Interrupt - Lösungen zu. Doch gehen wir systematisch vor:

Die Z - 80 PIO besteht aus zwei Teilen, die bis auf Kleinigkeiten völlig gleich sind. Jede Hälfte stellt ein 8 - Bit Port dar. Ein PIO - Baustein belegt deshalb zunächst zwei Portadressen. Beide Ports der PIO sind unabhängig voneinander programmierbar. Deshalb benötigt eine PIO ausser den Portadressen auch noch zwei Adressen, die den Steuerteilen der PIO - Hälften zugeordnet sind. Das liest sich zunächst etwas kompliziert, gehen wir deshalb zu einem praktischen Beispiel für den Nanocomputer über.

Vorarbeiten

Unterhalb des Experimentierfeldes Ihres Nanocomputers finden Sie drei 40 - polige IC - Fassungen. Wir verabreden, die linke Fassung FL, die mittlere FM und die rechte FR zu nennen. An FL liegen in der unteren Reihe die PIO - Anschlüsse. Die eine PIO - Hälfte wird PIO C genannt, die entsprechenden Datenleitungen PC 0 bis PC 7. Die zweite PIO - Hälfte ist PIO D mit PD 0 bis PD 7.

Wir wollen nun die PIO C als Ausgangsport betreiben. Mit den Ausgangsdaten steuern wir die Leuchtdioden LM 0 bis LM 7 an. Schalten Sie den Nanocomputer jetzt erst einmal aus und suchen Sie das Plastic - Kästchen mit den Drähten. Verbinden Sie schön der Reihe nach alle PC - Anschlüsse von FL (Fassung links) mit den LM - Anschlüssen von FR. Wenn Sie den Nanocomputer nun wieder einschalten, bleiben die Anzeigedioden dunkel. Die PIO wird beim Einschalten der Betriebsspannung zurückgesetzt, sie geht in eine definierte Anfangsstellung, bei der die Datenleitungen den hochohmigen Zustand annehmen (weitere Details: Band III, S. 313).

Programmierung der PIO

Die PIO C ist als Ausgangsport zu programmieren. Der entsprechende Befehl lautet OF.
Diesen Befehl gilt es nun,per Programm an den Steuereingang der PIO C mit der Portadresse OA zu senden. Hier das erforderliche Programm:

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
00 00	3 E OF	PIO	LD A, OF	Steuerwort"Ausgang" zum Akku
02	D 3 OA		OUT OA, A	OUTPUT:Akkuinhalt zum Steuerpor
05	FF		RST 38	

Mit dieser Prozedur programmieren Sie die PIO C als Ausgangsport. Die Übertragung des Steuerwortes vom Akku in die Steuerung der PIO erledigt hier der OUT - Befehl an die Portadresse der PIO - Steuerung. Geben Sie dieses Programm ein und lassen Sie es dann ablaufen. Damit ist die PIO zum Ausgangsport geworden. Sie bleibt solange Ausgang, bis die Steuerung ein anderes Steuerwort empfängt oder die Betriebsspannung abgeschaltet wird, manchmal gelingt es auch die PIO mit der Reset - Taste zurückzusetzen.

Ausgabe von Daten

Überprüfen Sie nun die Funktion des Ports. Laden Sie das folgende Programm und lassen Sie es im Einzelschritt - Betrieb ablaufen.

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
01 00	3 E FF		LD A, FF	Anfangsbedingung —— Lampen ei
01 02	D 3 08		OUT 08, A	Ausgabe an Datenport PIO C
01 04	3 D	SCHLEIFE	DEC A	AKKU - 1
01 06	D 3 08		OUT 08, A	Ausgabe an Datenport PIO C
01 07	18 FB		JR - 5	Sprung zu SCHLEIFE

Ihre Lampen müssen das Verhalten eines binären Rückwärtszählers zeigen. Falls Sie das Programm mit GO starten, leuchten Ihre Lampen gleichmäßig, da die Augen der hohen Zählgeschwindigkeit nicht folgen können.

Deshalb sollten wir in dieses Programm eine Zeitverzögerung einarbeiten! Geeignet ist die Subroutine"Universelle Zeitschleife", die Sie mit dieser Lieferung als Programmbeispiel bekommen haben. Bitte laden Sie diese Zeitschleife ab Adresse 0200. Für die Versuche zu den Kapiteln 9 und 10 wird die Zeitschleife ebenfalls benötigt. Wir legen die Zeitverzögerung in die Zählschleife, direkt vor den Rücksprung. Da es sich um ein Unterprogramm handelt, das keine CPU - Register verändert, entstehen keine neuen Probleme.

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
00 00	3 E OF	PIO C	LD A, OF	Steuerwort Ausgang zum Akku
02	D 3 OA		OUT OA, A	Ausgabe zum Steuerport PIO C
04	C 3 00 01		JP 0100	
01 00	3 E FF		LD A, FF	Anfangsbedingung
02	D 3 08		OUT 08, A	Ausgabe Datenport PIO C
04	3 D	SCHLEIFE	DEC A	AKKU - 1
05	00 00 00		NOP (3x)	freihalten für spätere Änderunge
08	D 3 08		OUT 08, A	Ausgabe
OA	CD 00 02		CALL ZEIT	Zeitverzögerung
OD	18 F5		JR - 11	Sprung: SCHLEIFE

Starten Sie dieses Programm mit GO. Überzeugen Sie sich, daß die gewünschten Funktionen ablaufen. Folgende Variationsmöglichkeiten bieten sich an:

1. Statt DEC A = 3 D, ein INC A, ergibt einen Vorwärtszähler.
2. Die Anfangsbedingung für den Vorwärtszähler in 00 abändern.
3. Unterschiedliche Zeitbytes in der Zeitschleife ausprobieren.

Spielen Sie diese Möglichkeiten einmal durch. Versuchen Sie auch, ein Gefühl für die Fein- und Grobabstimmung der Zeitschleife zu bekommen. Falls Ihnen ein digitaler Frequenzzähler mit der Einrichtung zur Zeitintervallmessung zur Verfügung steht, könnten Sie den Einfluß der Zeitbytes auch messen. Der Zähler ist an PC 0 anzuschließen.

Nun folgen die Übungen zu den Kapiteln 9 und 10. Teile des Zählprogramms und die Zeitschleife können Sie dabei verwenden, schalten Sie deshalb Ihren Nanocomputer noch nicht aus.

Noch etwas sehr wichtiges! Bitte vermeiden Sie Kurzschlüsse der PIO - Leitungen, besondere Vorsicht ist an der Fassung FR geboten. Dort liegen die Betriebsspannungen + 12 V, -12 V, -5 V. Diese Spannungen sind für jeden Logikschaltkreis tödlich. Überkleben Sie diese Anschlüsse mit Tesafilm oder brechen Sie die Stifte an der Zwischenfassung aus. Dies ist leicht möglich da wir auf die Fassungen des Nanocomputers noch einmal zusätzliche Fassungen gesteckt haben.

Übrigens sind die PIO - Ausgänge sehr leistungsschwach. Sie dürfen nur mit einem TTL - Eingang belastet werden, der zulässige Strom beträgt:

2,0 mA im 0 - Zustand
0,25 mA im 1 - Zustand

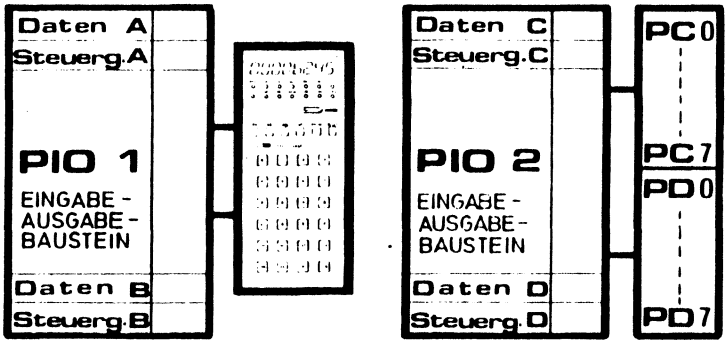
Die PIO - Bausteine sind keine Kraftmeier, ihre Vorteile liegen im Software - Bereich.

Die PIO als Eingabe - Port

Das Steuerwort 4 F programmiert die PIO zu einem Eingangsport. Wie vorher schon, muß das Steuerwort über einen OUTPUT - Befehl an das Steuerport der gewünschten PIO - Hälfte geschickt werden. Erst dann können Daten von dem entsprechenden Datenport entgegengenommen werden. Ein Programmbeispiel soll diese PIO - Betriebsart verdeutlichen. Die Adressen liegen so, daß keine Kollision mit den vorher eingegebenen Programmen auftreten können. Dieses Programm fragt die Stellungen der Schalter SW 0 bis SW 7 ab und zeigt das Ergebnis über die LED'S LM 0 bis LM 7 an. Eingabeport wird PIO D. Stellen Sie die Drahtverbindungen zwischen den Datenleitungen PD 0 bis 7 auf FL und den Schalterausgängen SW 0 bis SW 7 auf FR her, bitte achten Sie auf die richtige Reihenfolge. Die Verbindungen der PIO C mit den Lampen besteht schon vom letzten Beispiel her.

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
0300	3 E 4 F	PIO D	LD A, 4 F	Akku: Steuerwort Eingang
0302	D 3 0B		OUT 0B, A	zum Steuerport PIO D
0304	3 E 0F	PIO C	LD A, 0F	Akku: Steuerwort Ausgang
0306	D 3 0A		OUT 0A, A	zum Steuerport PIO C
0308	DB 09	EINGABE	IN A,09	Einlesen vom Datenport 09, PIO
030A	00 00		NOP	freihalten für
030C	00 00		NOP	spätere Änderungen
030E	D3 08	AUSGABE	OUT 08, A	Ausgeben an Datenport 08, PIO C
0310	18 F6		JR - 10	

Wenn dieses Programm zufriedenstellend läuft, sollten Sie Ihren Nanocomputer noch nicht ausschalten. Die Programme und Drahtverbindungen verwenden wir auch im folgenden Abschnitt.



Die PIO's

PIO 1	A	B
Datenport	04	05
Steuerport	06	07

PIO 2	C	D
Datenport	08	09
Steuerport	0A	0B

Steuerworte	
Ausgang	0F
Eingang	4F
Bidirektional	8F
Steuerbar, Interr.	CF

Versuche zu den Kapiteln 9 und 10

Wir beginnen mit den Versuchen zu den Logikbefehlen. Zunächst arbeiten wir mit dem Programm auf Seite 4-13, beginnend bei der Adresse 0300.

Die freien Adressen 03 0A bis 030 D erlauben uns eine Veränderung der Eingangssignale, bevor die Ausgabe an die Lampen erfolgt.

1. Versuch, Wirkung des CPL - Befehls.
Setzen Sie an die Adresse 030 A den Befehl 2 F = CPL, Negation des Akkuinhalts.Starten Sie das Programm. Die Zuordnung der Lampensignale zu den Schalterstellungen ist jetzt entgegengesetzt gegenüber der ursprünglichen Fassung. Dies entspricht einer bitweisen Negation des Akkuinhalts.

2. Versuch, Wirkung des NEG - Befehls.
Der Befehl NEG = ED 44 bildet das Zweierkomplement des Akkuinhalts. Setzen Sie diesen 2 - Byte - Befehl in unser Versuchsprogramm bei Adresse 030A, 030B ein. Starten Sie das Programm. Um die Wirkung dieses Befehls richtig deuten zu können, sollten wir uns an die binäre Zählweise und die Darstellung der negativen Zahlen im Mikroprozessor erinnern. Der Mikroprozessor stellt negative Zahlen als Zweierkomplement dar, er zählt "rückwärts" beginnend bei 00, FF, FE, FD usw. Genau das erledigt dieses Programm jetzt.
Die mit den Schaltern eingegebene Binärzahl wird auf den Lampen als "negative" Zahl in der 8 - Bit- Zweierkomplementform dargestellt. Sicher denken Sie jetzt an die Aufgabe 2.3. Dort entspricht der Wert FB fünf Rückwärtsschritten.
Hier noch weitere Zahlenbeispiele:

Schalter	Lampen	
00	00	es gibt keine "negative" Null
01	FF	-1 für den Mikropozessor
0A	F6	der Rücksprung von Adresse 03 10

3. Versuch, Maskierungstechnik
Stellen Sie sich vor, Sie wollten nur die Stellungen der Schalter SW 1, SW 4 und SW 5 auswerten. Es gilt nun, die restlichen Schalterinformationen auszublenden. Wir suchen eine passende Maske und führen damit eine AND - Verknüpfung mit dem Akku inhalt durch:

Schalter:	SW 7	SW 6	SW 5	SW 4	SW 3	SW 2	SW 1	SW 0
Maske :	0	0	1	1	0	0	1	0
AND :	0	0	SW 5	SW 4	0	0	SW 1	0

Das Maskenwort lautet als Hex - Zahl 32.
Fügen Sie den AND N - Befehl in unser Übungsprogramm ein:

030 A E6 32 AND 32

Die Wirkung ist sofort sichtbar, nur noch die Werte der Schalter SW 1, SW 4 und SW 5 werden ausgegeben, die übrigen Eingänge des Ports werden ignoriert. Dieses Verfahren wird bei Steuerungen angewandt, um zu bestimmten Zeiten einzel ne Schalter oder Gruppen abzufragen. Suchen Sie sich selbst eine Schalterkombination, die Sie ablesen wollen. Bilden Sie ein neues Maskenwort, das Sie dann in das Programm einfügen.

4. Versuch, die OR - Verknüpfung

Stellen Sie sich vor, Sie wollten die Lampen LM0 und LM7 ständig leuchten lassen, unabhängig von der Stellung der zugeordneten Schalter, während weiterhin alle übrigen Schalter die Lampen ohne Veränderung ansteuern. Der Mikroprozessor müßte dem Bitmuster, das die Schalter abgeben, für LM0 und LM7 eine 1 dazusetzen, auch wenn SW0 und SW7 0-Signale liefern. Die Lösung ist sehr einfach, wir verknüpfen das eingelesene Bitmuster der Schalter mit einem Zahlenwert, der für LM0 und LM7 1-Signale ergibt. Die Zahl heißt 81, der vollständige Befehl lautet OR 81 = F6 81. Setzen Sie ihn an die Adresse 0304 und überprüfen Sie die Funktionsweise. Probieren Sie auch noch andere Werte aus.

5. Versuch, die XOR-Verknüpfung

Das XOR-Versuchsprogramm ist länger als 4 Schritte, es ist deshalb in Form eines Unterprogramms geschrieben. Der Aufruf dieses Unterprogramms muß in das Versuchsprogramm eingesetzt werden:

030A CD 0004 CALL 0400

Bitte laden Sie das Unterprogramm XOR:

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
0400	AB	XOR	XOR E	Vergleich: Eingabe mit E-Reg.
01	3 E 00		LD A,00	Lampen löschen
03	20 01		JR NZ, +1	Sprung +1, wenn Werte ungleic
05	2F		CPL	Lampen einschalten
06	C9		RET	

Dieses Unterprogramm vergleicht das Bitmuster im E-Register und von den Schaltern miteinander. Sind beide gleich, werden alle Lampen eingeschaltet, bei ungleichen Zahlen bleiben die Lampen dunkel. Die Auswertung der XOR - Verknüpfung übernimmt das Zero - Flag. Die Lampen werden nur eingeschaltet, wenn Z = 1 ist. Der bei 0401 eingeschobene Ladebefehl verändert keine Flags, das Ergebnis der XOR - Verknüpfung kann auch nach dem Ladebefehl ausgewertet werden.
Laden Sie eine beliebige Zahl über die Tastatur in das E-Register. Starten Sie das Programm bei 0300. Es muß nun so etwas wie ein Zahlenlotto 1 aus 256 ablaufen. Nur bei der von Ihnen in E vorgegebenen Schalterkombination leuchten die Lampen.

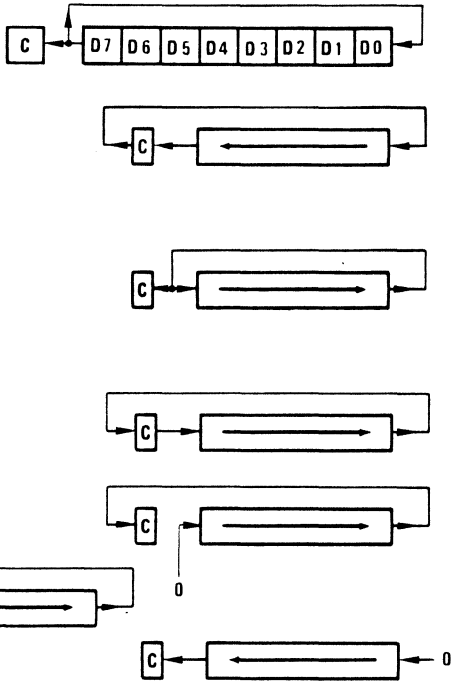
6. Versuch, Rotate - und Shift - Befehle

Die Wirkungsweise dieser Befehle können Sie gut mit unserem Übungsprogramm "Rückwärtszähler mit Zeitschleife" von Seite 4 - 11 studieren. Wir werden lediglich die Anfangsbedingung in LDA, 01 (Zeile 0100) und den Rechenbefehl in 0104 variieren. Stellen Sie die Zeitschleife auf die längste Verzögerungszeit ein. Die Lampen werden wieder vom Port der PIO C angesteuert. Hier noch einmal das Übungsprogramm, eingetragen sind bereits die Werte für den RLCA - Befehl:

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
00 00	3 E 0F	PIO C	LD A, 0F	Steuerwort Ausgang zum Akku
02	D 3 0A		OUT 0A, A	Ausgabe zum Steuerport PIO C
04	C 3 00 01		JP 0100	
01 00	3 E 01		LD A, 01	Anfangsbedingung
02	D 3 08		OUT 08, A	Ausgabe Datenport PIO C
04	07	SCHLEIFE	RLCA	AKKU 1 x links rotieren
05	00 00 00		NOP (3x)	freihalten für spätere Änderur
08	D 3 08		OUT 08, A	Ausgabe
0A	CD 00 02		CALL ZEIT	Zeitverzögerung
0D	18 F5		JR - 11	Sprung: SCHLEIFE

Starten Sie dieses Programm bei 00 00.Ein einelnes 1-Signal wird nach links durch Ihre Lampenreihe laufen. Bitte probieren Sie nacheinander auch die anderen Rotate - und Shift - Befehle aus. Die folgende Tabelle gibt Ihnen die Befehlscodes und die Anfangswerte (0101) an. Achten Sie auf die Unterschiede, die sich durch die Einbeziehung des Carry - Flag ergeben.

Befehl (0104)	Anfangswert (0101)	Funktion
RLCA = 07	01	
* RLC A = CB 07	01	
RLA = 17	01	
* RL A = CB 17	01	
RRCA = 0F	80	
* RRC A = CB 0F	80	
RRA = 1F	80	
* RR A = CB 1F	80	
SRL A = CB 3F	80	
SRA A = CB 2F	80	
SLA A = CB 27	01	



Die mit * bezeichneten Befehle entsprechen den vorangehenden. Die Doppelbelegung mit unterschiedlichen Op-Codes ergibt sich durch die Erweiterung des 8080 - Befehlssatzes im Z-80.

7. Versuch, die Einzelbit - Operationen

Das Versuchsprogramm, das die Befehle BIT, SET und RESET demonstrieren soll, finden Sie unter den DIN A 4 - Programmbeispielen, der Programmname lautet: 7. Versuch. Es verwendet die universelle Zeitschleife bei 02 00 und den Anfang des Eingabe-Ausabe-Programms. Die Drahtverbindungen PC - LM und PD - SW werden weiterhin gebraucht. Das Programm läßt, abhängig von der Stellung des Schalters SW0, zwei unterschiedliche Lauflichter über die Lampen laufen. Nach links bewegt sich ein einzelnes Licht, nach rechts eine Dunkelstelle. Die unterschiedlichen Anfangsbedingungen werden über SET- und RESET - Befehle eingestellt, in diesem Beispiel ein etwas umständlicher Weg. Jedoch erkennen Sie die Wirkungsweise der Befehle hieran recht gut. Die Schalterabfrage zeigt die Anwendung des BIT - Befehls. Die verschiedenen Bitmuster rotieren in den Registern B und C. B enthält das Muster für das Einzellicht, C für die Dunkelstelle. Dieses Programm bietet viele Variationsmöglichkeiten. Ihnen sind keine Grenzen für eigene Versuche gesetzt. Bitmuster, Rotationsrichtung und - geschwindigkeit sowie Schalterzuordnung lassen sich leicht verändern.
Nur Mut!

Die Verwendung der Nanocomputer-Anzeige

Tastatur und Anzeige des Nanocomputers sind ganz typisch für diese Art von Geräten. Sie sind im wesentlichen Software - gesteuert, die Leitungsverbindungen bestehen nur aus 2 x 8-Bit Datenleitungen der PIO A und B. Die Hardware, obwohl sie sehr interessant ist, wollen wir nicht weiter betrachten. Schaltungsdetails sind in Band III, ab Seite 207 und 223 beschrieben.

Das Nanocomputer-Betriebssystem enthält das Unterprogramm DISPL = F9 09, das die Verwendung der Anzeige ermöglicht. Die Anzeige wird bei Aufruf dieses Unterprogrammes nur einmal kurz aufleuchten. Deshalb müssen Ihre Programme, die die Anzeige verwenden, Schleifen enthalten, die DISPL wiederholt anrufen. Die Wiederholzeit sollte unter 10 ms bleiben, damit die Helligkeit ausreicht.

Anzuzeigende Daten können dem Nanocomputer-Betriebssystem direkt im Code der Siebensegment-Anzeigen oder in der üblichen Art als Hex-Ziffer angegeben werden. Für beide Möglichkeiten gibt es im oberen RAM-Bereich spezielle Speicherzellen, die bestimmten Anzeigestellen zugeordnet sind, den Display-Buffer für Siebensegment-Codeworte und den Zeichenspeicher für Hex-Ziffern. Bei Eingabe von Hex-Ziffern übernimmt ein besonderes Unterprogramm CONVDI die Umwandlung in den Siebensegment-Code. Das Ausblenden einzelner Ziffern ist in dieser Betriebsart über ein Maskenwort möglich.

Bei der direkten Ansteuerung im Siebensegment-Code ist kein Maskenwort erforderlich, die Dunkelsteuerung einer bestimmten Ziffer übernimmt das Siebensegment-Codewort 00 oder 01.

Zwei Speicherplätze innerhalb des Display-Buffers sind den Kontroll-LED's BRK bis ARS zugeordnet, sie werden genauso behandelt, wie eine Anzeigeziffer.

Versuche

1. Ansteuerung der ERR-LED

Je sieben LED's sind einem Hex-Wort innerhalb des Display-Buffers zugeordnet. Bit 0 wird bei allen Anzeigestellen nicht benutzt.

Aus Tabelle 4.1 entnehmen Sie, daß die ERR-LED mit einem 1-Signal im Bit 2 der Speicherzelle 0F B8 anzusteuern ist. Das entspricht der Hex-Ziffer 04. Diesen Zahlenwert müssen Sie nun per Programm an die Adresse 0F B8 laden und anschließend das Anzeigeunterprogramm DISPL aufrufen:

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
00 00	3E 04	ERR	LD A,04	Codewort:ERR-LED ein
02	32 B8 0F		LD 0F B8,A	Akku zum Display-Buffer LEDH
05	CD 09 F9		CALL DISPL	SR-Anzeige im Betriebssystem
08	18 FB		JR -5	Schleife für kontinuierliche Anz

Variieren Sie dieses Programm, verändern Sie das Codewort und die Adresse. Ein Ausstieg ist jederzeit durch die Break-Taste möglich. Da Sie nur LEDH verändern, bleiben die übrigen Anzeigestellen unverändert, die Anzeige zeigt: 0000 3E, zusätzlich leuchtet die ERR-LED.

Adresse: 0F B8 Label: LEDH

BIT	7	6	5	4	3	2	1	0
LED	BRK	I/O	MEM	PC	SP	ERR	ARS	-

Adresse: 0F B9 Label: LEDL

BIT	7	6	5	4	3	2	1	0
LED	IR	AF	BC	DE	HL	IX	IY	-

BIT = 1: LED leuchtet

Tabelle 4.1 Anordnung der Kontroll-LED's innerhalb der Codeworte in LEDH und LEDL
SR-DISPL: F9 09

2. Ansteuerung einer bestimmten Siebensegment-Ziffer

Die dritte Ziffer von rechts soll das Zeichen L zeigen. Das Siebensegment-Codewort zu diesem Zeichen entnehmen Sie der Tabelle 4.2. Die Segmente F, G, E, D sollen leuchten, die Bitfolge lautet demnach: 0001 1110, entsprechend 1E als Hex-Ziffer. Da Bit 0 nicht verwendet wird, könnten Sie genausogut 1F eingeben.

BIT	7	6	5	4	3	2	1	0
Segment	A	B	C	D	E	F	G	-

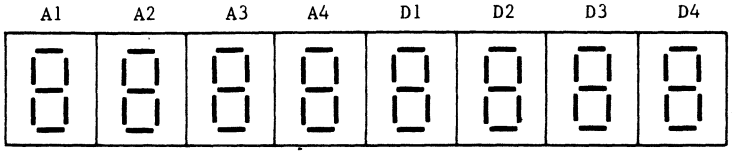
BIT = 1: Segment leuchtet

Tabelle 4.2 Stellung der Segmente einer Anzeigeziffer innerhalb des Codewortes

Tabelle 4.3 gibt die zu einer Stelle der Anzeige zugehörige Adresse im Display-Buffer an. Die dritte Stelle von rechts, D2, liegt an der Adresse 0F BF, Label: Data 7 + 1. Also muß Ihr Programm die Hex-Zahl 1E nach 0F BF laden:

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
00 00	3E 1E		LD A,1E	Siebensegment-Codewort
02	32 BF 0F		LD 0F B8,A	übertragen zum Display-Buffer
05	CD 09 F9		CALL DISPL	anzeigen
08	18 FB		JR -5	

Prinzipiell besteht also kein Unterschied zwischen der Ansteuerung einer Kontroll-LED und einer Siebensegment-Ziffer.



Label	ADD7	ADD7+1	ADD7+2	ADD7+3	DATA7	DATA7+1	DATA7+2	DATA7+3	Display –
Adresse	0FBA	0FBB	0FBC	0FBD	0FBE	0FBF	0FC0	0FC1	Buffer

SR - DISPL : F9 09

Tabelle 4.3 Siebensegment-Anzeige und zugeordnete Adressen im Display-Buffer

3. Programmbeispiel W U S E L

Das WUSEL-Programm basiert auf den vorangegangenen Beispielen 1 und 2. Der Zahlenwert im B-Register wird während eines Durchlaufes in alle Adressen des Display-Buffers geladen und angezeigt. Der Vorgang wiederholt sich, nachdem B zweimal incrementiert wurde. Der Zeittakt ergibt sich durch N-maliges Aufrufen der SR-DISPL bei jeder neuen Anzeigestelle. So werden Ihnen nacheinander alle mit einer Siebensegment-Anzeige darstellbaren Zeichen vorgeführt.

WUSEL belegt folgende Register:

AKKU: Enthält den N-Wert, der die Taktzeit bestimmt.

B: Enthält den Siebensegment-Code, der gerade angezeigt wird.

C: Wirkt als Zähler für die zu ladenden Speicheradressen des Display-Buffers, läuft daher von 0A bis 00

HL: Indirekte Adressierung der Zellen des Display-Buffers. Außerdem werden A und HL noch in DISPL verwendet.

4. Anzeige von Hex-Ziffern mit Codewandlung

Das mühsame Aufsuchen von Siebensegment-Codeworten ist für die Anzeige von Hex-Ziffern nicht nötig, da hierzu das Unterprogramm CONVDI verwendet werden kann. CONVDI liest vier 2-stellige Hex-Zahlen aus einem definierten Zeichenspeicher ab, setzt sie in acht Siebensegment-Codeworte für die acht Anzeigeziffern um und lädt die Codeworte in den Display-Buffer. Zusätzlich wird über ein spezielles Maskenwort die Dunkelsteuerung einzelner Anzeigeziffern möglich. Das Maskenwort muß vor dem Aufruf des Unterprogramms CONVDI in den Akku des alternativen Registersatzes geladen werden. Die Zusammenhänge zwischen Anzeigeziffer, Maskenwort und Adressen des Zeichenspeichers gibt Tabelle 4.4 an. Das folgende Programm HEX zeigt die Anwendung in einem Beispiel.

Die Aufgabe lautet, die Hex-Zahl 8F an den Stellen A1 und A2 anzuzeigen, alle übrigen Anzeigeziffern sollen dunkel bleiben. Sie können dieses Beispiel selbst erweitern und eigene Beispiele ausprobieren, auch sollten Sie versuchen, mehrere Anzeigeziffern anzusteuern.

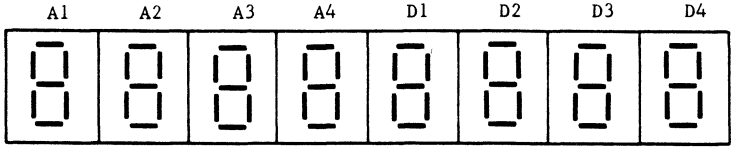
Wie das Programmbeispiel HEX zeigt, verlangt CONVDI bestimmte Vorbereitungsprozeduren:

1. Maskenwort im AKKU des ARS angeben.
2. HEX-Ziffern im Zeichenspeicher bereitstellen.
3. In DE die letzte Adresse des Zeichenspeichers laden.
4. In HL die Adresse vor dem Display-Buffer laden (LEDL)

Unter 3. teilen Sie dem Unterprogramm mit, wo die zu wandelnden Daten zu finden sind, unter 4. bestimmen Sie die Speicheradressen, die die Siebensegment-Codes aufnehmen sollen. Dadurch ist CONVDI auch an beliebigen anderen Adressen verwendbar. CONVDI und DISPL sind im Band III, Seite 231 ausführlich beschrieben.

Falls Sie vorhaben, alle 8 Ziffern anzusteuern, müssen Sie nacheinander ADDH, ADDL, DATAH, DATAL mit den gewünschten Hex-Ziffern laden. Das Maskenwort lautet dann 00. Im übrigen bleibt das Programm HEX unverändert.

Adresse	Op-Code	Label	Mnemonic	Bemerkungen
05 00	3E 8F	HEX	LD A,8F	HEX-Ziffer zum
02	32 E5 0F		LD 0F E5,A	Zahlenspeicher ADDH
05	08		EX AF,AF'	Maskenwort über Austausch-
06	3E 3F		LD A,3F	befehle zum Akku des ARS
08	08		EX AF,AF'	
09	21 B9 0F		LD HL,0F B9	HL mit Adresse LEDL laden
0C	11 E5 0F		LD DE,0F E5	DE mit Adresse ADDH laden
0F	CD 7C FA		CALL CONVDI	Umwandlung HEX-Siebensegment
12	CD 09 F9		CALL DISPL	Anzeige ansteuern
15	18 FB		JR -5	Schleife für kontinuierliche An



Siebensegment-
Anzeigefeld

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0

Maskenwort in A'(ARS)
Dunkelsteuerung von
Ziffern

BIT = 1 : Ziffer dunkel

Label	ADDH	ADDL	DATAH	DATAL
Adresse	0FE5	0FE4	0FE3	0FE2

CONVDI
Zeichenspeicher

SR - CONVDI : FA 7C

Tabelle 4.4 Adressenzuordnung im Zeichenspeicher des Unterprogramms CONVDI zur Umwandlung und Darstellung von HEX-Ziffern

Erweiterung des Additionsprogramms aus Teil 3

Das ursprüngliche Additionsprogramm wird in diesem Abschnitt auf die Verarbeitung zweistelliger Hex-Ziffern und Anzeige der Eingabe- und Ausgabewerte erweitert. Die Eingabe der vier Tastenwerte haben Sie bereits im 3. Test bearbeitet, lediglich der Schlußbefehl FF muß in C9 = RET abgeändert werden und das Programm ist als Unterprogramm für unseren Zweck hier verwendbar. Die Adressen behalten wir bei. In die Zelle 03 04, die auf die Tastenwert-Speicher folgt, laden wir später das Ergebnis der Rechnung.
Ein weiteres Unterprogramm setzt zwei 4-Bit-Hexzahlen in eine zweistellige 8-Bit-Hexzahl um. Die erste Ziffer wird dabei um 4 Bit nach links verschoben und mit der zweiten Ziffer OR - verknüpft. Da die Gefahr besteht, daß versehentlich auch irgendwelche Funktionstasten gedrückt werden, maskieren wir zunächst die oberen 4 Bit jeder Tasteneingabe, bevor die Umsetzung erfolgt. Dies ist jedoch nur eine Notlösung, besser wäre es, die Überprüfung auf Zahleneingaben bereits bei der Tasteneingabe zu erledigen, hier haben Sie noch reichlich Spielraum für eigene Versuche.
Die eigentliche Addition wird ganz ähnlich wie auf S. 3 - 13 durchgeführt. Die Unterprogramme ADDIT 2 und ADDIT 3 unterscheiden sich nur hinsichtlich der Adressierung der Zahlenspeicher (siehe Adressenplan).
Das Hauptprogramm A-HAU verwaltet im wesentlichen die Registerpaare DE und HL, die die indirekte Adressierung der Tastenwerte und Anzeigezahlenspeicher übernehmen. Ab 01 17 finden Sie die Prozedur zum Unterprogramm CONVDI, die bis auf das Maskenwort dem Programm auf Seite 4 - 25 entspricht. Weitere Einzelheiten entnehmen Sie bitte den Programmlisten, die einzelnen Schritte sind ausreichend erläutert.

Das erweiterte Additionsprogramm eignet sich vorzüglich zur Berechnung von Entfernungsbytes bei relativen Sprüngen. Einzige Änderung:
Der Rechenbefehl in Adresse 02 22 muß von 86 = ADD A, (HL) in 96 = SUB A, (HL) abgeändert werden. Es entsteht dadurch ein Subtraktionsprogramm, das folgendermaßen benutzt wird:

1. und 2. Tasteneingabe: LOW-Byte der Zieladresse
3. und 4. Tasteneingabe: LOW-Byte der Startadresse

Ergebnis: Entfernungsbyte, vorzeichenbehaftete 8-Bit HEX-Ziffer.

Beispiel:

Tasteneingabe:	2 4	24: LOW-Byte Zieladresse
	2 9	29: LOW-Byte Startadresse
Ergebnis:		FB = -5

Adressenplan:

0300	Tastenwert 1		
0301	Tastenwert 2		nach 2 STELL: 1. Summand S1
0302	Tastenwert 3		
0303	Tastenwert 4		nach 2 STELL: 2. Summand S2
0304	Ergebnis ES		

0100	bis	0128	Hauptprogramm A-HAU
0130	bis	0146	SR - 2 STELL
0200	bis	0219	SR - 3. Test
021A	bis	0227	SR - ADDIT 3

Alle Programme sind frei verschiebbar, jedoch sind die Aufrufe der Unterprogramme in A-HAU und die Tastenwertspeicher absolut adressiert.

Karl Bloching SDR
Name und Rundfunkanstalt

4.1 Das Übungsprogramm PIO C, S. 4 - 11 zeigt im Normalbetrieb mit GO den Anfangswert nicht. Dieser Fehler wirkt besonders störend im Versuch 6, S. 4 - 17. Geben Sie an, welche Änderungen nötig sind, um den Anfangswert ebenso lange auf den Lampen zu sehen, wie die folgenden Werte.

Nach dem Befehl D 308 muß eine Zeit-
verzögerung "Call Zeit" eingefügt werden
damit die Ausgabe durch die Dioden
angezeigt wird und dann erst der
RLCA Befehl ausgeführt wird.

4.2 Schreiben Sie das Programm "7. Versuch" so um, daß die Richtung der Rotation nicht mehr vom Schalter SW 0 abhängt, sondern selbsttätig umkehrt, sobald das Einzellicht Bit 7 oder die Dunkelstelle Bit 0 erreicht hat.

5. Teil

Bearbeitungszeitraum
Dezember 1981, Januar 1982

Inhalt Seite

Übersicht	5 - 2
Kapitel 11, Anleitung	5 - 3
Versuche zu den Arithmetik-Befehlen	5 - 5
Kapitel 11. Fortsetzung	5 - 8
Die Anwendung des Compare-Befehls	5 - 9
Die Interrupt-Technik	5 - 11
Versuche zur Interrupt-Technik	5 - 12
5. Test	5 - 22

5. Teil - Übersicht

Sie werden im 5. Teil zwei sehr wichtige Themen bearbeiten. Einmal geht es um das Rechnen mit dem Mikroprozessor, um die Grundrechenarten Addition und Subtraktion und darauf aufbauend, die Multiplikation und die Division.
Die Interrupttechnik, das zweite Thema, ist mehr hardwarebetont und wird deshalb die Praktiker ansprechen.
Einige der Versuche, die die Lehrbücher zu diesen Abschnitten anbieten, sind teilweise zu aufwendig und dadurch auch undurchsichtig. Deshalb werden Sie auch Versuche mit Programmen aus der Arbeitsmappe durchführen.
Sprungmöglichkeiten bestehen im Abschnitt über die Interrupttechnik, Sie finden dort entsprechende Hinweise.

Die Arithmetik–Befehle ermöglichen das Rechnen mit dem Mikroprozessor. Grundsätzlich sind die beiden Rechenarten Addition und Subtraktion vorgesehen. Alle weiteren mathematischen Funktionen und Verfahren müssen per Software daraus abgeleitet werden, oft ein sehr schwieriges und zeitaufwendiges Vorhaben für den Anwender eines Systems.
Diese Schwierigkeiten kann man dadurch umgehen, daß man ein komplettes Softwarepaket vom Hersteller im ROM kauft, oder das Rechnen einem speziellen Baustein, einem "Arithmetic–Processor" oder "Number–Cruncher" überläßt. Beide Möglichkeiten scheiden aus, wenn der Mikroprozessor in kleinen Systemen sowieso nicht ausgelastet ist und nur einfache Rechnungen wie Additionen oder Multiplikationen auszuführen sind. Aus diesem Grund lohnt es sich für uns, die Arithmetik–Befehle genauer zu betrachten:
Der Z–80 erlaubt Rechnungen in 8 Bit und in 16 Bit Breite. Durch die Einbeziehung des Carry–Flags lassen sich diese beiden Datenbreiten jeweils verdoppeln, d. h. der Übergang von 8 zu 16 Bit und von 16 Bit zu 32 Bit Datenbreite ist möglich. Ebenso ist eine Verarbeitung von BCD–Zahlen mit 4 Bit für 10 Codewerte vorgesehen. Bei allen diesen Möglichkeiten spielen die Flags eine herausragende Rolle. Erst nach Auswertung der Flags, ergeben scheinbar sinnlose Rechenergebnisse sinnvolle Lösungen. Im Text des Kapitels 11 wird deshalb den Flags sehr viel Platz eingeräumt.

Lesen Sie bitte nun ab Seite 237 den Abschnitt "Die acht-Bit Arithmetik-Gruppe". Bitte korrigieren Sie vorher die Druckfehler nach den folgenden Angaben. Lesen Sie anschließend in der Arbeitsmappe weiter.

Druckfehler Seite 237, unten in der Tabelle

Zeile 4:

ADD A, 80 00 80 10 – 0 – 000

Zeile 5:

ADD A, 80 80 00 01 – 0 – 101

Druckfehler Seite 238, unten in der Tabelle

Zeile 6:

SBC A, 01 00 1 FE 10 – 1 – 011

Zeile 7:

SBC A, 80 00 1 7F 00 – 1 – 011

1. Addition und Subtraktion von 3-Byte Binärzahlen

Die Berechnung von Mehrbytezahlen wird aus der wiederholten Addition und Subtraktion von 1-Bytezahlen aufgebaut. Grundsätzlich ist das mit einem linearen Programm oder als Programmschleife möglich. Die beiden Programmbeispiele ADD und SUB zeigen den Unterschied sehr deutlich. Die Programmschleife ist die elegantere Form, ein deutlicher Vorteil zeigt sich bereits in unserem Beispiel, der 3-Byte Addition. Das lineare Programm belegt mehr Speicherraum als die Schleife.
Beide Programme sind an sich sehr ähnlich. Es müssen die Adressen der beteiligten Speicherplätze verwaltet werden, hierzu verwenden wir die Indexregister IX und IY. Im ersten Rechenschritt, der die niederwertigsten Bytes verarbeitet, darf das Carry-Flag das Ergebnis nicht verfälschen. Deshalb beginnt das lineare Programm die Rechnung mit dem ADD-Befehl, das Schleifenprogramm setzt mit XOR A das Carry-Flag am Anfang auf Null.

Speicherbelegung:

ADD: $X_3\,X_2\,X_1 + Y_3\,Y_2\,Y_1 = Z_3\,Z_2\,Z_1$ je 3 Byte

IX + 2 : X_3 später Z_3 IY + 2 : Y_3

IX + 1 : X_2 später Z_2 IY + 1 : Y_2

IX + 0 : X_1 später Z_1 IY + 0 : Y_1

SUB: $X_3 X_2 X_1 - Y_3 Y_2 Y_1 = Z_3 Z_2 Z_1$ je 3 Byte

Zuordnung wie ADD, jedoch andere Anfangswerte für IX und IY

Bitte laden Sie diese Programme und lassen Sie sich einige Zahlenbeispiele durchrechnen. Falls es Ihre Zeit zuläßt, sollten Sie noch versuchen, das Schleifenprogramm zu einem universellen Unterprogramm umzuwandeln. Da nur ein einziger Rechenbefehl vorkommt, ist es leicht, von außen die Rechenart (Subtraktion oder Addition) zu bestimmen. Hierzu könnten Sie eine weitere RAM-Zelle oder ein CPU-Register als Steuerung einsetzen. Ebenso könnten Sie die Zahlenbreite (N-Wert in B) variabel gestalten. Vielleicht wäre es auch ganz nützlich, wenn Sie das Setzen des Carry-Flags bei der Berechnung des höchstwertigen Bytes irgendwie berücksichtigen könnten, da in der vorliegenden Form ein Übertrag oder Borgen an höchster Stelle nicht im Ergebnis sichtbar wird.

2. Multiplikation und Division von Binärzahlen

Die Multiplikation und Division im Z–80 erfordern einigen Programmaufwand, da sie aus wiederholten Additionen und Subtraktionen hervorgehen. Das Verfahren ist der gewohnten schriftlichen Multiplikation/Division im Dezimalsystem sehr ähnlich, ein recht anschauliches Beispiel für die Multiplikation gibt der Versuch Nr. 1 Seite 245. Es genügt für uns, die Schritte 1 und 2 durchzuarbeiten. Das Programm 32 enthält einige Druckfehler:

Adresse:	Korrektur:
0 1 0 D	; wenn DE = 0, springe
0 1 1 C	CA 2 9 0 1
0 1 2 3	CALL C, 0 0 3 8

Das Verfahren zur Division stellt der Versuch Nr. 3 auf Seite 251 dar. Spielen Sie auch dieses Beispiel durch, es gelten hier zwar einige Einschränkungen, die durch die 8–Bit–Breite der Ergebniszelle vorgegeben sind, dafür bleibt das Programm selbst aber überschaubar. Die Fortsetzung finden Sie in der Arbeitsmappe.

Man hat Ihnen vom Anfang an die hexadezimale Zählweise zugemutet, schließlich rechnet man mit Ihrer Anpassungsfähigkeit und Lernbereitschaft. Anders ist es, wenn der Mikroprozessor als Steuerung in einem Fahrkartenautomat oder einer Zapfsäule an der Tankstelle eingebaut ist. Ein Kunde wird kaum richtig reagieren, wenn die Zapfsäule beispielsweise statt 90,13 DM nur 5 A, 0 D DM anzeigte.
Der Z-80 kann selbst nicht im BCD-Code rechnen, vielmehr muß die Binär-BCD-Umwandlung nach jedem Rechenschritt erfolgen. Hierzu dient der DAA-Befehl, der das H-Flag auswertet und für das richtige BCD-Ergebnis sorgt. Die Anwendung ist problemlos, beachten Sie aber, daß die Eingangsgrößen für Ihre Rechnung bereits als BCD-Zahlen vorliegen müssen. Da das oft nicht der Fall ist, haben wir Ihnen das Programmbeispiel HEX DE mitgeschickt, das 2-stellige Hexziffern in 3-stellige BCD-Ziffern umwandelt.
Bitte lesen Sie den Rest des 11. Kapitels von Seite 240 bis 243 durch. Für die DAA und 16-Bit-Arithmetik-Befehle führen Sie keine Versuche durch. Zum letzten Abschnitt über die Compare-Befehle, geben wir Ihnen auf der nächsten Seite ein Beispiel.

Die Anwendung des Compare-Befehls

Den Vergleich von Datenworten haben Sie im 4. Teil mit den Logik-Befehlen durchgeführt. Nachteil dabei war, daß nach der AND- oder XOR-Verknüpfung das Vergleichswort im Akku überschrieben wurde. Die neue Methode mit dem Compare-Befehl vergleicht Datenworte ohne sie zu verändern, Ergebnisse werden über die Flags mitgeteilt. Dadurch sind wiederholte Vergleiche und solche Befehle wie die Blocksuchbefehle möglich. Die Auswertung erfolgt über die bedingten Sprung-, Call- oder Returnbefehle. Folgende Aussagen sind möglich:

1. Z = 1: Beide Werte sind gleich
2. C = 1: Suchbyte ist größer als Akkuinhalt
3. C = 1, S = 1: Suchbyte ist höchstens 80 H größer als Akkuinhalt
4. C = 0, Z = 0: Suchbyte ist kleiner als Akkuinhalt

Zur Demonstration dieser Möglichkeiten dient das Programmbeispiel DEMO-CP. Führen Sie dieses Programm im SS-Modus aus und notieren Sie in der angegebenen Tabelle die Werte der Flags. Versuchen Sie dann, die oben angegebenen Aussagen dort einzuordnen. Vergleichsbyte ist für alle Fälle der Wert 20 H im Akku.

Ein echtes Suchprogramm finden Sie unter dem Programmnamen SUCH 3. Es ist als Unterprogramm geschrieben und kann beliebig eingesetzt werden. Beispielsweise ist möglich mit diesem Programm nach einem Codewort in Texteingaben zu suchen.

Texte verarbeitet der Mikroprozessor in Form von ASCII–Zeichen. Das sind 7–Bit Codeworte für 128 Schrift– und Steuerzeichen, die heutzutage von fast allen modernen Datensichtgeräten und Druckern verstanden werden. Die Codetabellen sind auf S. 53 im Z–80 Taschenbuch angegeben.

Basic–Tischcomputer verwenden Texte in ASCII zur Programmierung über eine Schreibmaschinentastatur. Hierbei muß der Mikroprozessor mit einem Suchprogramm die eingegebenen Texte in Befehle der Maschinensprache umwandeln. Die meisten kleinen Systeme arbeiten dabei mit drei Zeichen pro Befehl wie im Programmbeispiel SUCH 3. Aus dem Basic–Befehl GO TO 12 34 muß der Mikroprozessor einen Sprungbefehl zur Adresse 12 34 ableiten. Das Suchprogramm erkennt die ersten drei Buchstaben GO T und geht in ein Unterprogramm, das die Adresse in den Hexcode umwandelt und den Sprung ausführt.

Die Interrupt-Technik

Interrupts, zu deutsch Unterbrechungen, bringen Leben in die starre, an einen vorgegebenen Ablauf gebundene Programmierung. Sie erlauben sehr schnelle Reaktionen auf Umweltereignisse und sinnvolle Mehrfachausnutzung eines Prozessors.

Zur Einführung in diese Technik verwenden wir die Texte im Lehrbuch Band III, Seite 254 bis 265. Falls Sie es sich zutrauen, können Sie den allgemeinen Teil überspringen und gleich auf der Seite 261 beginnen. Eine Zusammenfassung enthält das Z–80 Taschenbuch auf den Seiten 44 bis 46. Die Versuche zu den verschiedenen Interrupts folgen in der Arbeitsmappe, lesen Sie zunächst den Einführungstext.

Die folgenden Versuche sind möglichst zusammenhängend zu bearbeiten, da die Programme stufenweise ausgebaut werden. Sie ersparen sich dadurch einige Eingaben.

1. Der nichtmaskierbare Interrupt NMI

Dieser Interrupt besitzt im Z-80 die höchste Priorität. Er ist weder durch Hardware, noch durch Software zu blockieren. Die Wirkung entspricht einem Call zur Adresse 0066, der durch ein 0-Signal am $\overline{\text{NMI}}$-Eingang der CPU ausgelöst wird. An dieser Adresse 0066 muß nun das gewünschte Interrupt-Service-Programm beginnen, diese und die nächste Adresse sind mindestens für einen relativen Sprung freizuhalten, also nicht mehr allgemein verwendbar. Das spezielle NMI-Unterprogramm muß mit dem RTN-Befehl schließen, damit der Prozessor das unterbrochene Hauptprogramm fortsetzen kann. Zur Demonstration des NMI sind nun vier Voraussetzungen zu schaffen:

1. Ein lauffähiges Hauptprogramm, das "sichtbar" unterbrochen werden kann.
2. Eine NMI-Service-Routine mit anschaulicher Wirkung.
3. Der Sprung zur NMI-Routine muß nach 0066 geladen werden.
4. Es ist für eine entsprechende Verdrahtung des $\overline{\text{NMI}}$-Eingangs der CPU zu sorgen.

Üblicherweise wird man die Punkte 3 und 1 zusammenfassen. Damit ist sichergestellt, daß sofort nach dem Start des Hauptprogramms ein NMI möglich ist.

Für diesen und spätere Versuche verwenden wir das Hauptprogramm BLINK 1. Die NOP–Befehle darin dienen der späteren Erweiterung. Der erste Teil des Hauptprogramms lädt die Adressen für den Sprung zur NMI–Service–Routine:

0066	C3
0067	00
0068	02

Dieser Sprungbefehl ist bereits der erste Befehl dieses NMI–Programms. Es läßt ein Lauflicht von rechts nach links über die Lampen laufen.

Geben Sie die Programme BLINK 1 und NMI ein. Folgende Drahtverbindungen sind erforderlich:

Alle PC (FL-Fassung links) mit allen LM (FR–Fassung rechts) in der richtigen Reihenfolge.

$\overline{\text{BNMI}}$ (FM–mittlere Fassung) mit $\overline{\text{P1}}$ auf FR. Diese letzte Drahtverbindung gibt den NMI–Impuls an die CPU. Der Impuls wird vom entprellten Impulsgeber P1 erzeugt (zweiter Schalter von rechts).

Starten Sie nun das Hauptprogramm. Die beiden 4er–Lampengruppen müssen im Gegentakt blinken. Nach einer kurzen Betätigung des P1 läuft das Lauflicht einmal über die Lampenreihe. Anschließend wird das Blinkprogramm fortgesetzt.

Wenn Sie nun, während das NMI–Programm läuft, mehrmals den P1 betätigen, werden Sie eine Überraschung erleben. Die CPU registriert alle NMI–Impulse, sie verschachtelt selbständig die einzelnen Unterbrechungsanforderungen und arbeitet sie nacheinander ab.

Übrigens verwendet das Nanocomputer–Betriebssystem den NMI für die Break–Funktion. Die Break–Taste hat in Ihrem Programm deshalb dieselbe Funktion wie der P1. Erst wenn Sie den Reset betätigen, arbeitet die Break–Taste wieder in der altgewohnten Weise, sehen Sie sich dann einmal die Sprungadressen 0066 bis 0068 an!

2. Der maskierbare Interrupt

Als Beispiel für einen maskierbaren Interrupt wählen wir die Interrupt–Methode IM1. Ein 0–Impuls am $\overline{\text{INT}}$–Eingang der CPU löst dabei einen CALL zur Speicheradresse 0038 aus. Dieser Interrupt wird jedoch nur akzeptiert, wenn das Hauptprogramm den Interruptfreigabebefehl EI enthält. Wie müssen folgendermaßen vorgehen:

1. Die CPU ist auf die Interrupt–Methode IM1 umzuschalten.
2. Der Interrupt ist zu sperren, damit vor Beginn des Hauptprogramms keine unkontrollierten Interrupts möglich werden.
3. Es ist ein Programm für den maskierbaren Interrupt bereitzustellen. Es muß mit dem Befehl RETI schließen (siehe Taschenbuch S. 46).
4. Der Sprung zu dem Interrupt–Service–Programm muß an die Speicheradressen 0038, 39, 3A geladen werden.
5. Der $\overline{\text{INT}}$–Eingang der CPU ist zu beschalten.
6. Das Hauptprogramm muß an den gewünschten Stellen mit den Interruptfreigabe– oder Interruptsperr–Befehlen versehen werden.

Die Punkte 1, 2 und 4 müssen bereits in der Einleitung zum Hauptprogramm Blink 1 abgearbeitet werden. Der besseren Übersicht wegen, lagern wir diesen Teil aus und geben ihm den Programmnamen INTV für Interrupt–Vorbereitung. INTV beginnt bei Adresse 0140, deshalb muß am Anfang des Programms BLINK 1 nun ein Sprung statt der NOP–Befehle eingesetzt werden. Bitte setzen Sie diesen Sprung jetzt ein und laden Sie auch die Programme INTV und INT1.

1. Änderung in BLINK 1:

0100	C3
0101	40
0102	01

Es ist wichtig, jetzt die Übersicht zu behalten. Fassen wir noch einmal zusammen:

Der Versuch soll die Arbeitsweise der maskierbaren Unterbrechung demonstrieren. In unserer Maschine ist bereits ein Hauptprogramm und ein nichtmaskierbarer Interrupt vom 1. Versuch her vorhanden. Zusätzlich werden nun eine Service–Routine für den maskierbaren Interrupt und die notwendigen Prozeduren dazu bereitgestellt. Mit der Eingabe von INTV und INT1, sowie der 1. Änderung in Blink 1, sind folgende Bedingungen erfüllt: 1, 2, 4 in INTV, 3 in INT1.

Es fehlen noch die Bedingungen 5 und 6.

Zu 5.:

Verbinden Sie den Anschluß $\overline{\text{BINT}}$ an FM mit $\overline{\text{PO}}$ an FR. $\overline{\text{BINT}}$ liegt neben $\overline{\text{BNMI}}$.

Zu 6.:

Ein maskierbarer Interrupt soll nur während der Hell–Phase der rechten Lampengruppe zugelassen werden, die Hell–Phase der linken Lampengruppe wollen wir vor Störungen durch maskierbare Interrupts schützen. Fügen Sie deshalb folgende Befehle in das Hauptprogramm BLINK 1 ein:

2. Änderung in BLINK 1

0116	FB	EI	Interruptfreigabe
011E	F3	DI	Interruptsperre

Nun sind alle Vorarbeiten abgeschlossen. Studieren Sie in aller Ruhe die Arbeitsweise der Interrupts. Folgende Möglichkeiten bieten sich an:

1. INT–Impuls von PO während der Hellphase der linken Lampengruppe (nur sehr kurz drücken!).
2. INT–Impuls von PO während der Hellphase der rechten Lampengruppe.
3. NMI–Impuls von P1 während das INT–Programm läuft.
4. INT–Impuls von PO während das NMI–Programm läuft.

Diese Kombinationen lassen aufgrund der eindeutigen Auswirkungen auf das Lampenfeld genaue Rückschlüsse auf das gerade laufende Programm zu.
Kombination 1 ist schwer zu treffen, hiermit können Sie Ihre Reaktionszeit testen.

3. Der Vektorinterrupt

Während alle übrigen Interruptarten auf bestimmte Anfangsadressen im unteren RAM–Bereich (HI–Byte 00) angewiesen sind, läßt der Vektorinterrupt alle Adressen im gesamten Speicherraum als Anfangsadressen der Interrupt–Service–Routinen zu. Durch diese Betriebsart gewinnt die Interruptverarbeitung eine wesentlich größere Flexibilität, jedoch muß der Interrupt über einen der Z–80 Peripheriebausteine eingespeist werden. Die CPU ermittelt nach dem Erkennen eines INT–Impulses einen sog. Interruptvektor in Zusammenarbeit mit dem Peripheriebaustein, der den Interrupt entgegengenommen, oder selbst erzeugt hat. Jeder Teilbereich eines solchen Bausteins enthält ein spezielles Interrupt–Vektor–Register, das über den Steuereingang mit dem LO–Byte einer Adresse zu Programmbeginn geladen wird. Das HI–Byte wird dem I–Register der CPU entnommen. Der Interruptvektor selbst stellt noch nicht die Adresse der Interrupt–Service–Routine dar. Er gibt die Adresse an, bei der die Anfangsadresse der Interrupt–Service–Routine abgelegt ist. Vektor und Anfangsadressen können beliebig oft und unabhängig voneinander mit unterschiedlichen Werten geladen werden. Dadurch sind mit derselben Hardware–Beschaltung zu verschiedenen Zeiten unterschiedliche Interrupt–Service–Routinen einschaltbar. Um diesen idealen Zustand zu erreichen, ist natürlich ein gewisser Programmieraufwand erforderlich. Die Prozedur sieht folgendermaßen aus:

Vorbereitung der CPU:

1. Sperren des Interrupts, um Störungen zu verhindern.
2. Einschalten der Interruptbetriebsart IM2.
3. Laden des I-Registers mit dem HI-Byte der Anfangsadresse der gewünschten Interrupt-Service-Routine.
4. Bereitstellen von Haupt- und Interruptprogramm.
5. Laden der Anfangsadresse an die Speicherzellen, die der Interruptvektor angibt.
   LO-Byte = Interruptvektor
   HI-Byte = Interruptvektor + 1

Vorbereitung des Peripheriebausteins:

Zur Demonstration des Vektorinterrupts verwenden wir die PIO D in der Betriebsart 3, Bit Ein/Ausgabe.

1. LO-Byte der Interrupt-Vektor-Adresse an das Steuerport, PIO D ausgeben. Vorschrift: Bit 0 = 0, deshalb sind nur gerade Adressen für den Interruptvektor möglich.
2. Steuerwort CF für Betriebsart 3 an PIO D, Steuerport ausgeben.
3. Im nächsten Steuerwort angeben, welche Leitungen Eingänge werden sollen. Bit = 1-Eingang, Bit = 0-Ausgang.
4. Interruptsteuerwort an PIO D, Steuerport ausgeben, es enthält mehrere Informationen zur Hardware-Anpassung:
   Bit 7: 0 = Interrupt ausschalten, 1 = Interrupt einschalten.
   Bit 6: 0 = Jeder Eingang kann einzeln den Interrupt auslösen.
   1 = Alle Eingänge gemeinsam lösen den Interrupt nach Art der UND-Verknüpfung aus.

Bit 5: 0 = Ein 0–Signal am Eingang löst den Interrupt aus.
1 = Ein 1–Signal am Eingang löst den Interrupt aus.

Bit 4: 0 = Alle Eingänge können den Interrupt auslösen.
1 = Es folgt ein Steuerwort, das einen Teil der Eingänge maskiert, so daß sie unwirksam werden.

Bit 3 bis Bit 0: 0111 für das Interruptsteuerwort festgelegt.

5. Ist im Interruptsteuerwort Bit 4 = 1, so muß nun ein Maskenwort an das Steuerport der PIO D ausgegeben werden, das festlegt, welche Eingangsleitungen den Interrupt auslösen sollen. Die 1 im Maskenwort verhindert einen Interrupt der betreffenden Leitung, die 0 läßt den Interrupt zu.

Aus dieser umfangreichen Liste wird deutlich, wie vielseitig diese Interruptbearbeitung sein kann. Hiermit lassen sich die Vorzüge der Mikroprozessortechnik in extremer Weise ausnutzen, denn die Z–80 PIO kann per Programm jeder beliebigen Hardwarebeschaltung angepaßt werden.

Nun zu dem eigentlichen Versuchsprogramm. Um Ihnen die Arbeit zu erleichtern, arbeiten wir mit den vorhandenen Programmen weiter, BLINK 1 bleibt Hauptprogramm, NMI bedient weiterhin den nichtmaskierbaren Interrupt und INT1 den maskierbaren Interrupt. Unterschiedlich ist die Art der Auslösung des INT1–Programmes. Es wird deshalb nötig sein, ein völlig neues Vorbereitungsprogramm, das die Programmierung von CPU und PIO D übernimmt,

einzugeben. Dieses Programm führt den Programmnamen VOR2. Bitte geben Sie VOR2 ein.

Im Hauptprogramm BLINK 1 ist eine Adresse zu ändern:

0102 statt 01 nun 04.

Weiterhin sind Drahtverbindungen von den Schaltern SW0 bis SW7 auf FR zu PD0 bis PD7 auf FL herzustellen. Bitte ziehen Sie den Draht von $\overline{PO}$ nach $\overline{BINT}$ heraus.

Nach diesen Vorbereitungen müßte sich eine Situation wie im 2. Versuch ergeben, mit dem Unterschied, daß der Interrupt nicht mehr mit PO ausgelöst wird, sondern sobald die Schalter SW0 und SW1 in "1"-Lage stehen. Nun bieten sich viele schöne Variationsmöglichkeiten an. Sofern es Ihre Zeit zuläßt, sollten Sie einen Teil davon ausprobieren:

1. SW0 und SW1 lösen den Interrupt einzeln aus.
2. Statt SW0, SW1 andere Schalter zur Interruptsteuerung heranziehen.
3. Verändern Sie den Interruptvektor und das Programm VOR2, um die Vorbereitungsprozedur genauer kennenzulernen.
4. Versuchen Sie eine neue Interrupt-Service-Routine zu schreiben, die über andere Schalter ausgelöst wird.

Aufschlüsselung des Interruptsteuerwortes in VOR2:

Steuerwort : F7 = 1111 0111

Bit 7 = 1 Interrupt einschalten

Bit 6 = 1 Eingänge UND-verknüpft

Bit 5 = 1 1-Signal löst Interrupt aus

Bit 4 = 1 Maske folgt

Bit 3 bis 0 durch PIO-Vorschrift vorgegeben.

5. Test

Wenn das Programm INT1 von einem NMI unterbrochen wird, ergibt sich anschließend eine völlig falsche Anzeige auf dem Lampenfeld, da kein Output–Befehl mehr vorkommt. Bitte versuchen Sie, ein neues Programm INT 2 zu schreiben, das auch nach Ablauf der NMI–Routine wieder alle Lampen einschaltet. Bitte geben Sie auf dem Programmformular Ihren Namen und die Rundfunkanstalt an.

6. Teil

Bearbeitungszeitraum:
Februar, März 1982

Inhalt	Seite
Übersicht	6 - 2
Die Z - 80 Zentraleinheit	6 - 3
Berechnung der Befehlsausführungszeit	6 - 5
Z - 80 Systembausteine	6 - 6
Weitere Mikroprozessoren	6 - 9
6. Test	6 - 10

6. Teil - Übersicht

Nachdem Sie sich bisher ausführlich mit dem Befehlssatz und der Programmierung des Z - 80 Mikroprozessors beschäftigt haben, werden wir Ihnen zum Schluß des Kurses noch einen kurzen Einblick in die Hardware vermitteln.

Zunächsteinmal geht es um die wichtigsten CPU - Steuersignale und die zeitsynchronen Abläufe bei der Befehlsausführung. Diese Themen liegen noch sehr nahe bei der Programmierarbeit. Durch die taktgebundene Arbeitsweise der CPU ergeben sich schließlich die zeitlichen Zusammenhänge zwischen Taktfrequenz, Maschinenzyklen und Bearbeitungsdauer eines Programmes.

Es lohnt sich, diesen Abschnitt sorgfältig zu durchdenken, denn es arbeiten fast alle programmgesteuerten Maschinen (nicht nur Mikroprozessoren) in ähnlicher Weise.

Anschließend informieren wir Sie über weitere Z -80 Systembausteine und schließen mit einer Zusammenstellung der wichtigsten Mikroprozessor - Familien.

Die Z - 80 Zentraleinheit, Anleitung

Der Text zu diesem Abschnitt befindet sich im Lehrbuch Band III, Seite 14 bis 26. Die ersten Erläuterungen beziehen sich auf das Blockschaltbild 1 - 2, Seite 15. Hierin wird die CPU als komplexe Einheit aus mehreren Baugruppen dargestellt. Bisher hatten Sie die CPU nur aus der Sicht des Programmierers betrachtet, etwa so, wie es das Bild 1 - 3 wiedergibt.

Leider sind in dem Text einige Druckfehler enthalten. Bitte korrigieren Sie die Fehler, bevor Sie den Text lesen.

Druckfehler:

Seite 17, Zeile 30:

. . . oder einem Ein-/Ausgabegerät ablesen will.

Seite 18, Zeile 20:

$\overline{NMI}$ Eingang, getriggert von der negativen Flanke . . .

Seite 19, Zeile 5:

$\emptyset$ Einphasiger TTL - Takteingang.

Zeile 17:

7. Ausstieg infolge HALT-Befehl.

Zeile 27:

. . . Der Speicherbaustein interpretiert diese Signale...

vorletzte Zeile

Der Z - 80 entschlüsselt den Code 3 E als unmittelbaren Ladebefehl für den Akku.

Seite 23, letzte Zeile:

. . . über (überstrichene Signale (negiert!) sind im Ruhezustand high, während nichtüberstrichene normalerweise low sind).

Ergänzend zu diesem Text, könnten Sie noch die Seiten 52 bis 67 lesen, hier finden Sie weitere Informationen zu den Zeitabläufen bei verschiedenen Befehlen.

Berechnung der Befehlsausführungszeit

Die Zeitdauer, für die ein Mikrocomputer mit der Ausführung eines bestimmten Programmteils beschäftigt ist, läßt sich genau bestimmen. Unabhängig vom jeweiligen Gerät und dessen Taktgenerator, sind für jeden Befehl die erforderlichen Taktzyklen in den Befehlslisten angegeben. Bei Befehlen, die Bedingungen auswerten, sind zwei verschiedene Werte angegeben, die für die erfüllte oder nicht erfüllte Bedingung gelten. Die Erläuterung der Berechnung anhand von Beispielen finden Sie im Band I, Seite 276 und 277. Bitte arbeiten Sie diese Seiten durch. Im Gegensatz zum Text dort, hat sich in der Praxis folgende Vereinfachung bewährt:

Statt für jeden Befehl zusätzlich zur Anzahl der Taktzyklen auch noch die Ausführungszeit zu berechnen, genügt es, die Summe der Taktzyklen aller Befehle des Programms zu ermitteln und erst dann mit der Taktzeit zu multiplizieren.

Einmal ermittelte Programmausführungszeiten halten die Mikroprozessoren sehr genau ein, da die Taktfrequenzoszillatoren in der Regel quarzstabilisiert sind.

Z - 80 Systembausteine

Sieht man einmal von sogenannten Einchip-Mikrocomputern ab, so sind die CPU-Bausteine allein nicht lebensfähig. Für ein komplettes Mikrocomputersystem sind noch weitere Bausteine wie Speicher und Ports erforderlich.

Speicher, als RAM oder ROM, sind Massenware und universell einsetzbar, deshalb ist es nicht üblich, spezielle Speicherbausteine für eine CPU vorzuschreiben. Es ist lediglich eine Abstimmung bezüglich der Taktfrequenz des Systems und der Zugriffszeit der Speicher nötig (s. Bd. III, S. 66).

Anders verhält es sich mit den Peripheriebausteinen. Diese sind oft mit zahlreichen komplexen Funktionen ausgerüstet, die auf die Steuerung durch einen ganz bestimmten Mikroprozessortyp zugeschnitten sind. So ist jeder Hersteller bemüht, seine CPU mit einer vielseitigen Bausteinfamilie auszurüsten, die die besonderen Eigenschaften seines Systems voll zur Geltung bringen.

Dies gilt für den Z - 80 fast nicht mehr. Als Universal-Prozessor kann er leicht mit Systembausteinen anderer Mikroprozessor-Familien zusammenarbeiten, jedoch muß man dann auf den Vektor-Interrupt verzichten. Der Vektor-Interrupt ist die herausragende Fähigkeit des Z - 80 Systems, für die das Z - 80 Familienkonzept ausgelegt ist. Dinge wie Prioritäts-Steuerung der Interrupts und vielseitige Steuermöglichkeiten durch die Software

eine Rolle. Im Rahmen dieses Einführungskurses können wir diese Einzelheiten nur kurz ansprechen. Ausführlichere Informationen finden Sie im Lehrbuch Band III, Kapitel 7 und 9, hier besonders ab Seite 384.

Es folgen Kurzbeschreibungen der Z - 80 Systembausteine.

Z - 80 PIO (s. Bd. III, S. 306)

Parallel Input/Output Interface Controller

Zwei unabhängige 8 - Bit bidirektionale Ports

Vier Betriebsarten durch Software anwählbar

Eigene Interrupt - Logik

Z - 80 CTC (s. Bd. III, S. 376)

Counter - Timer - Circuit

Vier voneinander unabhängig programmierbare 8 - Bit Zähler/Zeitgeber - Kanäle

Rückwärtszähler mit Konstanten - Register

Eigene Interrupt - Logik für jeden Kanal

Wählbare Triggerflanken

Z - 80 SIO

Serial Input/Output Controller

Zwei unabhängige Vollduplex - Kanäle für serielle Datenübertragung

Steuer- und Statussignale für Datenaustausch

Übliche asynchrone und synchrone Verfahren wählbar

Eigene Interrupt - Logik für selbsttätige Arbeitsweise

Z - 80 DART

Dual Asynchronous Receiver/Transmitter

Zwei unabhängige Vollduplex-Kanäle für serielle Datenübertragung. Asynchrone Betriebsart mit Statussignalen des Modems. Übliche Übertragungsverfahren programmierbar.

Eigene Interrupt - Logik

Z - 80 DMA

Direct Memory Access

Automatischer Datentransfer zwischen Ports und Speicher, unabhängig von der CPU

Vielseitige, programmierbare Funktionen

Im Z - 80 System keine weitere Logik erforderlich

Eigene Interrupt - Logik

Weitere Mikroprozessoren

Nachdem Sie sich ausführlich mit dem Z - 80 Mikroprozessor beschäftigt haben, wird es Ihnen leicht fallen, auch andere Systeme zu verstehen. Die grundsätzliche Arbeitsweise aller Mikroprozessoren ist etwa gleich. Gravierende Unterschiede gibt es hauptsächlich in der mnemonischen Schreibweise der Befehle und den Befehlscodes. Die Befehle selbst sind oft identisch oder ähneln sich. Auffällig ist dabei, daß einige der älteren Mikroprozessoren einzelne Adressierungsarten oder bestimmte Befehlsgruppen überhaupt nicht kennen. Insofern können Sie den Übergang vom Z - 80 zu anderen Systemen leicht bewältigen, jedoch werden Sie oft einige der komfortablen Befehle vermissen.

Unterschiede bestehen auch in der Hardware-Beschaltung und bezüglich der internen Register der CPU. Hierzu finden Sie Hinweise auf den DIN A 4 - Übersichtsblättern.

6. Test

Karl Blocking SDR

Name und Rundfunkanstalt

6.1 Wodurch meldet der Z - 80 nach aussen, daß er gerade dabei ist, einen Befehlscode zu lesen?

M1 / RD ✓

6.2 Wie lange dauert die Bearbeitung des folgenden Programms in einem Z - 80 mit 2.5 MHz Taktfrequenz?

19,64 sec / gestoppt 20 sec

wichtig!

Adresse	Op-Code	Label	Mnemonic
	01 CO FO		LD BC, FO CO
	3E 30		LD A, 30
	3D		DEC A
	20 FD		JR NZ, -3
	OB		DEC BC
	78		LD A, B
	B1		OR C
	20 F6		JR NZ, - 10
	OO		NOP

6.3 Preisfrage

Das beiliegende Programm "Preis" läßt die LEDs des Nanocomputers nur dann leuchten, wenn vorher die Schalter SW7 bis SW0 in einer bestimmten, vom Programm vorgegebenen Weise eingestellt werden.

Drahtverbindungen:
SW7-SW0 mit PD7 -PD0 und LM7-LM0 mit PC7-PC0.

Wie müssen die Schalter betätigt werden, damit alle LEDs leuchten?
Bitte geben Sie die Schalterstellungen in HEX-Schreibweise und in der richtigen Reihenfolge an.

FF, 18 H, F6, 49 H

00 H