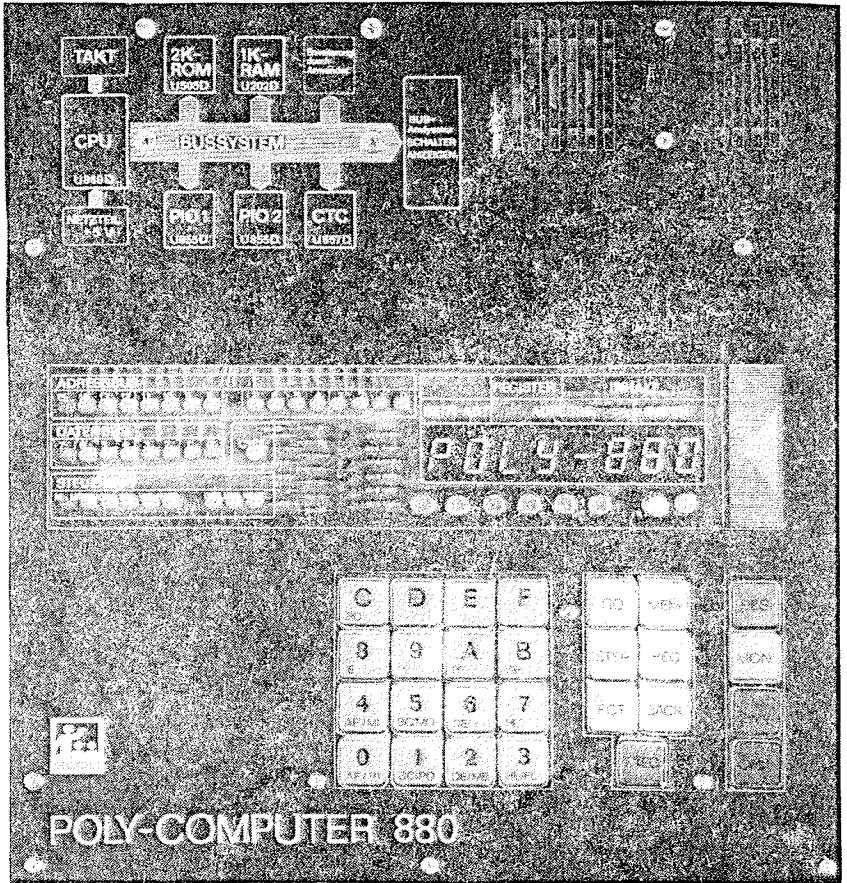


# POLY-COMPUTER 880

Arbeitsbuch

11 541 9.01



D - 0883 - 11541

VEB KOMBINAT  
POLYTECHNIK UND PRÄZISIONSGERÄTE  
KARL-MARX-STADT



A r b e i t s b u c h

P O L Y - C O M P U T E R 8 8 0

T e i l I: Grundlagen der Geräte- und  
Programmtechnik von Mikrorechnern

Autorenkollektiv

Dipl.-Ing. Steffen Burkhardt  
Dipl.-Ing. Uwe Hübner  
Dipl.-Ing. Andreas Troll

I n h a l t

Vorwort zum Arbeitsbuch

Erstes Kapitel: Einführung in den Problembereich Digitalrechner

- 1.1. Grundlagen
  - 1.1.1. Der Rechner
  - 1.1.2. Hardwaregrundstruktur eines Rechners
  - 1.1.3. Softwaregrundbegriffe
- 1.2. Zahlensysteme und rechnerinterne Darstellung
  - 1.2.1. Dezimales und binäres Zahlensystem
  - 1.2.2. Operationen im Binärsystem
  - 1.2.3. Hexadezimale Darstellung
- 1.3. Der Speicher eines Rechners und seine Organisation
  - 1.3.1. Speicherorganisation
  - 1.3.2. Speicherzugriff
  - 1.3.3. Speicherarten
- 1.4. Die Struktur der Zentralen Verarbeitungseinheit (CPU)
  - 1.4.1. Die Funktionseinheiten der CPU
  - 1.4.2. Die Ausführung eines Programmes
  - 1.4.3. Der Befehlszyklus
  - 1.4.4. Der Programmzähler (PC) und das Befehlsregister
  - 1.4.5. Der Akkumulator (A)
- 1.5. Erste Versuche mit dem POLY - COMPUTER
  - 1.5.1. Der Monitor
  - 1.5.2. Zur Bedienung des POLY-COMPUTER
  - 1.5.3. Zusammenfassung
- 1.6. Das erste eigene Programm
  - 1.6.1. Einige Prozessorbefehle
  - 1.6.2. Schreiben des Programmes
  - 1.6.3. Abspeichern(Laden) des Programmes im POLY-COMPUTER
  - 1.6.4. Ausführung des Programmes
  - 1.6.5. Zusammenfassung
- 1.7. Schlußwort zum ersten Kapitel

Zweites Kapitel: Ausgewählte Befehle und einige Flags der CPU U880

- 2.1. Mehrbytebefehle
- 2.2. Befehlszyklus und Maschinenszyklus
  - 2.2.1. Wichtige Zyklusarten und Steuersignale der CPU U880. - Der Steuerbus
  - 2.2.2. Die Arbeit mit dem Busanalysator des POLY-COMPUTERS
  - 2.2.3. Start von Programmen und Prüfpunkte im POLY-COMPUTER
- 2.3. Die Programmverzweigung
  - 2.3.1. Die unbedingte Verzweigung
  - 2.3.2. Einige Flags der CPU U880
  - 2.3.3. Die bedingte Verzweigung
  - 2.3.4. Zusammenfassung

Drittes Kapitel: Grundsätze für die Gestaltung von Programmen

- 3.1. Elemente zur Programmstrukturierung
  - 3.1.1. Die Folge
  - 3.1.2. Der Zyklus
  - 3.1.3. Die Auswahl
  - 3.1.4. Der Programmablaufplan
- 3.2. Strukturierung von Programmen
- 3.3. Zusammenfassung

Viertes Kapitel: Die Register der CPU U880

- 4.1. Übersicht
- 4.2. Der Hauptregistersatz
  - 4.2.1. Überblick
  - 4.2.2. Der Ladebefehl
  - 4.2.3. Ein Multiplikationsprogramm
- 4.3. Der Alternativregistersatz
- 4.4. Die Spezialregister
  - 4.4.1. Die Indexregister IX und IY
  - 4.4.2. Der Stapelzeiger (SP)
- 4.5. Zusammenfassung

Fünftes Kapitel: Unterprogramntechnik und Stapelspeicher

- 5.1. Aufruf von Unterprogrammen und Rückkehr zum aufrufenden Programm
- 5.2. Übergabe von Variablen an Unterprogramme - Rettung der Registerzustände
  - 5.2.1. Variablenübergabe in den CPU-Registern

- 5.2.2. Variablenübergabe auf Speicherplätzen
- 5.3. Gestaltung und Schachtelung (Nesting) von Unterprogrammen
- 5.4. Bedingte Ausführung von CALL und RET
- 5.5. Die für den Anwender nutzbaren Unterprogramme des Monitors
- 5.6. Zusammenfassung

Sechstes Kapitel: Logische und Bitbefehle

- 6.1. Rotations- und Verschiebefehle
  - 6.1.1. Rotieren im A-Register
  - 6.1.2. Rotieren in verschiedenen CPU-Registern und in Speicherplätzen
  - 6.1.3. Verschiebefehle
- 6.2. Bitbefehle
- 6.3. Logikbefehle
- 6.4. Verwendung von Logik- und Bitbefehlen zur Steuerung der Tastatur und des Displays des POLY-COMPUTERS
- 6.5. Zusammenfassung

Siebentes Kapitel: Anschlußtechnik des Mikrorechners

- 7.1. Das Buskonzept
  - 7.1.1. Prinzip
  - 7.1.2. Elektrische Bedingungen für den Busanschluß
  - 7.1.3. Der Systemsteckverbinder des POLY-COMPUTERS
- 7.2. Die zeitliche Steuerung der CPU (timing)
  - 7.2.1. Das Taktdiagramm
  - 7.2.2. Der Anschluß von Speicherschaltkreisen
- 7.3. Zusammenfassung

Achstes Kapitel: Techniken der Ein- und Ausgabe (E/A)

- 8.1. Grundlagen der Ein-/Ausgabe-Technik
  - 8.1.1. Einführungsbeispiel E/A-Befehle
  - 8.1.2. Weitere E/A-Befehle der CPU U880D
- 8.2. CPU-gesteuerte Ein-/Ausgabe durch Abfrage
- 8.3. Ein-/Ausgabesynchronisierung durch Verlängerung der E/A-Zyklen
- 8.4. Unterbrechungsgesteuerte Ein-/Ausgabe
  - 8.4.1. Das Unterbrechungsprinzip
  - 8.4.2. Nichtmaskierbare Unterbrechung - NMI
  - 8.4.3. Maskierbare Unterbrechung INT (Interrupt)
  - 8.4.4. Unterbrechungsbetriebsart und deren Abläufe
- 8.5. Direkter Speicherzugriff
- 8.6. Der Parallel-Ein-/Ausgabeschaltkreis U855 (PIO)

- 8.6.1. Überblick
- 8.6.2. Programmierung der PIO
- 8.6.3. Beispiel
- 8.7. Der Schaltkreis U857 (CTC)
- 8.7.1. Überblick
- 8.7.2. Programmierung des CTC
- 8.7.3. Beispiel
- 8.8. Unterbrechungssteuerung mit Peripherieschaltkreisen
- 8.8.1. Die Prioritätslogik der E/A-Schaltkreise
- 8.8.2. Programmbeispiel - Unterbrechungsgesteuerte Digitaluhr
- 8.9. Monitorunterstützung für Ein-/Ausgaben im POLY-COMPUTER
- 8.9.1. Belegte E/A-Kanäle
- 8.9.2. Zugriff zu E/A-Kanälen
- 8.9.3. Magnetbandanschluß des POLY-COMPUTERS
- 8.9.4. Der Fernschreiberanschluß des POLY-COMPUTERS
- 8.10. Zusammenfassung

#### Neuntes Kapitel: Binäre und dezimale Arithmetik

- 9.1. Darstellung und Operationen natürlicher Zahlen
- 9.2. Ganze vorseichenbehaftete Zahlen
- 9.3. Zahlenbereichsüberschreitungen beim Rechnen mit ganzen Zahlen und Zweierkomplementdarstellung
- 9.4. Festkommazahlen (fixed point numbers)
- 9.5. Gleitkommazahlen (floating point numbers)
- 9.6. BCD-Darstellung von Zahlen
- 9.7. Andere mathematische Operationen
- 9.8. Zusammenfassung

#### Zehntes Kapitel: Ergänzungen und Beispiele

- 10.1. Die vollständige Befehlsliste
- 10.1.1. Überblick
- 10.1.2. Relative Adressierung
- 10.1.3. Blockbefehle
- 10.2. Erzeugung von Maschinenkode mittels Programmunterstützung
- 10.2.1. Assembler
- 10.2.2. Die Assemblerliste
- 10.3. Beispiele
- 10.3.1. Der POLY-COMPUTER als einfacher Taschenrechner
- 10.3.2. Der POLY-COMPUTER macht Musik

#### V o r w o r t

Die Mikroelektronik hält in rasantem Tempo Einzug in alle Zweige der Volkswirtschaft und stellt höchste Ansprüche an Wissenschaft, Forschung und Technologie. Der Ausbildung, besonders auf dem Gebiet der Mikroelektronik-Anwendung, wird an Universitäten, Hoch- und Fachschulen, Betriebsakademien, durch die Kammer der Technik, die Urania und viele andere große Bedeutung beigemessen und es werden enorme Anstrengungen unternommen.

Mit dem POLY-COMPUTER 880 wird den Ausbildungseinrichtungen und jedem Interessierten ein modernes Lernmittel zur Verfügung gestellt, mit dem eine praxisnahe Aneignung fundamentaler Kenntnisse über die Programmierung und die Gerätetechnik von Mikrorechnern ermöglicht wird.

Zum POLY-COMPUTER wurde ein umfangreiches Anleitungs- und Faktenmaterial geschaffen, das folgende relativ selbständige Bücher umfaßt:

- Arbeitsbuch (Teile I und II) POLY-COMPUTER 880
- Systemhandbuch POLY-COMPUTER 880
- Bedienhandbuch POLY-COMPUTER 880

Das Bedienhandbuch enthält die Dokumentation des POLY-COMPUTERS, die Beschreibung der Bedienung und die Liste des Monitors (Steuerprogramm), die Schaltungsunterlagen usw. Es ist als Nachschlagewerk und Vervollständigung des Arbeitsbuches gedacht und liefert auch die notwendigen Informationen für gerätetechnische Erweiterungen durch den Nutzer.

Im Systemhandbuch sind in übersichtlicher und straffer Form Kurzbeschreibungen der U880-Schaltkreisfamilie einschließlich mehrerer Befehlslisten, elektrischer Kennwerte mit Diagrammen, Programmierung und einer Reihe nützlicher Tabellen enthalten. Diese, auch für den Fortgeschrittenen interessante Zusammenstellung, ist insbesondere nach Absolvierung des ersten Arbeitsbuchteils eine unentbehrliche Hilfe beim Programmieren und Entwerfen neuer Gerätetechnik.

Den umfangreichsten Teil des Anleitungsmaterials bilden die Arbeitsbücher. Der vorliegende erste Teil setzt keinerlei Kenntnisse auf den Gebieten Mikroelektronik-Schaltungstechnik oder Programmierung voraus, erwartet lediglich elementare Kenntnisse in der Elektrotechnik/Elektronik (z.B. Funktionsweise Transistor, Spannung, Strom, Potential, Widerstand) und der Mathematik.

Der Leser kann sich im Selbststudium die Grundlagen der Funktionsweise eines Digitalrechners (im Buch als Rechner bezeichnet), seine Schaltungstechnik und wesentliche Elemente der maschinennahen Programmierung aneignen. Bis auf wenige, notwendige Ausnahmen erfolgen Vermittlung neuer Fakten und Zusammenhänge und praktische Verifizierung am POLY-COMPUTER Hand in Hand. Somit sollte der POLY-COMPUTER bei der Lektüre stets greifbar sein.

Wichtige Begriffe werden bei ihrem ersten Auftreten im Text unterstrichen und nachfolgend erläutert. Begriffe und Abkürzungen in englisch werden dann verwendet, wenn der entsprechende deutschsprachige Begriff in der einschlägigen Literatur nicht üblich ist. Eine deutsche Übersetzung wird angegeben.

System- und Bedienhandbuch sind beim Studium des Arbeitsbuches (Teil I) nicht unbedingt erforderlich, die nötigen Bedien- und andere Detailkenntnisse werden schrittweise vermittelt. Diese Bücher sind jedoch zur Vertiefung und Vervollständigung des neu erworbenen Wissens bzw. später als Nachschlagewerk sinnvoll bzw. unentbehrlich.

Großer Wert wird von Beginn an auf "saubere", sprich "strukturierte" Programmieretechnik gelegt. Diese, möglicherweise anfangs nur schwer zu akzeptierende Strenge bei der Programmgestaltung kommt dem Leser bei später zu lösenden komplexen Problemstellungen zugute.

Nach gewissenhafter Absolvierung des Arbeitsbuches Teil I ist der Leser in der Lage, anspruchsvolle Programme in symbolischer Maschinensprache für den Prozessor U680 unter Einbeziehung seiner Parallelperipherie (PIO) und des Zeitgeberbausteins (CTC) und deren Unterbrechungsmöglichkeiten zu verfassen und mit Hilfe des POLY-COMPUTERS zu testen. Er besitzt darüberhinaus ausreichende Kenntnisse zur Gerätetechnik des Mikrorechners, um einfachere Fehler ort und beseitigen, sowie mit Hilfe des Systemhandbuches kleinere Mikrorechnersteuerungen entwerfen zu können.

Das Arbeitsbuch enthält eine Fülle von Beispielprogrammen zu den jeweiligen Abschnitten, die im Inhaltsverzeichnis nicht aufgeführt sind.

Im Arbeitsbuch Teil II wird vor allem auf die Anwendung des POLY-COMPUTERS als Steuerrechner unter Ausnutzung seiner peripheren Schnittstellen eingegangen und anhand praxisnaher Beispiele das erworbene Wissen vertieft.

## 1. Einführung in den Problembereich Digitalrechner

### 1.1. Grundlagen

#### 1.1.1. Der Rechner

Unter einem Rechner verstehen wir ein elektronisches System, das arithmetische und logische Operationen mit Daten entsprechend einem Programm (Folge von Anweisungen) ausführen kann. Das System besteht aus zwei Hauptkomponenten:

- Hardware (Gerätetechnik) und
- Software (Folge von Anweisungen).

Hardware (engl.-wörtlich übersetzt "harte Ware") stellt dabei einen Sammelbegriff für die gegenständlichen Komponenten elektronischer Systeme dar (Gesamtheit der elektronischen und mechanischen Bauelemente, wie z.B. Integrierte Schaltkreise, Transistoren, Widerstände, Leiterplatten, Schalter; aber auch z.T. die Art und Weise ihrer Verknüpfung).

Als Software (engl.-wörtlich übersetzt "weiche Ware") dagegen wird die Programmkomponente eines Systems bezeichnet. Zunächst wollen wir uns mit der Hardware-Grundstruktur eines Rechners beschäftigen.

#### 1.1.2. Hardwaregrundstruktur eines Rechners

Ein Rechner besteht aus drei Hauptbaugruppen:

- Zentrale Verarbeitungseinheit (ZVE)  
(oder engl. CPU für Central Processing Unit)
- Speicher (Memory)
- Eingabe (Input) - und Ausgabe (Output) - Einheiten  
(Abkürzung E/A- oder I/O-Einheiten)

Die CPU führt als Kern des Rechners die arithmetischen und logischen Operationen aus und vollzieht darüberhinaus zentrale Steuerfunktionen für den gesamten Rechner.

Im Speicher sind Daten und Befehle in entsprechender Reihenfolge abgelegt.

Die E/A-Geräte und -Kanäle ermöglichen den Austausch von Daten zwischen dem Rechner und seiner Umgebung. So dienen beispielsweise häufig Tastaturen zur Eingabe von Operationen und Bildschirmgeräte oder andere Anzeigeeinheiten zur Ausgabe der Resultate.

Diese drei Rechnerbaugruppen sind so untereinander verbunden, daß zumindest die CPU zu den anderen beiden zugreifen kann (siehe Bild 1.1.).

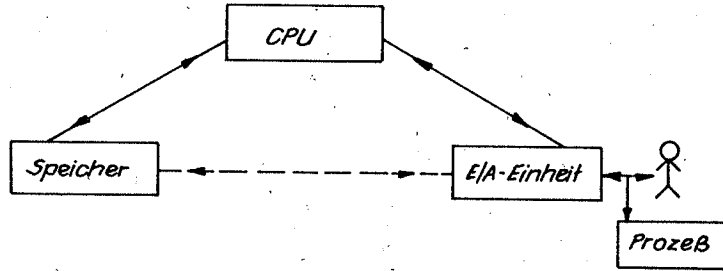


Bild 1.1. Hardwaregrundstruktur eines Rechners

Über die E/A-Einheiten hat beispielsweise der Mensch die Möglichkeit, auf den Rechner Einfluß zu nehmen bzw. werden Informationen zwischen Rechner und zu steuerndem Prozeß (z.B. eine Werkzeugmaschine) ausgetauscht.

Die sich im Rechner zwischen diesen drei Hauptbaugruppen vollziehenden Abläufe können folgendermaßen zusammengefaßt werden:

0. Über die E/A-Einheiten gelangt das auszuführende Programm in den Speicher (dieser Schritt entfällt dann, wenn bestimmte, fest programmierte Speicher eingesetzt werden, die bereits das Programm enthalten).
1. Aus einem Anfangszustand (Initialisierung) heraus beginnt die Programmabarbeitung. Dazu wird in einer vom Programm bestimmten Reihenfolge Befehl nach Befehl aus dem Speicher in die CPU geholt und dort ausgeführt.
2. Die für die Verarbeitung erforderlichen Daten gelangen meist programmgesteuert über die E/A-Einheit in den Speicher.
3. Entsprechend den Programmvorgaben werden die Daten verarbeitet.
4. Die Ergebnisse dieser Verarbeitung werden über die E/A-Einheiten ausgegeben.

### 1.1.3. Softwaregrundbegriffe

Ein Rechner vollzieht seine Funktionen nach einem Programm.

Unter einem Programm wollen wir eine Folge von Anweisungen verstehen, die bei Ausführung eine bestimmte (sinnvolle) Funktion (Berechnungen, logische Operationen) realisiert.

Beispiel: Folgende Aufgabe soll durch einen Rechner gelöst werden: Addiere zwei Spannungswerte und dividiere die Summe durch 2, so daß als Ergebnis der Mittelwert vorliegt!

Unser Rechner nach Bild 1.1. benötigt eine wesentlich detailliertere Angabe der einzelnen Arbeitsschritte, so daß ein Programm für ihn z.B. so aussehen könnte:

- (1) BEGINN
- (2) EINGABE ERSTER SPANNUNGSWERT
- (3) ABLAGE IM SPEICHER UNTER (SP1)
- (4) EINGABE ZWITTER SPANNUNGSWERT
- (5) ABLAGE IM SPEICHER UNTER (SP 2)
- (6) HOLE SP1 ZUR CPU AUS SPEICHER
- (7) HOLE SP2 ZUR CPU AUS SPEICHER
- (8) 'ADDIERE SP1 MIT SP2
- (9) ABLAGE SUMME IM SPEICHER UNTER (SUM)
- (10) HOLE (SUM) ZUR CPU AUS SPEICHER
- (11) HOLE '2' ZUR CPU AUS SPEICHER
- (12) DIVIDIERE (SUM) DURCH '2'
- (13) ABLAGE ERGEBNIS IM SPEICHER UNTER (MITTELWERT)
- (14) AUSGABE (MITTELWERT)
- (15) ENDE

In dieser verbalen Form ist ein Programm für einen Rechner jedoch nicht unmittelbar verständlich. Die einzelnen Programmschritte sind in eine für den Rechner verarbeitbare Sprache zu übersetzen. Für alle Rechner muß letztendlich ein Programm in Form von einer Folge von Zahlen vorliegen, die vom Rechner als Anweisungen akzeptiert, interpretiert und ausgeführt werden können. Diese Zahlen werden im Speicher, in der CPU usw. auf eine spezielle, den Möglichkeiten der Elektronik angepaßt und im schaltungstechnischen Aufwand optimale Weise dargestellt.

### 1.2. Zahlensysteme und rechnerinterne Darstellung

#### 1.2.1. Dezimales und binäres Zahlensystem

Für Berechnungen aller Art verwenden wir ausschließlich eine Zahlendarstellung, die auf Potenzen der Zahl 10 beruht. So repräsentieren die Ziffern einer Zahl (z.B. 8537) Faktoren für Potenzen der Zahl 10.

$$\begin{aligned}
 \text{Bsp.: } 7 \times 10^0 &= 7 \\
 3 \times 10^1 &= 30 \\
 5 \times 10^2 &= 500 \\
 8 \times 10^3 &= 8000 \\
 \text{Summe } &8537 \\
 &=====
 \end{aligned}$$

Diese Tatsache berücksichtigen wir, wenn auch mit zunehmender Übung unbewußt, bei sämtlichen Operationen (z.B. Addition, Multiplikation) mit Zahlen.

Die Anzahl der unterschiedlichen Ziffern wird als Basis des jeweiligen Zahlensystems bezeichnet. In unserem Fall existieren zehn verschiedene Ziffern (0,1,2,3,4,5,6,7,8,9). Somit ist die Basis unseres Zahlensystems 10 und es wird aus diesem Grunde als Dezimalsystem bezeichnet. Würde ein Rechner ebenfalls das Dezimalsystem für die interne Zahlendarstellung verwenden, müßte eine Möglichkeit gefunden werden, physikalisch 10 verschiedene Zustände (z.B. 10 verschiedene Spannungswerte) reproduzierbar zu erzeugen und zu speichern. Dieser prinzipiell mögliche Weg wird aus Gründen der Zuverlässigkeit und des hohen Aufwandes nicht gegangen. Verwendet man aktive elektronische Bauelemente als Schalter (z.B. Relais, Transistor, Bild 1.2.) so daß sie sich nur in einem von zwei möglichen Zuständen befinden können (offen, geschlossen), so

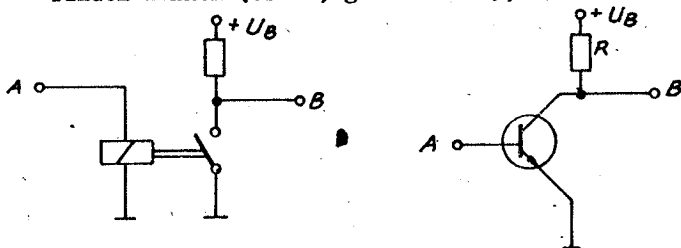


Bild 1.2. Aktive elektronische Bauelemente als Schalter Relais und Transistor

erhält man aufgrund der extremen Spannungsunterschiede am Ausgang des Elements zwischen den beiden Zuständen eine hohe Sicherheit bei der Unterscheidung der Zustände. Ebenfalls gestaltet sich die Ansteuerung der Bauelemente äußerst einfach. Bei Anlegen eines ausreichend positiven Potentials an Punkt A wird Punkt B auf Masse gelegt (entspricht etwa 0V- Zustand 1). Ist das Potential von Punkt A etwa 0V, öffnet das Relais bzw. sperrt der Transistor, und wir entnehmen an Punkt B ein Potential von etwa +U<sub>B</sub> (Zustand 2). Die Speicherung solcher zweiwertiger (binärer) Zustände bereitet mit elektronischen Mitteln ebenfalls nur geringe Mühe.

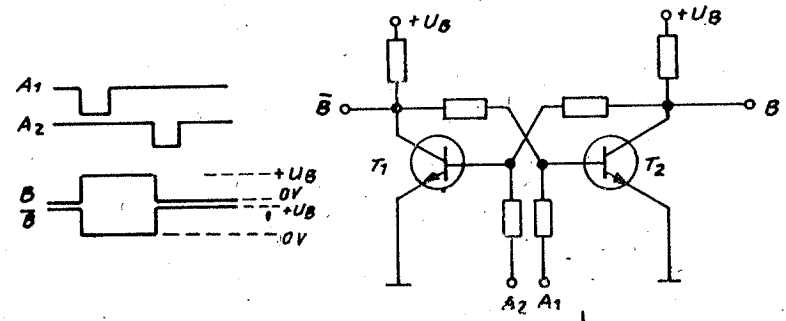


Bild 1.3. Binäres Speicherelement (Flip-Flop=FF)

Bild 1.3. zeigt ein Speicherelement mit zwei stabilen Zuständen. Die beiden Transistoren sind kreuzgekoppelt. Ein tiefes Potential an A<sub>1</sub> (0V) sperrt T<sub>2</sub>, dessen Kollektorpotential steigt an und öffnet T<sub>1</sub>. Steigt das Potential an A<sub>1</sub> wieder an bleibt der Zustand, gekennzeichnet durch die Potentiale +U<sub>B</sub> an B und 0V an B-bar, erhalten. Durch ein 0V-Potential an A<sub>2</sub> wird analog der zweite Zustand erreicht (B mit 0V, B-bar mit +U<sub>B</sub>).

Wir wollen für die weiteren Betrachtungen folgendes vereinbaren:

Der physikalische Zustand "hohes Potential" (entspricht etwa der Betriebsspannung +U<sub>B</sub>) entspricht dem Binärwert "1". Entsprechend wird das physikalisch "tiefe Potential" (entspricht etwa 0V) als Binärwert "0" vereinbart! (Positive Logik)

Aufgrund der demonstrierten relativ einfachen und zuverlässigen schaltungstechnischen Realisierbarkeit binärer (zweiwertiger) Zustände verwenden Rechner durchweg das Zahlensystem mit der Basis 2, das Binärsystem.

Im Binärsystem existieren entsprechend unserer Basisdefinition nur zwei unterschiedliche Ziffern, die mit "0" und "1" bezeichnet werden.

Eine Binärziffer wird als Bit bezeichnet,  
(der Name entstand durch Abkürzung des entsprechenden englischen Begriffes für Binärziffer - Binary digit)

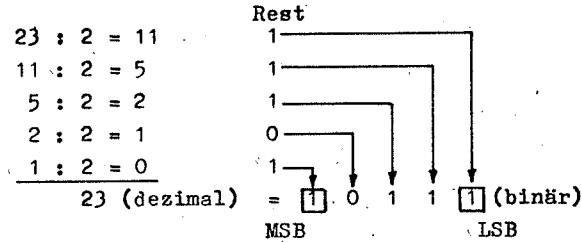
Die Handhabung binär dargestellter Zahlen bereitet anfangs ein wenig Mühe, ist aber recht schnell durch Übung erlernbar. Die Konvertierung (Umrechnung zwischen verschiedenen Zahlensystemen) von Binär- in Dezimaldarstellung kann wie folgt vorgenommen werden:

Bsp.: Binärzahl 10111

Position	4	3	2	1	0
Faktor	2 <sup>4</sup> (16)	2 <sup>3</sup> (8)	2 <sup>2</sup> (4)	2 <sup>1</sup> (2)	2 <sup>0</sup> (1)
Binärziffer	1	0	1	1	1
Produkt	16	0	4	2	1

10111 (binär) = (1 x 16) + (0 x 8) + 1 x 4 + (1 x 2) + (1 x 1)  
= 23 (dezimal)

Die Umkehrung (Dezimal - in Binärdarstellung) kann auf ähnliche Weise mit Hilfe einer solchen Tabelle erfolgen oder aber (auf schnellere Art) durch wiederholte Division durch 2 und Beachtung des auftretenden Restes. Eine gerade Zahl liefert bei Division durch 2 den Rest 0, eine ungerade Zahl den Rest 1. Aus diesen Restwerten wird die Binärzahl zusammengesetzt. Im folgenden Beispiel soll die Dezimalzahl 23 konvertiert werden:



Entsprechend der abnehmenden Wertigkeit der Bits von links nach rechts in einer Binärzahl wird das am weitesten links stehende Bit als höchstwertigstes Bit (MSB = most significant Bit) und das am weitesten rechts stehende Bit als niederwertigstes Bit (LSB = least significant Bit) bezeichnet.

1.2.2. Operationen im Binärsystem

Binäre Addition

Die binäre Addition erfolgt wie im Dezimalsystem stellenweise mit Übertragsbildung:

$$\begin{aligned}
 0 + 0 &= 0 \\
 0 + 1 &= 1 \\
 1 + 0 &= 1 \\
 1 + 1 &= 10 \quad (1 + 1 = 0 \text{ plus Übertrag})
 \end{aligned}$$

Einige weitere Beispiele mit der entsprechenden Dezimaldarstellung:

	<u>Binär</u>	<u>Dezimal</u>
0 + 0	0 = 0	0 + 0 = 0
10 + 11	101	2 + 3 = 5
101 + 11	1000	5 + 3 = 8
1000 + 1000	10000	8 + 8 = 16
111 + 10	1001	7 + 2 = 9

Logisches UND (AND)

Logische Operationen werden stets nur auf einzelne Binärziffern (Bits) einer Binärzahl angewendet. Bei Verknüpfung mehrerer Operanden werden stets nur die Bits gleicher Positionen miteinander verknüpft.

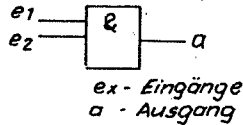


Die UND - Verknüpfung erfolgt zwischen zwei Operanden nach dieser Vorschrift:

- $0 \wedge 0 = 0$
- $0 \wedge 1 = 0$
- $1 \wedge 0 = 0$
- $1 \wedge 1 = 1$

$\wedge$  - Zeichen für UND

*Sinnbild*



D.h., nur wenn Operand 1 und Operand 2 gleich 1 sind, ist das Ergebnis 1, sonst 0.

Beispiel:  $(10111010) \wedge (01100000) = (00100000)$

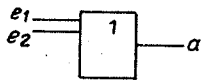
Logisches ODER (OR)

Die ODER-Verknüpfung erfolgt zwischen zwei Operanden entsprechend:

- $0 \vee 0 = 0$
- $0 \vee 1 = 1$
- $1 \vee 0 = 1$
- $1 \vee 1 = 1$

$\vee$  - Zeichen für ODER

*Sinnbild*



D.h., wenn Operand 1 oder Operand 2 oder beide gleich 1 sind, ist das Ergebnis 1, sonst 0.

Beispiel:  $(11110000) \vee (00001111) = (11111111)$

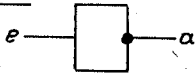
Negation (NOT)

Die Negation bewirkt die Änderung des Zustandes sämtlicher Bits eines Operanden:

- $\overline{1} = 0$
- $\overline{0} = 1$

$\overline{\quad}$  - Zeichen für Negation

*Sinnbild*



Beispiel:  $\overline{(10101010)} = (01010101)$

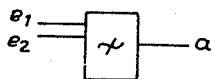
Exklusiv - ODER (XOR)

Die Exklusiv-ODER-Verknüpfung (oder auch Antivalenz) zweier Operanden ist folgendermaßen vorzunehmen:

- $0 \vee 0 = 0$
- $0 \vee 1 = 1$
- $1 \vee 0 = 1$
- $1 \vee 1 = 0$

$\vee$  - Zeichen für EX-OR

*Sinnbild*



D.h. das Ergebnis der Antivalenz ist dann gleich 1, wenn die beiden Operanden verschieden voneinander sind, sonst ist es 0.

Beispiel:  $(11001100) \vee (11000011) = (00001111)$

Weitere Operationen, wie beispielsweise Multiplikation, Subtraktion usw., werden auf die hier vorgestellten Operationen zurückgeführt. Ihre Erläuterung sowie die Darstellung negativer Zahlen erfolgt u.a. im Kapitel 9.

1.2.3. Hexadezimale Darstellung

Wie wir feststellen konnten, ist die binäre Zahlendarstellung für die Verwendung in einem Rechner bestens geeignet und wird auch angewendet. Allerdings ergibt sich bei der sehr kleinen Zahlensystembasis 2 im Binärsystem, daß die Zahlen gleichen Wertes wesentlich mehr Ziffern besitzen als beispielsweise im Dezimalsystem und damit die Übersichtlichkeit und Lesbarkeit von Binärzahlen sehr schnell mit steigenden Zahlen erschwert wird.

Z.B. ist die Binärzahl

1100111010110011

schwer lesbar und ihr etwaiger Wert kaum schnell zu berechnen. Zur Erhöhung der Übersichtlichkeit wäre die Zusammenfassung zu Bitgruppen sinnvoll, z.B. zu je 4 Bits in der Form:

1100 1110 1011 0011

Zusammenfassung zu je 3 oder 8 Bits wären ebenfalls denkbar, diese 4er-Gruppen oder Tetraden bilden aber den günstigsten Kompromiß.

Jede Tetrade kann  $2^4=16$  verschiedene Zahlen darstellen. Interpretieren wir jede mögliche Tetrade als eine Ziffer, so würden wir demnach 16 unterschiedliche Ziffern benötigen und zwar wird folgende Zuordnung festgelegt:

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

Bis zur Ziffer 9 entspricht diese Zuordnung der Dezimal-Binär-Konvertierung, für die restlichen 6 Werte werden die ersten Buchstaben des Alphabetes zu Hilfe genommen. Unsere Beispielbinärzahl würde in der neuen Schreibweise die folgende Gestalt annehmen:

1 1 0 0      1 1 1 0      1 0 1 1      0 0 1 1  
C                    E                    B                    3

Aus den 16 Binärziffern sind 4 Ziffern eines Zahlensystems mit 16 verschiedenen Ziffern d.h. der Basis 16 geworden, das als Hexadezimalsystem bezeichnet wird.

Einige Beispiele für Zahlendarstellungen im Dezimal-, Binär- und Hexadezimalsystem:

Dezimal	Binär	Hexadezimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
15	1111	F
16	10000	10
32	100000	20
64	1000000	40

Die Anwendung des Hexadezimalsystems ist im wesentlichen auf die Darstellung von Tetraden von Binärzahlen beschränkt. Es eignet sich hier ausgezeichnet zur gedrängten und übersichtlichen Darstellung und gestattet beispielsweise von den Leuchtdioden des "POLY-COMPUTERS" auf einen Blick die Binärdaten abzulesen und zu vergleichen.

Vereinbarung:  
Zur Unterscheidung zwischen dezimaler und hexadezimaler Zahlendarstellung fügen wir an Hexadezimalzahlen ein "H" an.  
Beispiel:      Hexadezimal      Dezimal  
                         15H                              15

1.3. Der Speicher eines Rechners und seine Organisation  
1.3.1. Speicherorganisation

Sämtliche Anweisungen eines Programmes und die zu verarbeitenden Daten sind als Binärzahlen dargestellt im Speicher des Rechners abgelegt.

Um diese Information gezielt speichern und wiederfinden zu können ist der Speicher in kleine, adressierbare Einheiten unterteilt. Eine solche Einheit wird als Wort bezeichnet. Ein solches Wort besteht meist aus mehreren Bits, in der Regel ein Vielfaches von 8 Bits.

Während große Datenverarbeitungsanlagen (EDVA, z.B. EC 1040) eine Wortbreite von 32 Bits und mehr aufweisen, besitzen sogenannte Minirechner meist 16 Bits Wortbreite, während heute die meisten Mikrorechner und so auch der POLY-COMPUTER 8 Bits Wortbreite aufweisen.

Aufgrund der zentralen Bedeutung der 8 Bit-Gruppe wurde für diese ein eigener Begriff geprägt, das Byte.

Ein Byte ist ein 8Bit-Wort !

z.B. 11001010 - ein Byte mit dem Wert 3AM.

Für den POLY-COMPUTER bezeichnen also die Begriffe Wort und Byte die gleichen Sachverhalte.

Jeder Speicherplatz enthält ein Wort (Byte) und besitzt eine eigene Adresse. Die Speicherkapazität wird in der Anzahl der Worte (Byte) angegeben.

Für die Adressierung des Speichers steht in jedem Rechner eine bestimmte feste Anzahl von Bits zur Verfügung. Im POLY-COMPUTER sind dies 16 Bits, d.h. es könnten maximal  $2^{16} = 65536$  Speicherplätze adressiert werden. Diese Größe nennt man Adreßraum. Demgegenüber ist die tatsächlich vorhandene Speicherkapazität wesentlich geringer.

Die Speicherkapazität wird im allgemeinen als Vielfaches von  $1024 = 2^{10}$  Bytes angegeben. Für diese Einheit wird die Bezeichnung verwendet. Beispielsweise wird eine Speicherkapazität von 2048 Bytes als 2K Bytes bezeichnet.

Ⓢ(Anzahl der vorhandenen Speicherplätze)

Anhand der folgenden Skizze werden Speicherumfang und Adresse der Grundausstattung des POLY-COMPUTER erläutert:

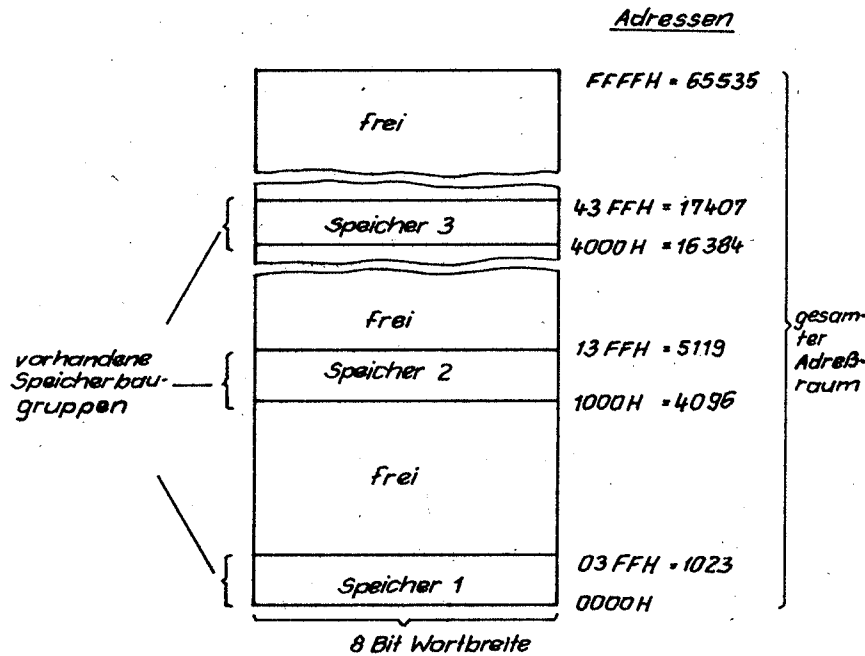


Bild 1.4. Speicherbelegung POLY-COMPUTER (Grundversion)

Die Zersplitterung der vorhandenen Speicherbaugruppen im Adreßraum ergibt sich aus der speziellen Hardwarestruktur, die eine Auffüllung der vorhandenen Lücken ohne wesentliche Änderungen zuläßt.

### 1.3.2. Speicherzugriff

Der Speicherzugriff, d.h. das Schreiben oder Lesen eines Bytes in den bzw. aus dem Speicher beginnt damit, daß die CPU die Adresse des gewünschten Speicherplatzes ausgibt. Diese Adresse erreicht den Speicher über ein Bündel von Leitungen, den Adreßbus.

Der Adreßbus ist eine Menge von Leitungen, die Adreßinformationen übertragen. Die Anzahl der Leitungen legt die Größe des Adreßraumes fest.

Der POLY-COMPUTER verfügt über einen Adreßbus von 16 Leitungen. Je nach Zustand des Adreßbusses wird aus den  $2^{16} = 65536$  Möglichkeiten mit Hilfe des Speicher-Adreßdekoders genau eine ausgewählt und der gewünschte Speicherplatz aktiviert. Jetzt kann dieses Informationsbyte, die Daten, über

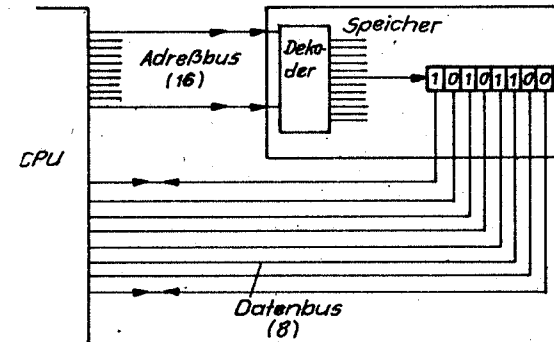


Bild 1.5. Busverbindungen CPU-Speicher

den Datenbus vom oder zum Speicher transportiert werden.

Der Datenbus ist eine Menge von Leitungen, die Dateninformationen übertragen. Die Anzahl der Leitungen legt die Wortbreite des Systems fest.

### 1.3.3. Speicherarten

Im POLY-COMPUTER werden zwei unterschiedliche Speicherarten verwendet:

- a) Schreib-Lese-Speicher sogenannte RAM (von random access memory)
- b) Lese-Speicher sogenannte ROM (von read only memory)

ROM sind Speicher, deren Inhalt (bereits bei der Herstellung programmiert) nur gelesen, nicht aber geändert werden kann.

Sie enthalten meist die Anweisungen des Programmes.

RAM sind Speicher, die sowohl lesbar als auch beschreibbar sind.

Im RAM werden z.B. Zwischenresultate von Berechnungen, sich ändernde Parameter bzw. noch zu testende Programme abgelegt. Im engeren Sinne werden unter ROM und RAM die Halbleiterspeicherschaltkreise mit den genannten Eigenschaften verstanden.

Im Bild 1.4. entsprechen Speicher 3 einem RAM und Speicher 1 und 2 einem ROM.

Schaltungstechnisch können RAMs durch Flip-Flops (s. Bild 1.2.) ROMs dagegen einfach mit Hilfe integrierter Leitungsanordnungen mit und ohne Verbindung (je nach gewünschtem Inhalt) realisiert werden.

#### 1.4. Die Struktur der Zentralen Verarbeitungseinheit (CPU)

Im POLY-COMPUTER fungiert als CPU ein einziger, hochintegrierter Schaltkreis vom Typ U880. CPUs auf einem Chip, d.h. auf einem winzigen Halbleiterplättchen werden als Mikroprozessoren bezeichnet. Der Mikroprozessor bildet den Kern eines Mikrorechners, der mit Speicher und Ein-/Ausgabeeinheiten ausgestattet, einen voll arbeitsfähigen Rechner darstellt. Der POLY-COMPUTER ist ein solcher universell verwendbarer Mikrorechner.

Die Begriffe Zentrale Verarbeitungseinheit, CPU und Mikroprozessor bezeichnen für den POLY-COMPUTER sowie für die meisten Mikrorechner den gleichen Sachverhalt.

Inzwischen haben wir bereits erfahren, über welche wesentlichen Leitungen die CPU auf den Speicher Einfluß nimmt, nämlich über Adreß- und Datenbus. Dieses Leitungsbandel verbindet die CPU auch mit den Ein-/Ausgabeeinheiten, so daß wir unser Bild der Rechnerstruktur (Bild 1.1.) vervollständigen können (Bild 1.6.).

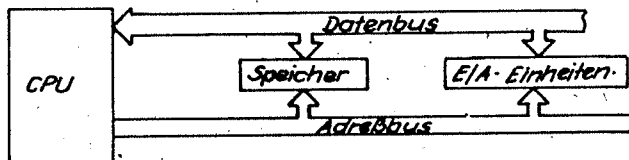


Bild 1.6. Präzisierte Rechnerstruktur

Der Adreßbus liefert Informationen nur in einer Richtung (von der CPU zu Speicher und E/A-Einheiten), er ist ein unidirektionaler Bus, während über den Datenbus sowohl von als auch zu der CPU Informationen übertragen werden. Dieser Fähigkeit wegen wird der Datenbus als bidirektionaler Bus bezeichnet.

#### 1.4.1. Die Funktionseinheiten der CPU

Eine CPU enthält im allgemeinen drei Hauptbestandteile:

- a) Arithmetik-Logik-Einheit (ALE)
- b) Befehlsdekoder und zentrale Steuerung
- c) Adresssteuerung.

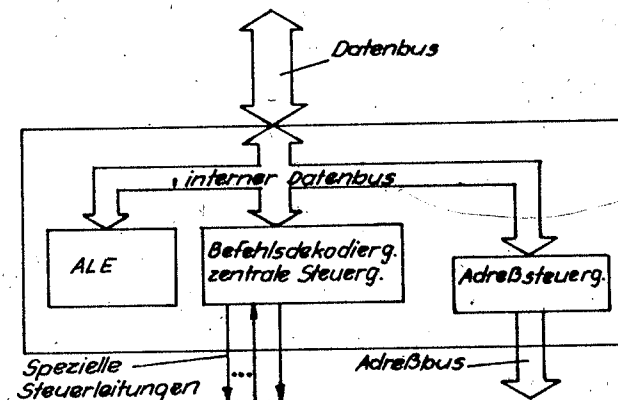


Bild 1.7. Grobstruktur CPU

Über den Datenbus gelangen Anweisungen und Daten zum Befehlsdekoder (über den CPU-internen Datenbus), werden dort interpretiert und Steuersignale für die übrigen Einheiten zur Ausführung des jeweiligen Befehls generiert. Jeder der Hauptbestandteile besitzt eine Reihe von kleinen Zwischenspeichern von ein oder zwei Byte Umfang, die Register.

Die CPU enthält Ein- oder Zweibyte-Register als Kurzzeitspeicher für verschiedenste Zwecke.

1.4.2. Die Ausführung eines Programmes

Zur Erfüllung einer bestimmten Aufgabe erhält die CPU ein vom Menschen geschriebenes Programm zur Ausführung. Ein Programm setzt sich im wesentlichen aus einer Folge von Befehlen zusammen, die in binärer Form im Speicher verfügbar sein müssen. Diese binäre Darstellung wird als Befehlskode oder nur Kode und das gesamte binäre Programm als in Maschinensprache vorliegend bezeichnet.

Ein Befehl ist die kleinste Einheit einer Rechnersprache, die den Rechner zu einer bestimmten Operation veranlaßt.

Ein Befehl seinerseits besteht aus einer Reihe sukzessive abzuarbeitender Elementarschritte.

1.4.3. Befehlszyklus

Unter dem Befehlszyklus ist das Laden und vollständige Ausführen eines Befehls zu verstehen.

Die Befehle eines Programmes sind im Speicher abgelegt. Daher ist als erste Aktion der Transport des Befehles aus dem Speicher in die CPU zu veranlassen. In der CPU ist der Befehl zu dekodieren und auszuführen. Darin eingeschlossen ist die Berechnung der Adresse des nächsten abzuarbeitenden Befehls. Aus diesen drei Aktivitäten setzt sich der Befehlszyklus zusammen. Die Länge des Befehlszyklus variiert z.T. erheblich je nach auszuführender Operation.

1.4.4. Der Programmzähler (PC) und das Befehlsregister

Die für das Lesen eines Befehlskodes aus dem Speicher erforderliche Adresse ist in einem CPU-Register, dem Programmzähler (PC) enthalten.

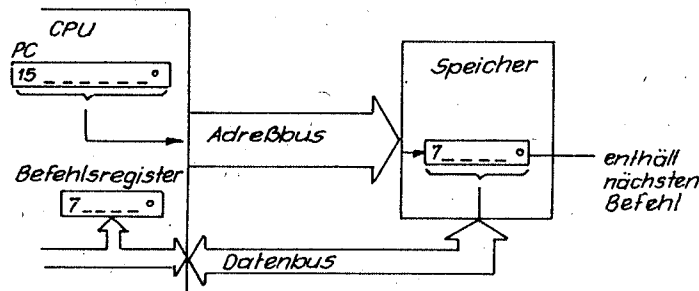


Bild 1.8. Laden eines 8Bit-Befehlskodes

Dazu wird der Inhalt des PC auf den Adreßbus geschaltet (PC ist 16 Bit-Register) und der Speicherplatz aktiviert, der den nächsten Befehl enthält. Dieser gelangt dann über den Datenbus zur CPU in das Befehlsregister. Der Programmzähler wird durch die Adressensteuerung anschließend auf seinen neuen Wert eingestellt.

Das Befehlsregister enthält den gerade auszuführenden Befehl.

Nach dem Laden des aktuellen Befehls in das Befehlsregister gelangt dieser zum Befehlsdekor, der aus dem Kode eine Vielzahl binärer Signale zur zeitlichen und inhaltlichen Steuerung der für die Befehlsausführung erforderlichen Schritte generiert.

1.4.5. Der Akkumulator (A)

Das wesentlichste Register der Arithmetik-Logik-Einheit ist der Akkumulator (auch als Register A bezeichnet). Bei den meisten arithmetischen und logischen Befehlen spielt er eine wichtige Rolle. Er enthält einen der zu verknüpfenden Operanden und in ihm wird anschließend das Ergebnis der Operation gespeichert.

1.5. Erste Versuche mit dem POLY-COMPUTER

Nachdem wir eine ganze Reihe wichtiger Fakten über Funktion und Aufbau von Rechnern und deren Baugruppen kennengelernt haben, können wir erste praktische Versuche mit dem Lernsystem unternehmen. Zunächst jedoch noch einige Bemerkungen zum Betriebsprogramm des POLY-COMPUTERS und dessen Bedienung.

1.5.1. Der Monitor

Der POLY-COMPUTER 880 ist mit einer CPU, Speicher (in der Grundversion 2 K Bytes ROM, 1 K Bytes RAM) sowie drei E/A-Einheiten zur Bedienung und Testung (Tastatur, Display, Busanalysator) ausgerüstet. Diese Hardware wird von einem Betriebsprogramm, das im ROM untergebracht ist, ergänzt. Es ermöglicht erst eine sinnvolle Handhabung des Lernsystems und enthält wichtige Komponenten zur Unterstützung beim Erlernen der Mikrorechnerprogrammierung und -anwendung. Dieses Betriebsprogramm wurde vom Hersteller im Lesespeicher (ROM) untergebracht und beginnt, mit dem Anschließen des POLY-COMPUTERS an das Netz zu arbeiten.

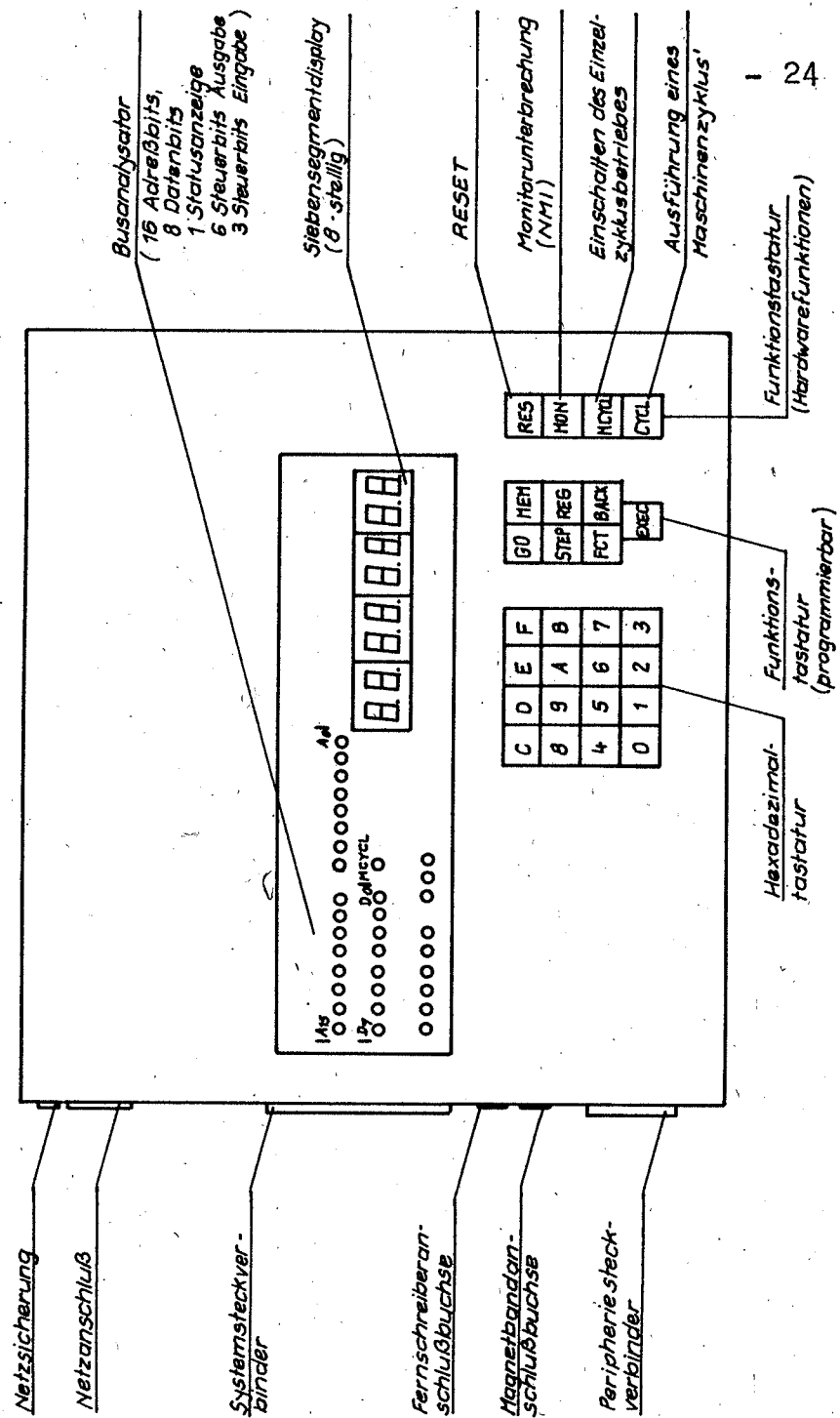


Bild 1.9. Bedien-, Anzeige- und Anschlußelemente des POLY-COMPUTERS

Ein Betriebsprogramm dieser Art wird als Monitor bezeichnet. Der vorliegende Monitor bedient die E/A-Einheiten (Tastatur und Display), gestattet Kontrolle und Änderung von Speicherinhalten und CPU-Registern, erlaubt den Start und die schrittweise Abarbeitung von Anwenderprogrammen und vieles andere mehr.

Während wir anfangs ausschließlich die Programme des Monitors zur Bedienung der E/A-Einheiten verwenden wollen, werden wir später in der Lage sein, eigene Programme für diesen Zweck zu schreiben.

### 1.5.2. Zur Bedienung des POLY-COMPUTERS

Die Tasten- und Anzeigegruppen des POLY-COMPUTERS zeigt Bild 1.9. Nach dem Anschluß des Gerätes an das 220V~ Netz erscheint auf dem Display die Ausschrift

POLY-880

und signalisiert die Betriebsbereitschaft. Aus sämtlichen Betriebszuständen, z.B. auch bei Eingabefehlern, gelangen wir durch die Taste **RESET** wieder in diesen Anfangszustand.

Für die Anzeige von Ziffern und einigen Buchstaben wird das 8-stellige Display verwendet, dessen Stellen aus 7 Balken und 1 Punkt gebildet werden (Bild 1.10.). Durch geeignete Ansteuerung von 7 Leuchtelementen (daher die Bezeichnung 7-Segmentanzeige)



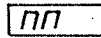
Bild 1.10. Eine Displaystelle lassen sich sämtliche Hexadezimalziffern sowie mit ein wenig Phantasie auch fast alle Buchstaben des Alphabetes darstellen und ablesen. Die folgende Tabelle zeigt alle im Monitor verwendeten Zeichen und ihre Darstellung als 7-Segment, wobei zur Unterscheidbarkeit z.T. Kleinbuchstaben Ver-

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
G	H	I	L	M	O	P	R	S	T	U	X	Y			
G	H	I	L	M	O	P	R	S	T	U	X	Y			

wendung fanden.

Die 8 Displaystellen werden vom Monitor fest eingeteilt (eine Ausnahme bildet Schriftausgabe bei Kommentaren). Die ersten beiden linken Elemente zeigen eine Abkürzung des gerade gewählten Kommandos.

Betätigen wir beispielsweise die Taste MEM, (entspricht dem Kommando "Anzeige und Änderung des Speicherinhaltes") so erscheint auf diesen Displaystellen die Ausschrift



als Abkürzung für Memory (Speicher). Die nächsten vier Anzeigestellen werden in den meisten Fällen für die Anzeige der aktuellen 16 Bit-Adresse in hexadezimaler Form verwendet.

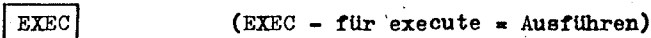
Wir geben dazu über die 16 Tasten umfassende Hexadezimaltastatur (siehe Bild 1.9.) die Ziffern



ein. Bei der Tastenbetätigung erscheint jede Ziffer zunächst in der rechten der vier Adreßstellen und wird bei jeder folgenden Ziffer um eine Stelle nach links gerückt. Sollte uns bei der Eingabe ein Fehler unterlaufen sein, so wiederholen wir einfach die Zifferneingabe solange, bis die gewünschte Zahl in der Anzeige erscheint.

Die noch verbleibenden zwei rechten Anzeigeelemente dienen zur Darstellung eines 8 Bit-Datenmusters (entsprechend zwei Hexaziffern).

Zur Demonstration betätigen wir jetzt die Kommandotaste



Diese Taste dient zur Bestätigung und zum Start sämtlicher Kommandos. Vor dem Drücken dieser Taste besteht in jedem Falle die Möglichkeit, die Kommandoparameter (z.B. Adressen) noch zu ändern bzw. ein anderes Kommando zu wählen. In unserem Beispiel bewirkt die Taste EXEC die Ausführung des Kommandos "Anzeige des Speicherplätzinhaltes 4000H". Diesen können wir jetzt auf den zwei rechten Anzeigeelementen ablesen.

Die Adresse 4000H ist unsere erste RAM-Adresse (siehe Bild 1.4.). Der Inhalt des RAM ist nach jedem Einschalten des Gerätes unbestimmt, da die Flip-Flops nach Ab- und Einschalten der Betriebsspannung eine zufällige Lage einnehmen.

Bei jedem weiteren Betätigen der Taste EXEC

wird die Adresse um 1 erhöht und der entsprechende Speicherinhalt angezeigt.

Bild 1.11. zeigt die eben erläuterte Aufteilung des Displays im Überblick.

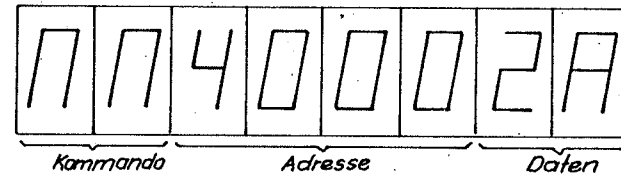


Bild 1.11. Aufteilung des Displays am Beispiel Speicheranzeige

Eine Abweichung von der Aufteilung im Bild 1.11. tritt lediglich bei der Anzeige von CPU-Registern auf. Diese werden prinzipiell als Doppelregister (16 Bit = 4 Hexastellen) angezeigt, so daß folgende Aufteilung zustande kommt: (Bild 1.12.)

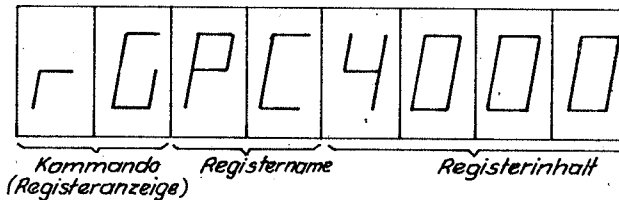


Bild 1.12. Aufteilung des Displays am Beispiel Registeranzeige

Bild 1.12. stellt einen möglichen Displayinhalt bei Anzeige des Programmzählers (PC) dar.

Zur Anzeige des Programmzählers wählen wir das Kommando "Anzeige und Änderung Register". Dieses Kommando wird mit der Taste



aufgerufen. Danach erscheint die Kommandoanzeige PC

Mit einer Taste der Hexastatur, die z.T. zwei- und dreifach belegt sind, wird nun das gewünschte Register angewählt.

Wir drücken die Taste 

C
PC

woraufhin der Registername 

rGPC
------

 auf dem Display erscheint.

Nach Ausführung des Kommandos mit der Taste 

EXEC
------

 erscheint schließlich der Programmzählerinhalt auf den 4 rechten Anzeigeelementen.

1.5.3. Zusammenfassung

Am Ende dieses und jedes der folgenden Abschnitte wollen wir die wichtigsten neuen Fakten kurz nochmals zusammenfassen.

Bedienung POLY-COMPUTER:

- Ausschrift bei Anschluß des Gerätes an das Netz:

Anzeige: 

POLY-880
----------

- Einstellung dieses Anfangszustandes (z.B. bei groben Eingabefehlern oder fehlender Displayanzeige)

Taste: 

RESET
-------

 (=Rücksetzen in Anfangszustand)  
Anzeige: 

POLY-880
----------

- Anzeige von Speicherinhalten

Kommando: Anzeige und Änderung von Speicherinhalten

Taste: 

MEM
-----

 Anzeige: 

nn
----

Eingabe der gewünschten Taster: Adresse über Hexastatur:

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

z.B.

nn4020.
---------

(beliebig oft wiederholbar)

Ausführung:

Taste: 

EXEC
------

Anzeige: 

nn4020 XX.
------------

Jedes weitere 

EXEC
------

 bewirkt Erhöhung der Adresse 1

und Anzeige des neuen Speicherinhaltes z.B.

- Anzeige des Programmzählers

nn4021 XX.
------------

Kommando: Anzeige und Änderung von Registerinhalten

Taste: 

REG
-----

Anzeige: 

rG
----

Wahl des Registers

Taste: 

C
PC

Anzeige: 

rGPC
------

Ausführung

Taste: 

EXEC
------

Anzeige: 

rGPC XXXX.
------------

 z.B.

Bei für den Autor nicht vorhersehbaren Anzeigewerten (z.B. RAM-Inhalt nach dem Neueinschalten des Gerätes) wird in diesem Lehrmaterial das Zeichen "X" eingetragen. Beim Anwender kommen natürlich zufällige Zahlen zur Anzeige.

1.6. Das erste eigene Programm

Zunächst wollen wir uns aus der Fülle der vom Mikroprozessor U880 angebotenen Befehle (siehe "Systemhandbuch POLY-COMPUTER 880") einige einfache beispielhaft herausgreifen, sie zu einem Programm zusammenstellen, dieses über die Tastatur in den RAM einschreiben und zur Ausführung bringen. Neben dem auch für den Rechner verständlichen Befehlscode, dargestellt als Binär- oder Hexawert, werden sämtliche Prozessorbefehle durch eine Abkürzung von 2, 3 oder 4 Buchstaben, dem sogenannten mnemonischen Code oder kurz Mnemonik bezeichnet. Dieses Mnemonik erhöht für den Programmierer wesentlich die Verständlichkeit seines Programmes, denn es ist eine verkürzte Darstellung der Bedeutung des Befehles.



1.6.1. Einige Prozessorbefehle

Der erste Befehl, den wir kennenlernen, löscht den Inhalt des Akkumulators (Register A).

Mnemonic:	XOR A	(Abkürzung für "exclusive or") siehe Abschnitt 1.2.2.
Befehlskode:		
binär	1 0 1 0 1 1 1 1	
hex	AF	
Wirkung:	Löscht den Inhalt des Akkumulators (Setzt alle 8 Bits auf 0)	

Aus diesem ersten Befehlsbeispiel ist bereits der Aufbau des Formates erkennbar. Dem Mnemonik für den Befehl (XOR) folgt in einem gewissen Abstand die Angabe, auf welches Register er anzuwenden ist (A).

Die nächsten beiden Befehle erhöhen bzw. verringern den A-Registerinhalt um eins.

Mnemonic:	INC	A
Befehlskode:		
binär	0 0 1 1 1 1 0 0	
hex	3C	
Wirkung:	Inkrementiert das A-Register. (Addiert zum A-Registerinhalt 1 dazu)	

Mnemonic:	DEC	A
Befehlskode:		
binär	0 0 1 1 1 1 0 1	
hex	3D	
Wirkung:	Dekrementiert das A-Register (Subtrahiert vom A-Register 1)	

Mit diesen drei Befehlen wollen wir ein einfaches Programm formulieren.

1.6.2. Schreiben des Programmes

Beim Schreiben eines Programmes sollten wir von anfang an einige methodische Hinweise beherzigen.

Zunächst sollte jedes Programm mit einer ausreichend ausführlichen Definition beginnen, die Aussagen über die Funktion des Programmes macht, und es sollte einen Namen erhalten.

Z.B. soll unser erstes Programm folgendes ausführen:

"Das Programm "ZAEHLEN" soll das A-Register auf den Wert Null setzen und durch 6-faches Inkrementieren den Wert 6 in das A-Register bringen. Anschließend soll durch zweimaliges Dekrementieren der Wert 4 im A-Register erscheinen".

Für ein Programm von diesem geringen Umfang können wir ohne Zwischenschritte direkt zum Aufschreiben der Befehlsfolge übergehen. Wir wollen uns dazu unser Blatt in vier Spalten einteilen, in die von links beginnend die Speicheradresse des jeweiligen Befehles, die hexadezimale Darstellung des Befehlskodes, das Mnemonik des Befehls und ein Kommentar geschrieben werden. Da unser Programm im RAM des POLY-COMPUTERS abgespeichert und ausgeführt werden soll, lassen wir es ab Adresse 4000H beginnen.

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	AF	ZAEHLER: XOR A	Lösche das A-Register
4001	3C	INC A	Inkrementiere A 6 mal
4002	3C	INC A	
4003	3C	INC A	
4004	3C	INC A	
4005	3C	INC A	
4006	3C	INC A	
4007	3D	DEC A	A enthält jetzt Wert 6 Dekrementiere A 2 mal
4008	3D	DEC A	A enthält jetzt Wert 4

Die Spalten für Mnemonik und vor allem für Kommentar sind etwas breiter als die anderen Spalten anzulegen. Der Programmname wird vor den ersten Programmbefehl im Mnemonikfeld geschrieben. Es empfiehlt sich, den Kommentar tatsächlich derart ausführlich vorzusehen und zwar aus zweierlei Gründen. Erstens versteht man damit auch nach Wochen und Monaten noch ohne längere Einarbeitungszeit das Programm, und zweitens macht sich diese Verfahrensweise vor allem dort notwendig, wo ein Kollektiv von Programmierern an einer Aufgabe tätig ist und ein zweiter sich in das Programm des Mitarbeiters einlesen muß.

1.6.3. Abspeichern (Laden) des Programmes im POLY-COMPUTER

Zum Laden des Programmes in den RAM des Rechners führen wir zunächst ein Rücksetzen in den Grundzustand aus,

Taste:  Anzeige:

und geben das Kommando "Anzeige und Änderung des Speicherinhaltes" durch

Taste:  Anzeige:

Jetzt geben wir über die Hexatastatur (Bild 1.9.) die Anfangsadresse unseres Programms ein

Tasten:   
  
  
 Anzeige:

Durch Betätigen der Taste

Anzeige:

wird der zufällige Inhalt des Speicherplatzes 4000H angezeigt.

Laut Programm müssen wir den Kode AFH auf diesen Platz transportieren. Dazu geben wir diesen Kode über die Hextastatur ein:

Taste:   
 Anzeige:

Mit Drücken der Taste

Anzeige:

wird das angezeigte Datenwort AFH auf Speicherplatz 4000H geschrieben und anschließend die um eins erhöhte Adresse und deren Inhalt angezeigt.

Jetzt kann der zweite Programmbefehl über die Hextastatur eingegeben werden:

Taste:   
  
 Anzeige:

Auf gleiche Weise erfolgt die Eingabe der restlichen Befehle bis zur Adresse 4008H.

Letzte Eingabe:

Taste:   
  
 Anzeige:

Zur Kontrolle der Eingabe lassen wir uns die Speicherinhalte von 4000H bis 4008H noch einmal anzeigen.

Taste:   
  
  
  
 Anzeige:   
  
  
  
usw. ( 8 mal  ).

1.6.4. Ausführung des Programmes

Nachdem wir uns von der Richtigkeit der Befehlskodes im Speicher überzeugt haben, wollen wir an die Ausführung und Testung unseres Programmes gehen. Für den Test von Programmen besitzt der POLY-COMPUTER zwei besondere Betriebsarten, Einzelbefehlsbetrieb und Einzelzyklusbetrieb. Damit sind wir in der Lage, die Programmabarbeitung nach jedem Befehl bzw. Befehlssteil anzuhalten und die Wirkung eines Befehles oder sogar einzelner Befehlssteile auf den Speicher, die CPU-Register oder E/A-Einheiten zu überprüfen.

Ohne diese Möglichkeiten des Lernsystems würde jedes Programm vollständig mit der sehr hohen Rechengeschwindigkeit (beim POLY-COMPUTER sind das mehr als 100 000 Befehle pro Sekunde !) abgearbeitet, womit eine Beobachtung einzelner Befehlswirkungen und ein effektiver Programmtest nicht mehr möglich ist.

Das Kommando "Ausführung eines Befehls" wird mit

der Taste  aufgerufen.

Für die Testung unseres Programmes verwenden wir jetzt dieses Kommando.

Da bereits der erste Befehl des Programmes das A-Register beeinflusst, lassen wir uns zunächst dessen Inhalt anzeigen. Kommando "Anzeige und Änderung Register"

Taste:  Anzeige:

Auswahl Register A

Ausführung:

Entsprechend Bild 1.12. werden beim Kommando "Anzeige und Änderung Register" stets zwei Bytes (4 Hexastellen) Registerinhalt auf den rechten vier Displaystellen angezeigt. Gemeinsam mit dem Register A (3. und 4. Stelle von rechts) wird auf den rechten zwei Stellen der Inhalt des noch später zu erläuternden Registers F angezeigt.

Wir setzen das A-Register folgendermaßen auf den Wert FFH vor:

Eingabe von FFH über Hexatastatur:

Taste:  Anzeige:

Einspeichern:

(Bemerkung: Der Punkt auf dem Display steht immer rechts von den zwei im Moment beeinflussbaren Hexastellen. Nach EXEC rückt er jeweils zwei Stellen weiter.)

Jetzt beginnen wir die Programmausführung mit dem ersten Befehl auf der Adresse 4000H.

Kommando "Ausführung eines Befehls"

Taste:  Anzeige:

Eingabe der Befehlsadresse (über Hexatastatur)

Ausführung eines Befehls:

Der Befehl XOR A (lösche A-Register) wurde ausgeführt und auf dem Display erscheinen Adresse und Befehlscode des nächsten Befehles.

Um die Wirkung des ausgeführten Befehles zu überprüfen, lassen wir uns den Inhalt des A-Registers anzeigen.

Kommando "Anzeige und Änderung Register"

Auswahl A-Register:

Ausführung:

Register A enthält erwartungsgemäß den Wert 00H.

Der Programmzähler wird durch diese Registeranzeige nicht beeinflusst, so daß wir ohne explizite (ausdrückliche) Angabe der Adresse des nächsten Befehles folgendermaßen die Programmausführung festsetzen können.

Kommando "Ausführung eines Befehls"

Ausführung:

Wir führen noch die restlichen INC A-Befehle bis einschließlich

EXEC  
EXEC  
EXEC  
EXEC

5E40043C  
5E40053C  
5E40063C  
5E40073d

Sollten sich während dieser Befehlsabarbeitung andere als die angegebenen Displaywerte ergeben bzw. die Anzeige völlig löschen, so wurden wahrscheinlich Fehler bei der Programmeingabe gemacht. Diese müßte entsprechend Abschnitt 1.6.3. wiederholt werden.

Nach diesem sechsmaligen Inkrementieren enthält A den Wert sechs. Überprüfung:

REG  
0  
AF  
EXEC

rG  
rGARF  
rGARF06.XX

Die letzten beiden Befehle unseres Programmes (siehe Programm-  
liste) subtrahieren jeweils eins von Register A.

STEP  
EXEC  
EXEC

5E  
5E40083d  
5E4009XX

Im Register A verbleibt der Wert vier.

REG  
0  
AF  
EXEC

rG  
rGARF  
rGARF04.XX

Unser Programm wurde vollständig und richtig ausgeführt.

1.6.5. Zusammenfassung

In diesem Abschnitt wurden folgende neue Monitorkommandos eingeführt:

(a) Kommando "Anzeige und Änderung Speicher".

Ablauf:

Kommando

MEM

Adreßeingabe:  
(Hex)  
(z.B. 4000H)

Anzeige:

EXEC

Eingabe neuer Wert: (z.B. AFH)

Einspeichern

EXEC

Eingabe neuer Wert: (z.B. 3CH)  
(hex)

Einspeichern: EXEC

**Achtung!** Für alle Monitorkommandos gilt:

Ein Kommando wird erst nach dem Betätigen der Taste **EXEC** tatsächlich ausgeführt, d.h. das Display zeigt lediglich an, welches Kommando mit welchen Parametern als nächstes zur Ausführung kommt. Bei den Kommandos **MEM** und **REG** wird nach **EXEC** auf die folgende Speicheradresse bzw. das nächste Register weitergeschaltet, so daß z.B. eine fortlaufende Programmeingabe oder Registerinitialisierung (-voreinstellung) vereinfacht wird. Das Kommando "Zurück",

Taste **BACK**, entspricht bei der Anzeige und Änderung von Registern und Speichern (**MEM**, **REG**) der Wirkung der Taste **EXEC** (Ausführen des gewählten Kommandos), jedoch wird anschließend auf die vorangehende Speicheradresse bzw. das vorher-

gehende Register weitergeschaltet. Dazu ein Beispiel:

Kommando: MEM

Adresse:

Ausführung: EXEC

Weiterschalten der Adresse (und Abspeichern des angezeigten Wertes): EXEC

Zurückschalten der Adresse (und Abspeichern des angezeigten Wertes): BACK

Wird beim Kommando "Anzeige und Änderung von Speicher" keine Adresse angegeben, so wird automatisch die erste RAM-Adresse eingesetzt.

MEM   
EXEC

(b) Kommando "Anzeige und Änderung Register"

Dieses Kommando wirkt einheitlich auf Doppelregister, d.h. es werden entweder zwei 8 Bit-Register gleichzeitig (z.B. AF) oder ein 16 Bit-Register (z.B. PC) angezeigt.

Ablauf: REG

Auswahl des Registers über zweite Bedeutung der Hexatasten:

Z.B. (Register BC)

EXEC

Die zwei links vom leuchtenden Punkt befindlichen Stellen sind jetzt über die Hexatastatur änderbar (sie entsprechen dem höherwertigen Teil des jeweiligen Doppelregisters).  
Z.B. mit

EXEC   
 EXEC   
EXEC

wird der Anzeigewert dieser beiden Stellen in das Register eingeschrieben, der leuchtende Punkt wandert zwei Stellen nach rechts und erlaubt die Änderung der nächsten zwei Stellen (niederwertiges Bytes des Doppelregisters),

Z.B.    
   
Mit EXEC

werden auch diese beiden Stellen in BC eingeschrieben und das Display bringt jetzt das nächste Doppelregister zur Anzeige

Über BACK

kann ähnlich wie beim Speicherkommando die Reihenfolge der angezeigten Register umgekehrt werden.

Fehlt die Angabe eines Registernamens wird die Anzeige mit AF begonnen.

REG   
EXEC

(c) Kommando "Ausführung eines Befehls"

Ablauf: STEP

Eingabe Befehlsadresse über Hexatastatur:

z.B.

Ausführung: EXEC

Der auf Speicherplatz 4025H befindliche Befehl wird ausgeführt und danach der neue Programmzählerinhalt (xxxx) und der dazu gehörige Befehlscode (yy) angezeigt.

Wird die Angabe der Befehlsadresse weggelassen, kommt der Befehl zur Ausführung, dessen Adresse im Programmzähler durch vorangegangene Manipulationen erzeugt wurde.

1.7. Schlußwort zum ersten Kapitel

Das zurückliegende Kapitel brachte wesentliche Grundlagen zum Verständnis von Hard- und Software eines Rechners. Das Verständnis dieser Fakten und Erläuterungen ist eine Voraussetzung, um den Ausführungen der nächsten Kapitel folgen zu können.

Nach dem Studium des ersten Kapitels sollte der Leser in der Lage sein, folgende Fragen beantworten zu können:

1. Was ist ein Mikrorechner?
2. Was ist ein Mikroprozessor?
3. Welche Hauptbestandteile besitzen Mikrorechner und Mikroprozessor?
4. Was ist Software? Was ist ein Programm?
5. Was ist Hardware?
6. Wie sieht die rechnerinterne Zahlen- und Befehlsdarstellung aus?
7. Was verstehen wir unter der Basis eines Zahlensystems?
8. Ist die Aussage  $27AC H = 10156$  wahr?
9. Wie sind die Operationen UND, ODER, Negation und Exklusiv-ODER definiert?
10. Was verstehen wir unter den Begriffen Bit und Byte?
11. Was ist ein Rechner-Bus?
12. Welche zwei Speicherarten unterscheiden wir beim Mikrorechner und worin unterscheiden sie sich?
13. Was ist ein Befehl?
14. Was ist ein Befehlszyklus?
15. Welche Aufgaben erfüllen die Register PC und A?

## 2. Ausgewählte Befehle und einige Flags der CPU U880

### 2.1. Mehrbytebefehle

Sämtliche bisher kennengelernten Befehle (XOR, INC, DEC) besitzen eine Länge von einem Byte. Befehle dieses Umfanges wirken meist nur auf die internen Prozessorregister (mit einigen Ausnahmen). Sollen aber beispielsweise mit einem Befehl Daten aus dem Speicher verarbeitet werden (damit ist ein Transport vom o. zum Speicher unter Angabe einer Adresse verbunden), so muß der Befehl zusätzliche Informationen enthalten. Das sind entweder

- (a) zwei Zusatzbytes, die die Adresse des interessierenden Speicherplatzes enthalten oder
- (b) ein oder zwei Zusatzbytes, die die zu verarbeitenden Datenbytes selbst darstellen.

Die Adressierung eines Speicherplatzes durch Angabe der Adresse innerhalb des Befehles wird als direkte Adressierung bezeichnet.

Die innerhalb eines Befehles auftretenden Datenbytes werden als Direktwerte bezeichnet.

Der Befehlsaufbau sieht bei der CPU U880 demnach so aus, daß das erste Byte des Befehles darüber entscheidet, welche Operation auszuführen ist (also den eigentlichen Befehlskode bzw. den Befehlstyp enthält), während eventuell nötige Befehlsparameter (Direktwerte, Adresse) in ein oder zwei weiteren zum Befehl gehörigen Bytes angegeben werden.

Mit einem Byte (8 Bits) Befehlskode lassen sich allerdings nur maximal  $2^8 = 256$  Befehlsvarianten kodieren. Der Prozessor U880 besitzt wesentlich mehr als 256 Befehlsvarianten, so daß bei einigen Befehlsarten zwei statt nur ein Byte Befehlskode auftreten, so daß wir feststellen können:

Ein Befehl der CPU U880 kann ein, zwei, drei oder vier Bytes enthalten!

Beispiele:

Typische Vertreter der Zweibytebefehle sind die Direktwertbefehle

Mnemonic:	ADD n	(n - Direktwert 8 Bits, z.B. Hexazahl, ADD steht für Addition)
Befehlskode: binär	11000110 ←-n-→	
hex	C6 n	
Wirkung:	Addiert zum A-Registerinhalt den Wert n hinzu	

Dreibytebefehle enthalten neben einem Befehlskodebyte meist zwei Bytes Adresse.

Mnemonic:	LD A, (nn)	(nn - 16 Bit-Adresse)
Befehlskode: binär	00111010 ←-n-→ ←-n-→	
hex	3A n n	
Wirkung:	Lädt das A-Register mit dem Wert, der im Speicher auf Adresse nn steht.	

Vergleichen wir die beiden letzten Befehle, so erkennen wir den Unterschied in der Schreibweise für Adressen und Direktwerte. Wir wollen folgende Vereinbarungen treffen:

Adressen werden in einem Befehl durch runde Klammern eingeschlossen; Direktwerte werden ohne Klammern angegeben. Mehrere Operanden werden durch Komma getrennt.

z.B. LD A, (410AH )  
bedeutet: Lade in das A-Register den Inhalt des Speicherplatzes 410AH, während  
  
LD A, 56H  
das Laden der Zahl 56H in das A-Register veranlaßt.

Mnemonic:	LD (nn), A	(nn-16 Bit-Adresse)
Befehlskode: binär	00110010 ←-n-→ ←-n-→	
hex	32 n n	
Wirkung:	Speichert den Inhalt des A-Registers auf den Speicherplatz mit der Adresse nn	

Sämtliche Befehle mit dem Mnemonik LD (für lade) vollziehen einen Transport von einer Quelle zu einem Ziel. Somit sind stets zwei Operanden anzugeben. Allgemein gilt für die Richtung dieses Transports:

In Befehlen mit zwei Operanden gibt der erste das Ziel, der zweite die Quelle der Daten an.

z.B. LD A, (410AH)  
Richtung: von Speicherplatz 410AH zum A-Register  
  
LD (410AH), A  
Richtung: von A-Register zum Speicherplatz 410AH.

Beispiele für Vierbytebefehle werden nach Behandlung der Universalregister der CPU U880 angegeben.

Am Schluß dieses Abschnittes wollen wir die neuen Befehle in einem Programmbeispiel anwenden.

Beispielprogramm mit Mehrbytebefehlen:

Das Programm "A D D I T I O N" soll zunächst das Register A löschen (auf 00H setzen!), danach die Hexzahlen 05H und 02H addieren und das Ergebnis auf die Speicherzelle 4100H transportieren.

Auch dieses Programm lassen wir auf dem ersten Platz unseres verfügbaren RAM-Speichers, also auf Adresse 4000H beginnen.

Adresse	Befehls- kode	Mnemonic	Kommentar
4000	AF	ADDITION: XOR A	Lösche A-Register A:=0
4001	C6	ADD 05H	Addiere zu A den Wert A:=A+5
4002	05		05H
4003	C6	ADD 02H	Addiere zu A den A:=A+2
4004	02		Wert 02H
4005	32	LD (4100H), A	Transportiere das Ergebnis
4006	00		(4100H):=A auf Platz 4100H
4007	41		

Die Testung des Programmes im POLY-COMPUTER erfolgt entsprechend den Ausführungen in den Abschnitten 1.6.3 und 1.6.4. D.h. zunächst werden die in der Spalte Befehlskode enthaltenen Bytes ab Adresse 4000H in den RAM eingegeben (MEM) und anschließend das Programm im Einzelbefehlsbetrieb ausgeführt (STEP). Eine wesentliche Tatsache ist noch zu berücksichtigen:

Bei Befehlen der CPU U880, die 16 Bit-Direktwerte oder Adressen enthalten, steht das niederwertige Byte auf der niedrigeren Adresse, das höherwertige Byte auf der nächsthöheren Adresse im Speicher.

Dementsprechend wird im obigen Programmbeispiel "A D D I T I O N" im Befehl LD (4100H),A die 16 Bit-Adresse 4100H so aufgeteilt, daß der niederwertige Teil (00H) auf Adresse 4006H, der höherwertige Teil (41H) auf Adresse 4007H abgespeichert wird.

Bei der Ausführung des Programmes "A D D I T I O N" ist nach jedem "STEP" der A-Registerinhalt zu kontrollieren (REG); es enthält nach dem dritten Befehl den Wert 07H. Mit dem Kommando "MEM" kann schließlich der Inhalt des Speicherplatzes 4100H nach dem letzten Befehl überprüft werden.

## 2.2. Befehlszyklus und Maschinenzklus

Ein U880-Befehl kann aus ein bis vier Bytes bestehen, d.h. allein zum Laden des Befehls in das Befehlsregister der CPU können bis zu vier Speicherzugriffe auf adreßmäßig benachbarte Speicherplätze nötig sein. Darüber hinaus veranlassen bestimmte Befehle die CPU zu weiteren Zugriffen zu Speicher oder Peripherie. Z.B. beinhaltet der Befehl LD (4000H),A, der ab Adresse 4005H gespeichert ist, insgesamt vier Speicherzugriffe bzw. Zyklen:

1. Zyklus: Laden erstes Befehlskodebyte (32H) von Adresse 4005H
2. Zyklus: Laden des niederwertigen Adreßbytes (00H) von Adresse 4006H
3. Zyklus: Laden des höherwertigen Adreßbytes (41H) von Adresse 4007H
4. Zyklus: Abspeichern des A-Inhaltes auf Adresse 4100H.

Die CPU-Aktivitäten für einen Speicherzugriff bzw. einen Ein-/Ausgabekanalzugriff werden als Maschinenzklus (oder nur Zyklus) bezeichnet. Ein Befehlszyklus kann aus ein bis sechs Maschinenzyklen bestehen.

### 2.2.1. Wichtige Zyklusarten und Steuersignale der CPU U880 - Der Steuerbus

Je nach dem Ziel eines CPU-Zugriffs unterscheiden wir verschiedene Maschinenzklusarten, von denen wir die wichtigsten jetzt kennenlernen wollen. Zur Unterscheidung liefert die CPU eine Reihe von Steuersignalen, deren Gesamtheit Steuerbus genannt wird. Dieser Steuerbus wird bzw. ausgewählte Signale daraus werden an die übrigen Rechnerbaugruppen ebenso angeschlossen, wie wir das von Adreß- und Datenbus her kennen. Die weiter präzisierete Rechnerstruktur zeigt Bild 2.1.

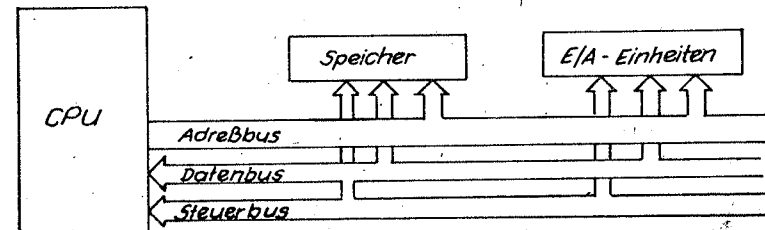


Bild 2.1. Präzisierete Rechnerstruktur

Einige ausgewählte Steuersignale der CPU U880:  
(Die Steuersignale sind im tiefen Zustand aktiv, d.h. ca. 0V aktiver Zustand, größer 2,4V inaktiver Zustand. Diese "Negation" wird durch einen Strich über der Signalbezeichnung gekennzeichnet)



- M1:** (maschine cyde 1 - Maschinenzklus 1) zeigt an, daß ein Befehlskode (erstes bzw.erstes oder zweites Befehlsbyte) vom Speicher geladen wird.
- MREQ:** (memory request - Speicherzugriff) zeigt an, daß in diesem Zyklus zum Speicher zugegriffen wird.
- IORQ:** (input/output request - E/A-Zugriff) zeigt an, daß in diesem Zyklus zu Peripherieeinheiten zugegriffen wird.
- RD:** (read-Lesen) zeigt an, daß in diesem Zyklus gelesen wird; d.h. Datentransport zur CPU.
- WR:** (write-Schreiben) zeigt an, daß in diesem Zyklus geschrieben wird; d.h. Datentransport von der CPU zu Speicher, E/A-Geräte o.a.

Einige ausgewählte Zyklusarten der CPU U880:

Befehlskode-Ladezyklus (M1-Zyklus)

- Kennzeichen:** - Adreßbus enthält Adresse des zu ladenden Befehlskodebytes von CPU  
 - Datenbus enthält Befehlskodebyte vom Speicher  
 - Steuerbus:  $\overline{M1}$ ,  $\overline{MREQ}$  und  $\overline{RD}$  aktiv

**Wirkung:** Dieser M1-Zyklus (weil Steuersignal  $\overline{M1}$  aktiv) dient zum Transport eines Befehlskodebytes aus dem Speicher zur CPU. Im Zusammenhang mit dem Anschluß sogenannter dynamischer Speicher (Kapitel 7) besitzt diese Zyklusart zentrale Bedeutung.

Speicherlesezyklus

- Kennzeichen:** - Adreßbus enthält aktuelle Speicheradresse  
 - Datenbus enthält Inhalt des ausgewählten Speicherplatzes  
 - Steuerbus:  $\overline{MREQ}$  und  $\overline{RD}$  aktiv

**Wirkung:** Transport eines Datenbytes (kein Befehlskode!) vom Speicher zur CPU

Speicherschreibzyklus

- Kennzeichen:** - Adreßbus enthält aktuelle Speicheradresse  
 - Datenbus enthält Daten für ausgewählten Speicherplatz  
 - Steuerbus:  $\overline{MREQ}$  und  $\overline{WR}$  aktiv

**Wirkung:** Transport eines Datenbytes von der CPU zum Speicher

E/A-Lesezyklus

- Kennzeichen:** - Adreßbus enthält aktuelle E/A-Geräteadresse  
 - Datenbus enthält Daten vom E/A-Gerät  
 - Steuerbus:  $\overline{IORQ}$  und  $\overline{RD}$  aktiv

**Wirkung:** Transport eines Datenbytes vom E/A-Gerät zur CPU  
E/A-Schreibzyklus

- Kennzeichen:** - Adreßbus enthält aktuelle E/A-Geräteadresse  
 - Datenbus enthält Daten für E/A-Gerät  
 - Steuerbus:  $\overline{IORQ}$  und  $\overline{WR}$  aktiv

**Wirkung:** Transport eines Datenbytes von der CPU zum E/A-Gerät

Die Speicher bzw. E/A-Einheiten werten sämtliche relevanten (für sie zutreffenden) Bussignale aus und reagieren entsprechend. Mit Hilfe des Busanalysators des POLY-COMPUTERS können wir den Ablauf auf den Bussen verfolgen.

2.2.2. Die Arbeit mit dem Busanalysator des POLY-COMPUTERS für das detaillierte Kennenlernen der Arbeitsweise der CPU sowie für Testzwecke ist neben dem bereits vorgestellten Einzelbefehlsbetrieb die schrittweise Ausführung einzelner Maschinenzklen sehr nützlich. Die CPU U880 besitzt für eine solche Betriebsart die gerätetechnischen Voraussetzungen. Bei Anlegen des Steuerbus-signalen WAIT ("warten") stoppt der Prozessor die Abarbeitung des aktuellen Zyklus zu einem Zeitpunkt, da alle Bussignale (Adreß-, Daten- und Steuerbus) den für diesen Zyklus gültigen Wert annehmen haben. Der POLY-COMPUTER besitzt die Hardwarevoraussetzungen zur Realisierung dieses Einzelzyklusbetriebes. Dazu zählen die 34 Leuchtdioden im Anzeigefeld des POLY-COMPUTERS sowie die Tasten **MCYCL** und **CYCL** der Bedientastatur (Bild 2.2.).

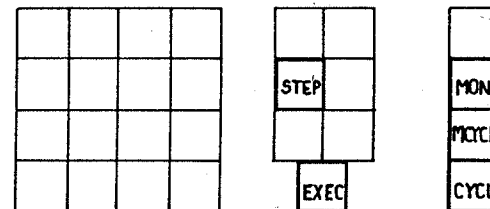
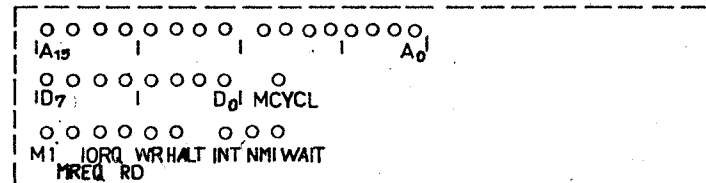


Bild 2.2. Elemente des Busanalysators

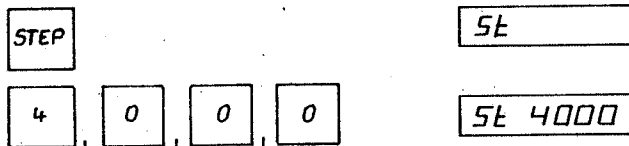
Die Leuchtdiodenanzeige ist entsprechend der Busstruktur der CPU in drei Reihen gegliedert, wobei in der oberen Reihe der Zustand der 16 Adreßbusleitungen, in der mittleren der Zustand der Datenbusleitungen und schließlich in der unteren der Zustand der wesentlichsten Steuerbusleitungen ständig angezeigt wird. Die rechte Leuchtdiode der Mittelreihe (MCYCL) leuchtet immer dann, wenn der Einzelzyklusbetrieb vorgewählt wurde. Wird nicht in dieser Betriebsart gearbeitet, so strahlen die Leuchtdioden je nach den auftretenden Signalen mehr oder weniger hell.

Anhand des Beispielprogrammes "ADDITION" (siehe Abschnitt 2.1.) wollen wir die Handhabung des Zyklusbetriebes kennenlernen.

Zunächst werden die vier Befehle des Beispielprogrammes in den RAM über die Hexastatur eingegeben (HEM).

In den Einzelzyklusbetrieb gelangen wir, indem wir zunächst die gleichen Bedienvorgänge wie beim Einzelbefehlsbetrieb vornehmen.

D.h.



Im Unterschied zum Einzelbefehlsbetrieb wird vor dem Kommando "Ausführung" ( EXEC ) die Vorwahltaste

MCYCL betätigt. Daraufhin leuchtet zur Betätigung die CYCL-Leuchtdiode auf. Über die Taste LICH kann diese Vorwahl jederzeit rückgängig gemacht werden.

Nach EXEC befinden wir uns schließlich im Einzelzyklusbetrieb. Das 7-Segmentdisplay verlischt und die Leuchtdioden des Busanalysators zeigen eindeutige Pegelverhältnisse an.

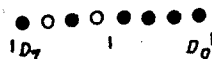
Zur Erleichterung des Ablesens der Adreß- und Datenbusinhalte wird durch die Beschriftung eine Gliederung in Vierergruppen (Tetraden) vorgenommen (siehe Abschnitt 1.2.3.), so daß unmittelbar die Hexawerte ablesbar sind.

Der Adreßbus enthält während des ersten Zyklus



● leuchtend  
○ nicht leuchtend

der Datenbus enthält



= AFH.

Von den Steuersignalen sind  $\overline{M1}$ ,  $\overline{MREQ}$ ,  $\overline{RD}$  und  $\overline{WAIT}$  aktiv: Somit handelt es sich bei diesem Zyklus um einen Befehlscode-Ladezyklus ( $\overline{M1}$  ist aktiv!), der vom Speicher ( $\overline{MREQ}$  ist aktiv!) ein Befehlscodebyte ( $\overline{M1}$  ist aktiv) liest ( $\overline{RD}$  ist aktiv). Die Steuerleitung  $\overline{WAIT}$  wird wie bereits erläutert zur Realisierung des Einzelzyklusbetriebes verwendet.

Zur Ausführung des jeweils nächsten Zyklus ist die Taste CYCL zu betätigen. Zur Kontrolle sind im folgenden die Buszustände während der einzelnen Zyklen des Programmes "ADDITION" angegeben:

Zyklus/ Zyklustyp	Adreßbus	Datenbus	Steuerbus (aktive Signale)
1/Befehlscode-Ladezyklus	4000H	AFH	$\overline{M1}$ , $\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
2/Befehlscode-Ladezyklus	4001H	06H	$\overline{M1}$ , $\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
3/Speicherlesezyklus	4002H	05H	$\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
4/Befehlscode-Ladezyklus	4003H	06H	$\overline{M1}$ , $\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
5/Speicherlesezyklus	4004H	02H	$\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
6/Befehlscode-Ladezyklus	4005H	32H	$\overline{M1}$ , $\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
7/Speicherlesezyklus	4006H	00H	$\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
8/Speicherlesezyklus	4007H	41H	$\overline{MREQ}$ , $\overline{RD}$ , $\overline{WAIT}$
9/Speicherschreibzyklus	4100H	07H	$\overline{MREQ}$ , $\overline{WR}$ , $\overline{WAIT}$

Im Programm "ADDITION" sind Ein-, Zwei- und Dreibytebefehle mit jeweils ein, zwei bzw. vier Zyklen enthalten. Der Einzelzyklusbetrieb ist vor allem dann vorteilhaft anwendbar, wenn einzelne Befehlszyklen studiert und deren Wirkung auf Speicher und peripherie Elemente untersucht werden sollen. Zwischen Einzelbefehls- und Einzelzyklusbetrieb kann jederzeit gewechselt werden. Der Übergang vom Einzelbefehlsbetrieb zum Einzelzyklusbetrieb erfolgt nach EXEC, wenn vorher die Vorwahltaste MCYCL gedrückt wurde.

Vom Einzelzyklusbetrieb gelangen wir umgekehrt zurück in den Einzelbefehlsbetrieb durch Betätigen der Taste

MON

MON

5E XXXXY

Jetzt können wieder sämtliche Monitorkommandos aufgerufen werden.

### 2.2.3. Start von Programmen und Prüfpunkte im POLY-COMPUTER

Bei Mikrorechner-testsystemen können hauptsächlich zwei verschiedene Typen der Prüfpunktrealisierung unterschieden werden, nämlich Hard- und Softwareprüfpunkte. Prüfpunkte mittels Hardware werden durch spezielle Vergleichsschaltungen realisiert, die ein in der Prüfpunktlogik eingespeichertes Bitmuster mit dem Busgeschehen ständig vergleichen und bei Übereinstimmung den Lauf des Anwenderprogrammes unterbrechen. Derartige Prüfpunkte sind auch für im ROM gespeicherte Programme einsetzbar und erlauben die Überwachung beliebiger Signale (Adressen, Daten, Steuersignale). Softwareprüfpunkte sind bei einem Echtzeittest (d.h. Programm läuft mit Rechnergeschwindigkeit ohne Intervention des Monitors) nur auf Adressen anwendbar, d.h. Unterbrechung des Anwenderprogrammes bei bestimmter Adresse. Realisiert werden sie dadurch, daß der Inhalt der Prüfpunkt-Speicheradresse mit einem Spezialbefehl geladen wird und nach Realisierung des Prüfpunktes wieder mit dem ursprünglichen Inhalt ausgetauscht wird.

Neben der befehlsweisen bzw. zyklusweisen Abarbeitung von Anwenderprogrammen können diese natürlich auch ohne Eingriffe des Monitors mit voller Rechnergeschwindigkeit ausgeführt werden. Der Start erfolgt mit Hilfe des Kommandos "Programmstart" mit Taste **GO**. Anschließend kann die Startadresse eingegeben werden. Ohne diese Angabe erfolgt der Programmstart ab der Adresse, die im PC enthalten ist.

Ablauf:

GO

GO

Adreßeingabe  
(kann entfallen)

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

GO XXXX

EXEC

Nach dem ersten **EXEC** erscheint auf dem Display **bI** für breakpoint input - Prüfpunkteingabe. Auf der Hextastatur kann jetzt eine Adresse eingegeben werden, die als Prüfpunkt fungiert.

Unter einem Prüfpunkt ist ein Bitmuster auf den Bussen der CPU zu verstehen, bei dessen Auftreten das Anwenderprogramm unterbrechen und in den Monitor übergegangen wird. Anschließend können unter Verwendung der Monitorkommandos die bis zu diesem Busereignis abgelaufenen Aktivitäten anhand der Register- und Speicherinhalte kontrolliert werden.

Im POLY-COMPUTER kann als Prüfpunkt ein Bitmuster des Adreßbusses verwendet werden.

Fortsetzung des Bedienablaufes:

Prüfpunkteingabe über Hextastatur

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

bI YYYY

Sind die mit yyyy gekennzeichneten Displaystellen nach dem vorangegangenen EXEC leer, so wurde bisher noch kein Prüfpunkt eingegeben. Erscheint dagegen eine Adresse, existiert bereits eine Prüfadresse, die über die Hextastatur änderbar bzw. durch Eingabe von 0000 gelöscht werden kann

EXEC

(Anzeige verlischt)

Tritt die Prüfadresse während des folgenden Programmlaufes auf, erfolgt eine Rückkehr in den Monitor mit der Ausschrift

`br XXXX YY`

mit `br` - für break (Monitorunterbrechung)

`xxxx` - Adresse des nächsten Befehls

`yy` - Datenmuster des nächsten Befehles.

Anschließend sind sämtliche Monitorkommandos verfügbar.

Dieser Adreßprüfpunkt ist nur für Adressen im RAM-Bereich verwendbar und muß stets auf das erste Befehlsbyte des jeweiligen Befehls gesetzt werden.

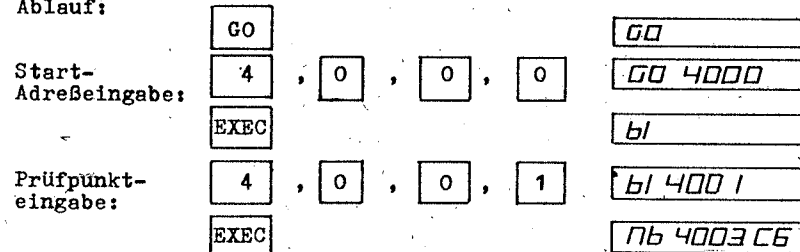
Beispiel:

Wir verwenden das Programm ADDITION aus Abschnitt 2.1.

Nach der Programmeingabe starten wir das Programm ab Adresse

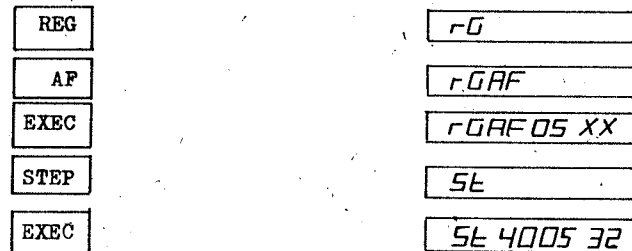
4000H und setzen einen Prüfpunkt auf Adresse 4003H.

Ablauf:



Nach dem Programmstart (nach dem zweiten `EXEC`) werden die Befehle XOR und ADD A, 05H ausgeführt. Dann tritt das gesuchte Adreßbitmuster 4003 auf und die Steuerung wird an den Monitor übergeben. Jetzt kann beispielsweise der A-Registerinhalt dargestellt und das Programm im Einzelbefehlsbetrieb fortgesetzt werden.

Ablauf:



usw.

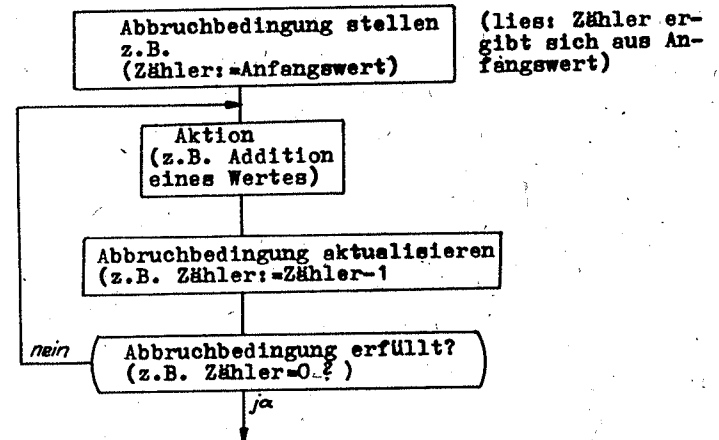
2.3. Die Programmverzweigung

Zum Abschluß jedes Befehlszyklus wird der Befehlszähler (PC) mit der Adresse des nächsten abzuarbeitenden Befehles gefüllt. Für die bisher behandelten Befehle bedeutet das z.B.:

Einbytebefehl: XOR A PC:=PC+1 ; PC wird um 1 erhöht  
Zweibytebefehl: ADD A,5 PC:=PC+2 ; PC wird um 2 erhöht  
Dreibytebefehl: LD (4000H),A PC:=PC+3 ; PC wird um 3 erhöht

Mit Befehlen dieser Art lassen sich lediglich Programme formulieren, die einmalig mit aufsteigenden PC-Werten durchlaufen werden, sogenannte "Geradeaus"-Programme. Sollen beispielsweise bestimmte Aktionen mehrfach in derselben Weise abgearbeitet werden, (z.B. 20-fache Addition eines Wertes zur Realisierung einer Multiplikation mit der Zahl 20, d.h. Wert + Wert +...+ Wert, = 20 x Wert) 20 mal

so ist es sehr speicherplatzaufwendig, diese Aktionen entsprechend oft im Programm aufzuschreiben. Günstiger ist die Programmierung einer Programmschleife, die allgemein folgendes Aussehen hat:



Für Konstruktionen dieser Art benötigen wir Befehle, die den Befehlszähler gezielt auf einen bestimmten Wert einstellen und damit Programmverzweigungen realisieren. Wir unterscheiden dabei zwischen Verzweigungen, die unabhängig von bestimmten CPU-Zuständen stets ausgeführt werden (unbedingte Verzweigung) und solchen, die nur bei Eintreten einer bestimmten Bedingung realisiert, sonst aber ignoriert werden (bedingte Verzweigung).

### 2.3.1. Die unbedingte Verzweigung

Programmverzweigungen werden beim Prozessor U880 durch die Gruppe der Sprungbefehle realisiert. Das sind Zwei- und Dreibytebefehle, die im zweiten bzw. zweiten und dritten Befehlsbyte den neuen PC-Wert enthalten.

Als erster Befehl dieser Gruppe soll der unbedingte absolute Sprungbefehl betrachtet werden.

Mnemonic:	JMP	nn	(Jump-springen) (nn = 16Bit-Adresse)
Befehlskode: binär	11000011		
	← n →		
	← n →		
hex	C3	n	n
Wirkung:	Füllen des Befehlszählers mit dem Wert nn und Verzweigen auf Adresse nn		

Mit einem solchen JMP-Befehl wird dafür gesorgt, daß die Programmausführung auf einem anderen als auf dem nachfolgenden Speicherplatz fortgesetzt werden kann.

Z.B.:

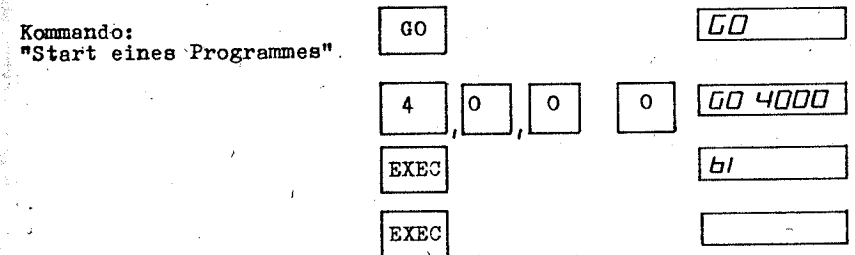
Speicher	
Adr.	Inhalt
4010	ADD A,5H
4012	JMP 4100H
40FB	
40FC	
40FE	
40FF	
4100	XOR A
4101	LD (43F0H),A

### Beispiel: Additionsschleife

Adresse	Befehlskode	Mnemonic	Kommentar
4000	AF	SCHLEIFE: XOR A	Lösche A-Register A:=0
4001	C6, 04	M1 ADD ,4H	Addiere 4 A:=A+4
4003	C3, 01,40	JMP M1	Verzweige zur Adresse 4001 endlose Schleife

Der Sprungbefehl wird im Beispiel so eingesetzt, das die Befehle ADD A,4H und JMP M1 alternierend (abwechselnd) endlos ausgeführt werden. Aus einer solchen Programmschleife gelangt der Prozessor nur durch externe Signale heraus (z.B. mit RESET).

Dieses Programm kann auf dem POLY-COMPUTER nach der Programmeingabe (MEM) gestartet werden durch:



Jetzt befindet sich der Prozessor in der Programmschleife, so daß keine Ausgaben zum Rechnerdisplay vorgenommen werden können. Am Zustand des Busanalysators kann in diesem Beispiel abgeschätzt werden, in welchem Adressbereich der Prozessor arbeitet. Es müssen die Dioden für die Adreßleitungen A<sub>0</sub>, A<sub>1</sub>, A<sub>2</sub> und A<sub>14</sub> leuchten (Adressen 4001 bis 4005).

### 2.3.2. Einige Flags der CPU U880

Arithmetische und logische Befehle (wir kennen bisher XOR A und ADD A,n) ergeben unter bestimmten Bedingungen nach ihrer Ausführung Ausnahmestände, die in Bitspeichern der CPU registriert und durch weitere Befehle auswertbar sind.

Beispielsweise kann die Addition zweier Zahlen dazu führen, daß die 8Bits des Akkumulators nicht mehr zur Aufnahme des Ergebnisses ausreichen.

	binär	hex	dez
<b>Beispiel:</b>	11001001	C9H	201
	+ 10000000	+ 80H	+ 128
Übertrag	<u>101001001</u>	149H	<u>329</u>
	8Bits		
	= 49H = 73		

Nach der Addition 201 + 128 steht im A-Register das falsche Ergebnis 73 ! Dieser Zustand wird in der CPU in einem Bedingungs-Flip-Flop (Flag-engl. = Flagge, Kennzeichen) gespeichert, das durch bestimmte Befehle auswertbar ist.

Ein Flag ist ein Einbitspeicher der CPU, der entsprechend dem Ergebnis hauptsächlich arithmetischer und logischer Befehle beeinflusst wird. Nach Bedarf kann das Flag programmtechnisch ausgewertet werden.

Der bei arithmetischen u.a. Befehlen auftretende Übertrag (ein Bit) wird in das Übertrags- oder Carry-Flag (C) gespeichert (kein Übertrag C:=0, sonst C:=1).

Ein weiteres wichtiges Flag ist das Null- oder Zero-Flag (Z).

Das Null- oder Zero-Flag (Z) wird nach arithmetischen und logischen Befehlen immer dann gesetzt (Z:=1), wenn das Ergebnis der Operation den Wert Null annimmt und ansonsten gelöscht (Z:=0).

**Beispiel:**

	10000000
	+10000000
	<u>100000000</u>
	A-Register

C:=1 - es tritt Übertrag auf  
Z:=1 - das Ergebnis im A-Register ist Null

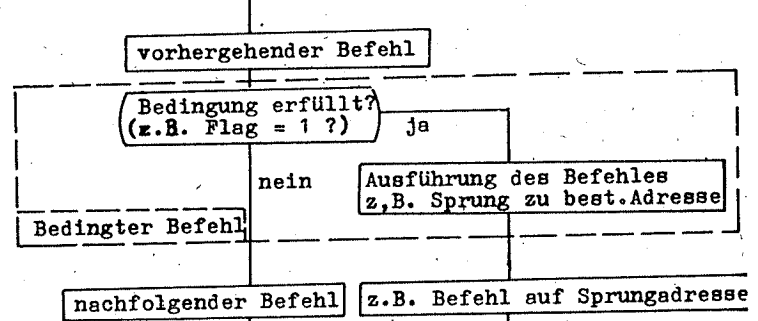
Die restlichen Flags werden im Abschnitt 4.3. erläutert.

### 2.3.3. Die bedingte Verzweigung

Bedingte Programmverzweigungen werden in Abhängigkeit vom Zustand der Flags ausgeführt.

Der Prozessor U880 besitzt dafür eine Vielzahl bedingter Sprungbefehle, die jeweils genau ein Flag auswerten. Je nach Zustand dieser Flags wird anschließend der Sprung ausgeführt (wenn Flag = 1 gesetzt) oder ignoriert (wenn Flag = 0 rückgesetzt) und der im Speicher nachfolgende Befehl ausgeführt.

Folgender Ablauf gilt allgemein für bedingte Befehle:



Im Befehlsmnemonik bedingter Befehle ist durch Symbole angegeben, welche Bedingung zur Ausführung des jeweiligen Befehls erfüllt sein muß. Es folgen vier Beispiele für bedingte Sprungbefehle.

Sprung bei Z=1: (Ergebnis gleich Null)

<b>Mnemonik:</b>	JZ nn	(Jump on zero - Sprung, wenn Ergebnis gleich Null bzw. Z-Flag gleich Eins zur Adresse nn)
<b>Befehlskode:</b>		
binär	11001010	
	← n →	
	← n →	
hex	CA	
	n	
	n	

**Wirkung:** Der PC wird folgendermaßen verändert

(a) wenn Z = 1 : PC := nn  
(b) wenn Z = 0 : PC := PC+1

d.h. wurde im Ergebnis einer vorangegangenen Operation das Z-Flag auf Eins gesetzt, wird der Sprung zur Adresse nn ausgeführt, sonst wird mit dem auf dem nächsten Speicherplatz stehenden Befehl fortgesetzt.

Sprung bei Z = 0: (Ergebnis ungleich Null)

Mnemonic	JNZ	nn	(Jump on not zero - Sprung, wenn Ergebnis ungleich Null bzw. Z-Flag gleich Null zur Adresse nn)
Befehlskode:			
binär	11000010		
	← n →		
	← n →		
hex	C2		
	n		
	n		
Wirkung: Der PC wird folgendermaßen verändert			
	(a) wenn Z=1:	PC: =PC+1	
	(b) wenn Z=0:	PC: =nn	

Sprung bei C=1: (Auftreten eines Übertrages)

Mnemonic:	JC	nn	(Jump on carry - Sprung bei Übertrag (C=1) zur Adresse nn)
Befehlskode:			
binär	11 011 010		
	← n →		
	← n →		
hex	DA		
	n		
	n		
Wirkung: Der PC wird folgendermaßen verändert			
	(a) wenn C=1:	PC: =nn	
	(b) wenn C=0:	PC: =PC+1	

Sprung bei C=0: (kein Übertrag aufgetreten)

Mnemonic:	JNC	nn	(Jump on no carry - Sprung, wenn kein Übertrag (C=0) zur Adresse nn)
Befehlskode:			
binär	11010010		
	← n →		
	← n →		
hex	D2		
	n		
Wirkung: Der PC wird folgendermaßen verändert			
	(a) wenn C=1:	PC: =PC+1	
	(b) wenn C=0:	PC: =nn	

Beispiel: Eine Programmschleife, die 250 mal durchlaufen wird, soll programmiert werden!

Lösung: Ein 8Bit-Register kann max.  $2^8=256$  Zustände annehmen, reicht also als Zählregister bis 250 aus.

Wir verwenden dafür das A-Register und den Test auf Ergebnis=0 (Z=1!).

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	3E	LD A,FAH	FAH=250-Zählgröße
4001	FA		
4002	3D	M1: DEC A	A:=A-1
4003	C2	JNZ M1	Sprung, wenn A ≠ 0 zu Adresse M1
4004	02		
4005	40		
4006	C3	M2: JMP M2	Sprung zu sich selbst (endlose Schleife)
4007	06		
4008	40		

Nachdem der zweite und dritte Befehl 250 mal ausgeführt sind, ist der A-Registerinhalt gleich Null und der Sprung zur Adresse M1 wird nicht ausgeführt sondern zu M2 übergegangen. Dort wurde eine 'Fangschleife' programmiert, aus der der Prozessor nur durch externe Signale herausgelangt (z.B. RESET).

Nach der Abarbeitung auf dem POLY-COMPUTER mit den

Kommandos  4000, ,  kann mit Hilfe der Taste  die Endlosschleife unterbrochen werden.

(für Monitorunterbrechung)

Damit gelangen wir wieder in das Monitorprogramm und können uns davon überzeugen, daß der Inhalt des A-Registers tatsächlich Null ist ( AF).

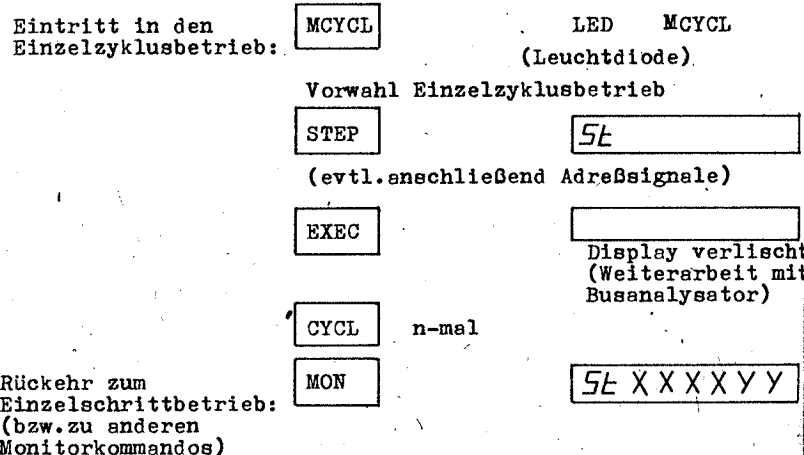
2.3.4. Zusammenfassung

In diesem Kapitel werden folgende neue Monitorcommandos eingesetzt:

(a) Kommando "Einzelzyklusbetrieb"

Unter Verwendung des Busanalysators können die Wirkungen einzelner Befehlszyklen studiert werden.

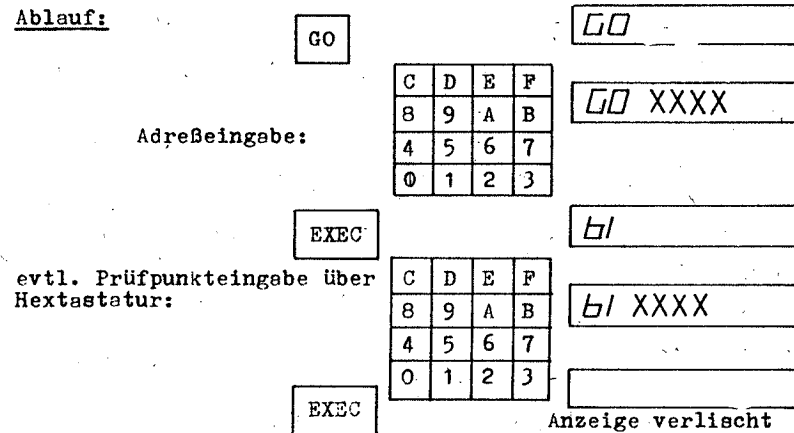
Ablauf:



(b) Kommando "Programmstart"

Das Kommando erlaubt den Start eines Anwenderprogrammes ab einer bestimmten Speicheradresse.

Ablauf:



Anschließend wird das Programm mit Rechnergeschwindigkeit abgearbeitet.

Der Prüfpunkt ist durch Eingabe von 0000 löschar. Der Monitor unterstützt genau einen Prüfpunkt. Neben diesem einen Monitorprüfpunkt können noch beliebige viele Anwenderprüfpunkte eingegeben werden (siehe Bedienhandbuch)

Nach dem Studium des zweiten Kapitels sollten folgende Fragen vom Leser beantwortet werden können:

1. Wieviele Bytes kann ein U880-Befehl umfassen?
2. Welche Informationen enthalten die Befehlsbytes?
3. Was wird unter einem Maschinenzklus verstanden? Worin unterscheiden sich Befehls- und Maschinenzklus?
4. Was ist ein Steuerbus?
5. Welche Vorgänge werden durch die Steuersignale  $\overline{M1}$ ,  $\overline{MREQ}$ ,  $\overline{IORQ}$ ,  $\overline{RD}$  und  $\overline{WR}$  jeweils gekennzeichnet?
6. Welche Zyklustypen wurden vorgestellt und welche Steuersignale sind jeweils aktiv?
7. Wieviele und welche Zyklen sind zur vollständigen Ausführung des Befehles LD A, (4020H) erforderlich?
8. Was verstehen wir unter bedingter und unbedingter Programmverzweigung?
9. Welche Aufgaben erfüllen Flags?
10. Welchen Zustand nehmen die Flags C und Z nach Ausführung der Addition 200 + 100 an?



### 3. Grundsätze für die Gestaltung von Programmen

Für die Lösung einer Aufgabe mit Hilfe eines Programmes existiert eine Vielzahl von Möglichkeiten. Der Programmierer wird bemüht sein, für seine konkrete Situation (Speicherumfang, geforderte Verarbeitungszeit, Programmierhilfsmittel) ein möglichst optimales Ergebnis anzubieten. Während für den Programmierer A z.B. die Minimierung des Programmes (kleinstmöglicher Speicherbedarf) im Vordergrund steht, sieht Programmierer B nach einer zeitoptimalen Variante, während schließlich für Programmierer C diese beiden Kriterien nur beiläufig interessieren und ihm an einer guten Lesbarkeit und Übersichtlichkeit besonders gelegen ist. Diese drei Gesichtspunkte sind in den seltensten Fällen vollständig vereinbar, so daß je nach Gewicht der Einflußfaktoren Kompromißlösungen gefunden werden müssen. Allerdings nehmen der Integrationsgrad der Schaltkreise und auch die Verarbeitungsgeschwindigkeiten von Mikroprozessoren und Speichern laufend zu und damit die Kosten je Bit ab, so daß zunehmend mehr Wert auf übersichtliche und gut strukturierte Programme auch zu Lasten von Umfang und damit Verarbeitungsgeschwindigkeit gelegt wird.

Kenngrößen für die Qualität von Software sind vor allem:

- Erfüllung der gestellten Zielstellung (logisch fehlerfrei)
- gute Wartbarkeit (übersichtlich, modular, gut dokumentiert, leicht änderbar und ergänzbar, d.h. eindeutige Software-schnittstellen)
- einfache, eindeutige Bedienbarkeit (falls notwendig)
- Portabilität (Möglichkeit der Abarbeitung auch auf anderen als dem Zielrechner).

Diese Zielstellungen sind mit den noch zu erläuternden Methoden der strukturierten Programmierung erreichbar.

Es ist beispielsweise offensichtlich, daß ein übersichtlich gestaltetes Programm weniger fehleranfällig vor allem bei Programmänderungen ist, als ein durch "Tricks" schwer durchschaubares. Nicht selten werden in solchen Fällen durch Programmkorrekturen mehr Fehler erzeugt als beseitigt. Damit erhöhen sich Entwicklungszeit und entsprechend Entwicklungskosten u.U. extrem. Die nun folgenden Ausführungen sollten deshalb aufmerksam studiert und bei künftigen Programmieraufgaben beherzigt werden.

### 3.1. Elemente zur Programmstrukturierung

Jede Programmierungsaufgabe läßt sich unter Verwendung von nur drei unterschiedlichen Programmstrukturen lösen und zwar

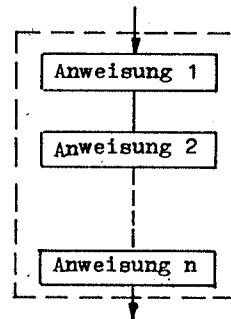
- der Folge,
- dem Zyklus
- und der Auswahl.

Die Verwendung ausschließlich dieser Strukturen gewährleistet eine weitgehend übersichtliche Programmgestaltung vor allem durch das Verbot von Sprunganweisungen zwischen den Strukturen sowie durch eindeutige Schnittstellen.

#### 3.1.1. Die Folge

Unter dem Strukturelement Folge wollen wir die Aneinanderreihung von Befehlen bzw. Anweisungen verstehen.

Eine Folge wollen wir grafisch folgendermaßen darstellen:

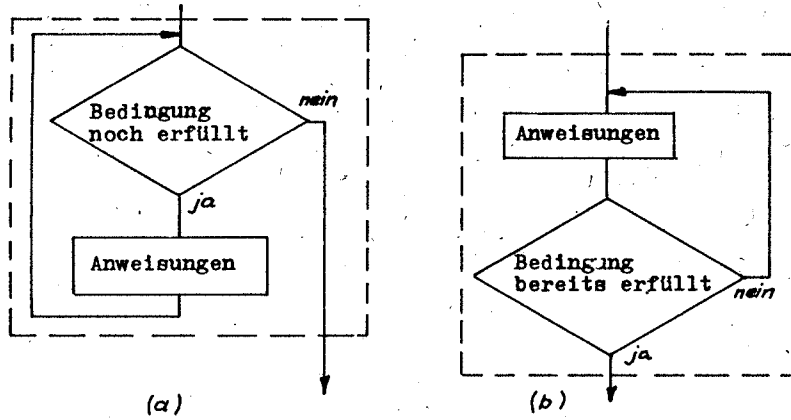


Unser Beispielprogramm ZAEHLER (Abschnitt 1.6.2.) ist eine solche Folge.

#### 3.1.2. Der Zyklus

Das Strukturelement Zyklus beinhaltet die wiederholte Ausführung einer bestimmten Menge von Befehlen solange eine bestimmte Bedingung erfüllt ist bzw. bis ein bestimmter Zustand eintritt.

Dementsprechend gibt es zwei Typen von Zyklen, die folgendermaßen veranschaulicht werden können:

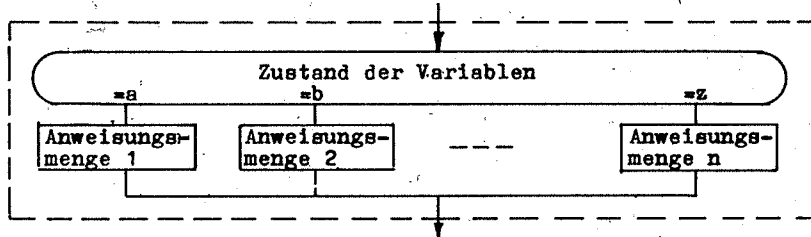


Die Programmschleife aus Abschnitt 2.3.3. entspricht der Version (b).

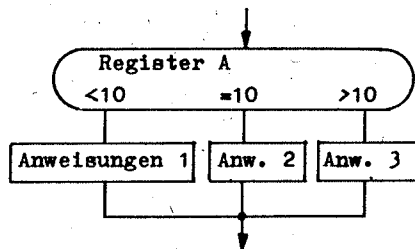
### 3.1.3. Die Auswahl

Das Strukturelement Auswahl gestattet die Ausführung einer Menge von Befehlen oder Anweisungen aus n möglichen Mengen in Abhängigkeit vom Zustand einer Variablen.

Symbolisch ist die Auswahl folgendermaßen darstellbar:



Beispielsweise sollen verschiedene Programme aufgerufen werden wenn das Ergebnis einer Operation größer, gleich oder kleiner 10 ist. Dann sähe die Auswahlstruktur so aus:



### 3.1.4. Der Programmablaufplan



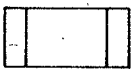

Der Programmablaufplan (PAP) ist die grafische Darstellung der Logik eines Programmes.

Programmablaufpläne sind ein Hilfsmittel bei der Programmierung, ein Zwischenschritt zur grafischen Formulierung des Programmablaufes auf einem abstrakteren und damit "überschaubareren" Niveau als das Programm mit seinen Prozessorbefehlen selbst. Der Programmablaufplan sollte folgende Anforderungen erfüllen:

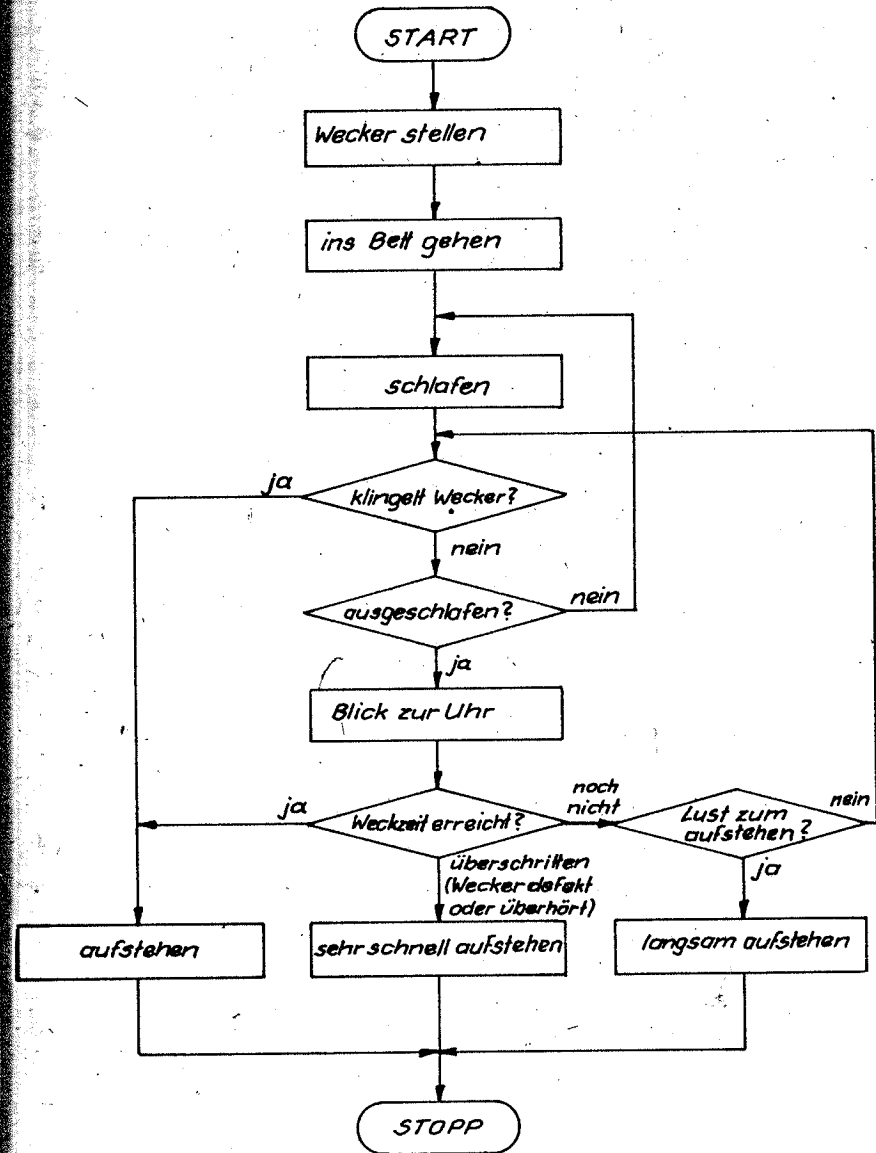
- (1) Darstellung der Reihenfolge der wesentlichen Programmaktivitäten,
- (2) Darstellung der Zusammenhänge zwischen verschiedenen Programnteilen (Programmhierarchie),
- (3) Darstellung soll unabhängig vom Typ des Rechners sein, auf dem das Programm später laufen soll,
- (4) Verwendete Unterprogramme müssen erkennbar und ausreichend beschrieben sein,
- (5) Namen, Marken, evtl. Adressen usw. sollten so vergeben werden, daß eine Bezugnahme zur Programmliste möglich ist,
- (6) Der PAP muß eindeutige Aussagen enthalten, um für einen "Schreibtischtest" brauchbar zu sein,
- (7) Je nach Problemstellung können grobe und feinere PAPs erforderlich sein.

Auf die Frage: Wie sollte man einen Programmablaufplan gestalten? werden wir in den nächsten beiden Abschnitten weiter eingehen. Jetzt soll uns zunächst nur der Formalismus, d.h. die grafischen Elemente und ihre Kombination interessieren.

Bild	Bezeichnung	Bedeutung
	Anweisung, Operation	Eine oder mehrere Operationen wie z.B. Addieren, Transport (je nach Abstraktionsniveau)
	Verzweigung	Variation des Programmablaufes in Abhängigkeit einer oder mehrerer Bedingungen (ein Eingang, mindestens zwei Ausgänge)
	Eingabe/Ausgabe	Sinnbild für beliebige E/A-Operationen

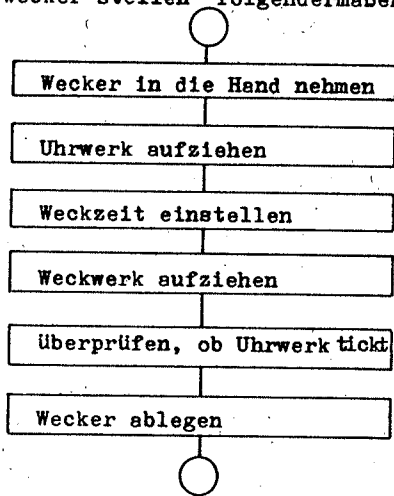
	Grenzstelle	Anfang, Ende oder Zwischenstopps von Programmen
	Verbindungsstelle	Sinnbilder dieser Art mit gleichem Inhalt (z.B. Nummer) stellen identische Programmstellen dar
	Unterprogramm	An dieser Stelle ist ein bereits bekanntes Programm einzufügen
	Richtungspfeil	Gibt Ablaufrichtung an (normalerweise von oben nach unten und von links nach rechts)

Beispiel: Wir wollen einen "Programmablaufplan" für den Vorgang "Schlafen" aufstellen:



Dieser Ablaufplan bedarf keiner eingehenden Erläuterung. In dieser Form lassen sich Abläufe beliebiger Art (wie demonstriert nicht nur von Rechnerprogrammen!) darstellen. Am Beispiel "Schlafen" ist deutlich die Problematik beim Aufstellen derartiger Ablaufpläne bereits zu erkennen. Der Verfasser muß ein bestimmtes Abstraktionsniveau wählen, sollte die Vorgänge nicht unzulässig vergrößern und darf andererseits keine für die umzusetzende Aufgabenstellung wesentlichen Einflüsse außer Acht lassen.

Es ist üblich, für komplexere Aktionen feiner gegliederte Teil-PAPs aufzustellen. Beispielsweise könnte ein solcher für die Aktion "Wecker stellen" folgendermaßen aussehen:



Nicht unerwähnt soll bleiben, daß der Programmablaufplan lediglich eines aus einer Reihe von Entwurfshilfsmitteln für die Programmierung darstellt. Daneben existieren Struktogramme, ausgewählte sprachliche Konstruktionen u.a. Sie besitzen gegenüber PAPs Vor- und Nachteile, die genannten Methoden sind aber im wesentlichen gleichberechtigt. Der Leser kann sich in der einschlägigen Literatur damit tiefgründiger beschäftigen. Das Arbeitsbuch verwendet ausschließlich PAPs.

### 3.2. Strukturierung von Programmen

Der Entwurf von Programmen vollzieht sich im wesentlichen in zwei Schritten:

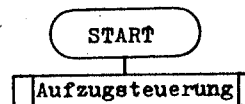
- 1) Strukturierung und Algorithmierung der Aufgabenstellung
- 2) Kodierung des Algorithmus in die verwendete Rechnersprache.

Während wir unter Strukturierung die Gliederung der Gesamtaufgabe in funktionelle Einheiten (Module) und deren Wechselbeziehungen verstehen wollen, bedeutet Algorithmierung das Umsetzen der geforderten Modulfunktionen durch geeignete Verfahren und rechen-technische Methoden. Von der Sorgfalt, mit der dieser erste Schritt des Programm-entwurfes ausgeführt wird, hängt der Erfolg des zweiten und der folgenden Testphase ab.

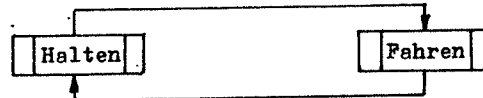
Für den Entwurf der weitaus meisten Programme empfiehlt sich ein Vorgehen nach der sogenannten "top-down"-Methode (zu dt: "von oben nach unten" bzw. im übertragenen Sinne: "Schrittweise Verfeinerung des Abstraktionsgrades").

Beispiel: Zur Demonstration dieser Methode verwenden wir ein häufig beanspruchtes Beispiel: Die Steuerung eines Aufzuges mittels Mikrorechner.

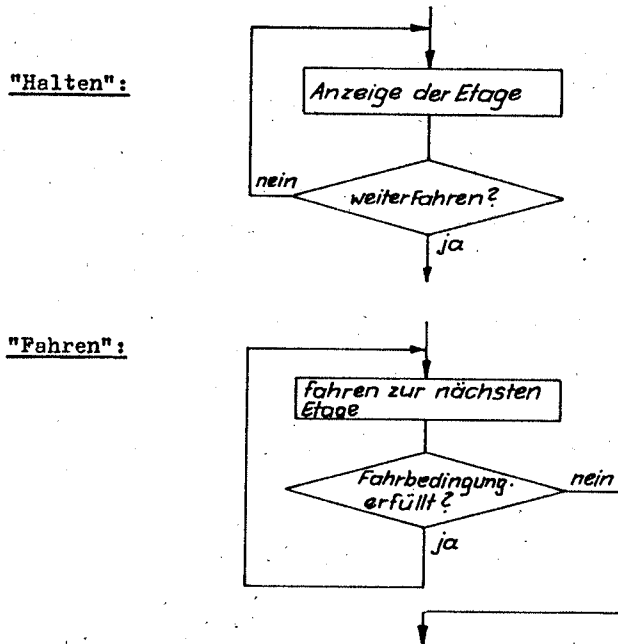
0. Abstraktionsebene:



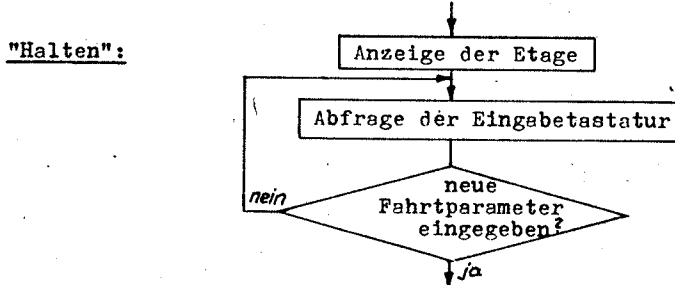
1. Abstraktionsebene: Ein Aufzug hat zwei Hauptbetriebszustände nämlich "Halten" und "Fahren", die ständig einander abwechseln.



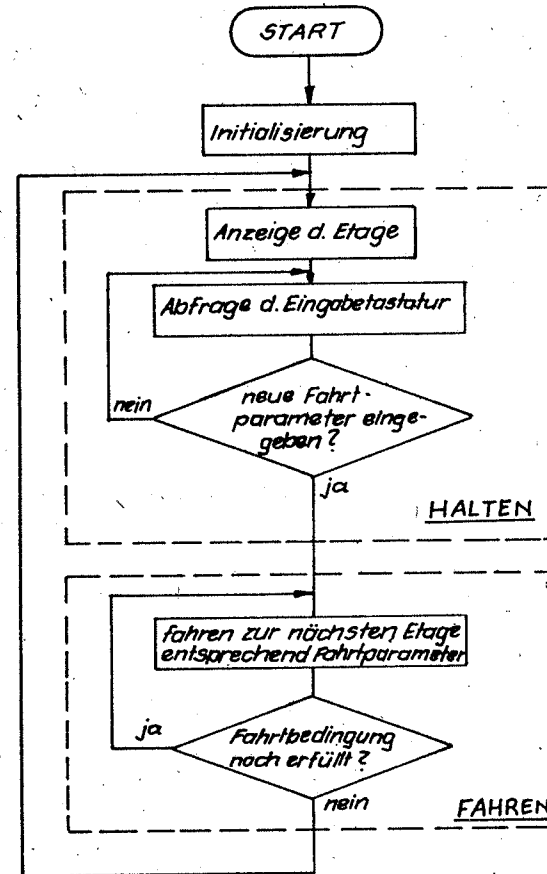
2. Abstraktionsniveau: Der Zustand "Halten" muß ebenso wie der Zustand "Fahren" untergliedert werden:



3. Abstraktionsniveau: Einige der Aktionen sind in dieser Form noch nicht als Programm kodierbar, so daß sich eine weitere Verfeinerung nötig macht



Wenn wir noch das Setzen von Anfangswertparametern (Initialisierung) ergänzen, nimmt unsere Aufzugsprogrammstruktur nun folgendes Gesamtbild an:



Für bestimmte Aktionen sind noch weitere Verfeinerungsniveaus denkbar, wir wollen aber an dieser Stelle abbrechen. "FAHREN", "HALTEN" und die Initialisierung bilden jeweils funktionelle Einheiten oder Module.

Ein Programm kann mehrere funktionelle Einheiten, die als Module bezeichnet werden, enthalten.

Kleinere funktionelle Strukturen mit mehreren Befehlen innerhalb eines Moduls (z.B. "Anzeige der Etage" im obigen Beispiel) werden als Prozeduren bezeichnet.

Zur Realisierung gut strukturierter Programme gilt bezüglich der Strukturen Modul und Prozedur folgendes:

- a) Ein Sprung ist nur an den Anfang einer Struktur erlaubt (oder innerhalb einer Struktur)
- b) Rückwärts darf nur zur Realisierung von Zyklen gesprungen werden, sonst nur vorwärts
- c) Innerhalb einer Struktur ist eine einheitliche Markengebung vorzunehmen:
  - Anfang erhält eine Marke z. B. NAME
  - Innerhalb der Struktur wird fortlaufend numeriert

z.B. NAME 1: usw.  
NAME 2:

d) Die Struktur sowie deren Anweisungen werden kommentiert.

Beispiel: Wir wählen den Modul "HALTEN" aus dem Aufzugsprogramm aus

```

HALT:      ; MODUL "HALTEN" DES AUFZUGSPROGRAMMS
HALT 1:    ; PROZEDUR "ANZEIGE"
            LD A,(PARAMETER 1)
            CALL ANZ           ;AUFRUF UP ANZEIGE

            HALT 2: ;PROZEDUR "ABFRAGE"
                    JP FAHREN ;ENDE HALTEN
    
```

Die Richtungspfeile im PAP "Aufzugssteuerung" lassen erkennen, daß die Forderungen a) und b) ebenfalls auf diesem Niveau erfüllt werden. Es wird grundsätzlich nur an den Anfang von Strukturen verzweigt, Rückwärtssprünge werden nur zur Realisierung von Abfragezyklen verwendet. Für die Übergabe der Fahrparameter müssen vor der Kodierung Vereinbarungen getroffen werden (siehe Kapitel 5).

### 3.3. Zusammenfassung

Strukturierte Programmierung ist ein zwingendes Erfordernis bei der effektiven Programmgestaltung. Mit zunehmender Komplexität der Aufgabenstellung gewinnt sie an Bedeutung. Auch für den Programmierer in Maschinsprache lassen sich Regeln und Wege zu strukturierten Programmen aufstellen, die vom Leser beachtet werden sollten, da Übersichtlichkeit und Änderbarkeit der Programme entscheidend zunehmen. Die hier dargelegten Grundsätze können nur bescheidene Ansätze einer sich stürmisch entwickelnde Softwaretechnologie vermitteln.

## 4. Die Register der CPU U880

### 4.1. Übersicht

Bisher lernten wir die CPU-Register A, PC sowie teilweise das Flagregister (Carry- und Zero-Flag) kennen. Darüber hinaus besitzt die CPU weitere Universal- und Spezialregister. Bild 4.1. zeigt die für den Anwender erreichbaren Register der CPU.

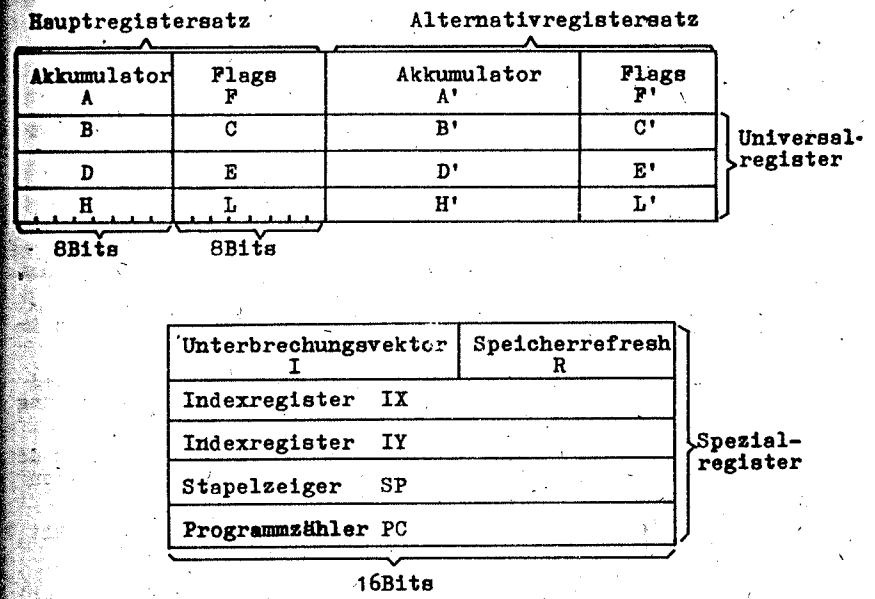


Bild 4.1. CPU 880-Registerkonfiguration

### 4.2. Der Hauptregistersatz

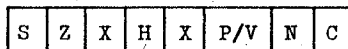
#### 4.2.1. Überblick

Der Hauptregistersatz besteht aus 8 8Bite-Registern: A, F, B, C, D, E, H, L. Während A und F (Akkumulator- und Flag-Register) spezielle Aufgaben zu erfüllen haben, dienen die Register B, C, D, E, H und L als universelle Zwischenspeicherplätze für Zählvariablen, Zwischenergebnisse, Adreßzeiger usw.

Register A (Akkumulator): Rechenregister; enthält bei arithmetischen und logischen Befehlen einen Operanden sowie nach Ausführung das Ergebnis.

**Register F (Flag-Register):** Enthält Bedingungs-Flip-Flops, die (siehe Abschnitt 2.3.2.) im Ergebnis arithmetischer und logischer Befehle gesetzt oder rückgesetzt werden und durch Programme auswertbar sind.

**Aufbau:** (Überblick, genaue Beschreibung der Flags erfolgt in den entsprechenden Abschnitten)

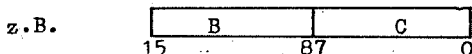


**Bedeutung:**

- S: Vorzeichen (Signum-)-Flag (gesetzt, wenn Ergebnis in Bit 7 = 1)
- Z: Null (Zero-)-Flag (gesetzt, wenn Ergebnis = 0; Abschnitt 2.3.2)
- X: ohne Bedeutung
- H: Halbübertrags-Flag (gesetzt, wenn Übertrag von Bit 3 nach Bit 4 erfolgt)
- P/V: Paritäts- oder Überlauf-Flag (Flag mit zwei Bedeutungen  
 bei logischen Operationen: Paritätsflag; gesetzt, wenn Anzahl der Einsen im Ergebnis gerade  
 bei arithmetischen Operationen: Überlauf-Flag; gesetzt, wenn Änderung von Bit 7
- N: Additions-/Subtraktionsflag (gesetzt, wenn vorhergehende Operation eine Subtraktion war)
- C: Übertrags (Carry)-Flag (gesetzt, wenn Übertrag von Bit auftritt - Abschnitt 2.3.2.)

**Register B, C, D, E, H, L (Universalregister):** Enthalten Operanden für verschiedene Aufgaben; Bezeichnung ist willkürlich.

Die Register B und C, D und E sowie H und L können für bestimmte Anwendungen jeweils als Registerpaare BC, DE und HL verwendet werden. Sie sind dann 16 Bits lang



und sind damit in der Lage, 16 Bit-Adressen o.ä. aufzunehmen. Spezielle Befehle der CPU U880 unterstützen diese 16 Bit-Anwendungen.

(Aus diesem Grunde zeigt das Monitorkommando REG auch stets beide Register eines Paares gleichzeitig an!)

**4.2.2. Der Ladebefehl**

Der Ladebefehl (Transportbefehl) besitzt eine zentrale Bedeutung im Befehlssatz der CPU U880. Tatsächlich handelt es sich um eine große Gruppe von Ladebefehlen, die aber alle eine Aufgabe erfüllen: Transport von Daten von einer Quelle zu einem Ziel.

Im Kapitel 1 lernten wir die Befehle

- LD A,n ; Laden eines Direktwertes nach A
- LD (nn),A ; Abspeichern A-Register auf Speicher
- LD A,(nn) ; Laden eines Speicherinhaltes nach A

kennen, die ebenfalls der großen Gruppe der Ladebefehle angehören.

Der Datentransport zwischen den Registern A,B,C,D,E,H und L erfolgt natürlich auch mit Hilfe von Ladebefehlen, die allgemein folgende Gestalt besitzen:

**Mnemonic:** LD r,r' (r-Zielregister  
r'-Quellenregister  
oder Direktwert)

**Befehlskode:** (hex)

Quelle Ziel	A	B	C	D	E	H	L	n
A	7F	78	79	7A	7B	7C	7D	3E
B	47	40	41	42	43	44	45	06
C	4F	48	49	4A	4B	4C	4D	0E
D	57	50	51	52	53	54	55	16
E	5F	58	59	5A	5B	5C	5D	1E
H	67	60	61	62	63	64	65	26
L	6F	68	69	6A	6B	6C	6D	2E

**Wirkung:** Transport eines Datenbytes vom Register r' (oder des Direktwertes n) zum Register r, keine Flagbeeinflussung

**Beispiel:** Zum Transport eines Datenbytes vom Register A zum Register B ist der Befehl

Kode	Mnemonic
47	LD B,A

zu verwenden.

### 4.2.3. Ein Multiplikationsprogramm

Wir besitzen jetzt die Voraussetzungen, ein einfaches Multiplikationsprogramm für natürliche Zahlen bis zu einer Größe des Ergebnisses von 255 (größte mit 8 Bits darstellbare Zahl) zu schreiben. Lediglich drei bereits bekannte Befehle müssen wir dazu noch verallgemeinern, nämlich die Addition von Registern, die Befehle Dekrementieren und Inkrementieren.

**Mnemonic:** ADD r (r-Universalregister oder Direktwert)

**Befehlskode:** (hex)

r	A	B	C	D	E	H	L	n
A	87	80	81	82	83	84	85	C6

**Wirkung:** Addition der Registerinhalte A und r, Ergebnis steht in A, r wird nicht verändert:  
Flags C, Z, V, S, N, H werden entsprechend dem Ergebnis beeinflusst

**Mnemonic:** DEC r (r-Registerinhalt)

**Befehlskode:** (hex)

r	A	B	C	D	E	H	L
DEC r	3D	05	0D	15	1D	25	2D

**Wirkung:** Registerinhalt wird um eins verringert  
( r:=r-1)  
Flags Z, V, S, N und H werden entsprechend dem Ergebnis beeinflusst

**Mnemonic:** INC r (r-Registerinhalt)

**Befehlskode:** (hex)

r	A	B	C	D	E	H	L
INC r	3C	04	0C	14	1C	24	2C

**Wirkung:** Registerinhalt wird um eins erhöht  
( r:=r+1)  
Flags Z, V, S, N und H werden entsprechend dem Ergebnis beeinflusst

Die Multiplikation zweier natürlicher Zahlen können wir ausführen, indem wir einen Operanden so oft zu sich selbst addieren, wie es der zweite Operand angibt.

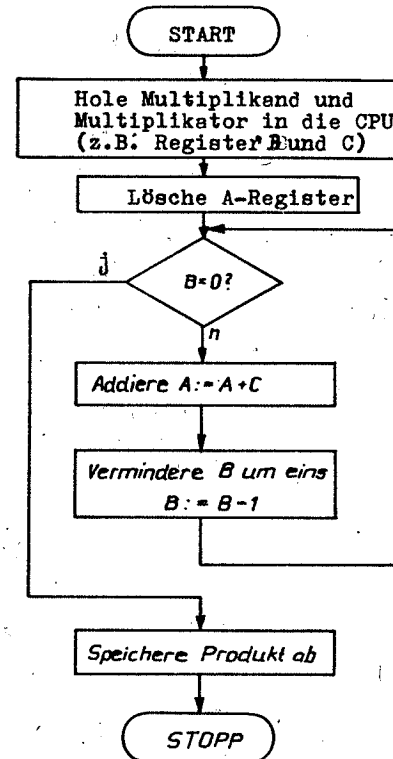
**Beispiel:**  $5 \times 7 = \underbrace{7+7+7+7+7}_{5x} = 35$

Mit Papier und Bleistift würde zwar kaum einer in dieser Art eine Multiplikationsaufgabe lösen, aber unser Rechner ist außerordentlich schnell und zyklische Wiederholungen einer Operation lassen sich sehr einfach programmieren.

Wir wollen folgende Vereinbarungen treffen:

Der Multiplikand soll sich auf dem Speicherplatz 4100H, der Multiplikator auf dem Speicherplatz 4101H und das Produkt schließlich auf Platz 4102H befinden.

Der PAP könnte folgendermaßen aussehen:





Programm: Binäre Multiplikation

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	3A, 00, 41	MULT: LD A, (4100)	;Hole Multiplikand
4003	47	LD B, A	;B:=Multiplikand
4004	3A, 01, 41	LD A, (4101)	;Hole Multiplikator
4007	4F	LD C, A	;C:=Multiplikator
4008	AF	XOR A	; lösche A-Register
4009	04	INC B	;Teste B auf 0
400A	05	DEC B	
400B	CA, 13, 40	MULT1:JZ ENDE	;Springe, wenn B=0
400E	81	ADD C	;A:=A+C
400F	05	DEC B	;B:=B-1
4010	C3, 0B, 40	JMP MULT1	;Schließe Zyklus
4013	32, 02, 41	ENDE: LD (4102), A	;Speichere Produkt a
4016	C3, 13, 40	JMP ENDE	;STOPP-Schleife (Sicherung gegen Fehlbedienung)

Zunächst laden wir das Programm ab Adresse 4000H in den POLY-COMPUTER ( MEM ) und geben anschließend Multiplikator und Multiplikand auf die Speicherzellen 4100H und 4101H ein ( MEM ).

(z.B. 4100H:=2, 4101H:=3)

Anschließend wird im Einzelbefehlsbetrieb ( STEP ) das Programm ausgeführt.

STEP , 4 , 0 , 0 , 0  
EXEC

5E40034

Jetzt müßte sich der Multiplikand im A-Register befinden:

REG AF EXEC

7GAF02XX

STEP EXEC

5E40043

Nach Ausführung des zweiten Befehles muß B den Multiplikanden enthalten:

REG BC EXEC

7G6C02XX

STEP EXEC STEP EXEC

5E4008AF

Jetzt enthält C den Multiplikator:

REG BC EXEC

7G6C0203

Weiter im Programm:

STEP EXEC

5E400904

EXEC

5E400A05

EXEC

5E400BCA

EXEC

(kein Sprung, denn B≠0)

5E400EB1

EXEC

5E400F05

A enthält jetzt die erste Zwischensumme 03:

REG AF EXEC

7GAF03XX

STEP EXEC

5E4010C3

B ist jetzt um eins verringert:

REG BC EXEC

7G6C0103

STEP EXEC (Sprung)

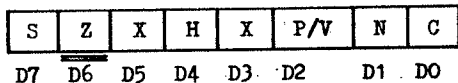
5E400BCA

Wir überprüfen das Z-Flag, ob es gesetzt ist (d.h. ob Ergebnis des DEC B-Befehls=0)

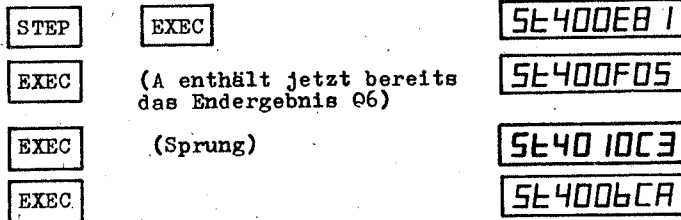
REG AF EXEC

7GAF03XX

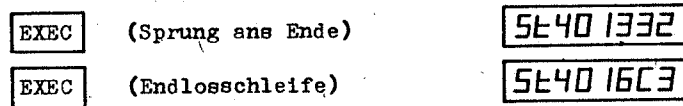
Der Gesamtzustand des Flagregisters ist für den Autoren nicht vorhersehbar. Wenn wir aber die beiden Hexzahlen xx als Binärzahl darstellen, dann muß entsprechend der Konfiguration des Flag-Registers



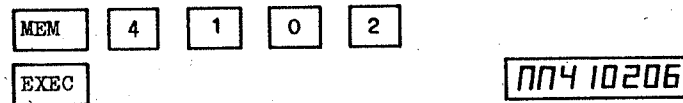
das Bit D6=0 sein, d.h. die letzte Flagbeeinflussende Aktion (DEC B) hatte ein von Null verschiedenes Ergebnis. Der bevorstehende Sprungbefehl wird also diesmal noch nicht ausgeführt (JZ ENDR).



Eine Überprüfung des Flag-Registers (siehe oben) ergäbe jetzt, daß das Bit D6 gesetzt (=1) ist, d.h. DEC B hat das Ergebnis Null geliefert.



Das Multiplikationsergebnis befindet sich jetzt auf dem Speicherplatz 4102H:



Das Programm kann mit anderen Faktoren in der gleichen Weise wiederholt werden oder aber mit dem Kommando



werden.

In diesem Fall ist anschließend über die Taste MON in den Monitor zurückzukehren und mittels



kann das Ergebnis überprüft werden.

### 4.3. Der Alternativregistersatz

Der Alternativregistersatz (oder auch zweiter Registersatz genannt) ist identisch zum Hauptregistersatz (oder ersten Registersatz) aufgebaut, d.h. er enthält ebenfalls 8 8Bit-Register, die mit A', F', B', C', D', E', H' und L' bezeichnet werden und den gleichen Verwendungszweck wie die entsprechenden Register des Hauptregistersatzes haben. Der zweite Registersatz wird vor allem angewendet für

- a) schnelle Umschaltung zwischen zwei Programmen, die wechselseitig abgearbeitet werden (vor allem bei Unterbrechungsbehandlungen, siehe Kapitel 8)
- b) Zwischenspeicherung von zusätzlichen Variablen, die im ersten Registersatz keinen Platz mehr finden (selten angewendet).

Die Register des Alternativregistersatzes sind über dieselben Befehle wie die Register des Hauptregistersatzes zugreifbar. Die Umschaltung erfolgt über die Spezialaustauschbefehle EXX und EXAF.

Mnemonic:	EXX	(exchange-Austausch)
Befehlskode:		
binär	11011001	
hex	D 9	
Wirkung:	Die Universalregister des Hauptregistersatzes B, C, D, E, H, L werden gegen die Universalregister des Alternativregistersatzes B', C', D', E', H', L' ausgetauscht.	
Mnemonic:	EXAF	(exchange-AF ↔ AF' Austausch Registerpaar AF)
Befehlskode:		
binär	00001000	
hex	08	
Wirkung:	Die Register A und F des Hauptregistersatzes werden mit den Registern A' und F' des Alternativregistersatzes ausgetauscht.	

Nach dem Rücksetzen **RESET** der CPU U880 wird prinzipiell mit den Registern des Hauptregistersatzes gearbeitet. Nach den Befehlen

EXX  
EXAF

sind nur noch die Register des Alternativregistersatzes vom Programm aus greifbar. Eine Rückumschaltung zum Hauptregistersatz erfolgt mit der gleichen Befehlsfolge

EXX  
EXAF.

Die Befehle EXX und EXAF sind natürlich auch einzeln anwendbar. Die Register des Alternativregistersatzes sind ebenfalls mit dem Kommando **REG** des Monitors änderbar. Im allgemeinen werden die Alternativregister wenig verwendet.

#### 4.4. Die Spezialregister

Die sechs Spezialregister der CPU U880 sind überwiegend zur Organisation des Programmablaufes eingesetzt. Die Beschreibung der Funktionen des Programmzählers (PC) erfolgte bereits im Abschnitt 1.4.4., die Register I und R werden in den Kapiteln bzw. 8 erläutert.

##### 4.4.1. Die Indexregister IX und IY

Die Register IX und IY sind 16Bit-Register, die sehr universell einsetzbar sind. Ihre Bezeichnung "Index"-Register rührt daher, daß unter Verwendung bestimmter Befehle mit ihnen eine indexierte Adressierung möglich ist.

Das Prinzip veranschaulicht Bild 4.2.

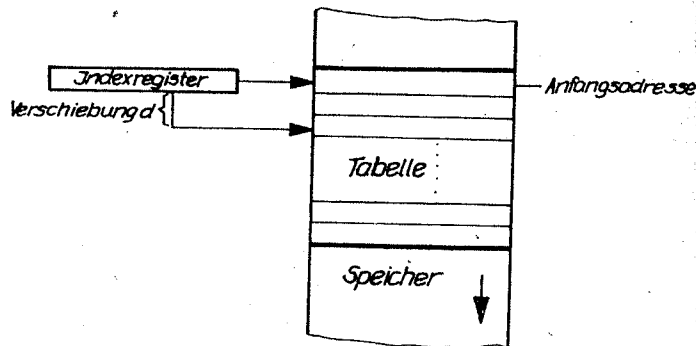


Bild 4.2. Indexierte Adressierung

Im **Indexregister** ist die Basisadresse eines interessierenden **Speicherfeldes** enthalten (erste Adresse). Durch Addition eines **Verschiebewertes** gelangt man dann auf einfache Art und Weise zu jedem Platz innerhalb dieses Bereiches, ohne die absolute Adresse kennen zu müssen. Diese Verschiebung muß im Rahmen des **Befehlskodes** mit angegeben werden und sich in den Grenzen von **-128 bis +127** als vorzeichenbehaftete Zahl in Zweierkomplement-Darstellung bewegen (siehe Abschnitt 9.2.). Darüber hinaus sind **IX** und **IY** in vielen Fällen wie Universalregisterpaare einsetzbar.

Drei der wichtigsten Befehlstypen für Indexregister sollen kurz vorgestellt werden.

<b>Mnemonic:</b>	LD IX,n	LD IY,nn (nn-16-Bit-Wert)
<b>Befehlskode:</b> (hex)	DD 21 n n	FD 21 n n
<b>Wirkung:</b>	Laden eines <i>Index</i> registers mit einem 16-Bit-Direktwert. Keine Flagbeeinflussung.	

<b>Mnemonic:</b>	LD r,(IX+d)	LD r,(IY+d)	(r-Universalregister oder A, d-Verschiebung)																
<b>Befehlskode:</b> (hex)	DD xx d	FD xx d																	
	<table border="1"> <tr> <td>r</td> <td>A</td> <td>B</td> <td>C</td> <td>D</td> <td>E</td> <td>H</td> <td>L</td> </tr> <tr> <td>xx</td> <td>7E</td> <td>46</td> <td>4E</td> <td>56</td> <td>5E</td> <td>66</td> <td>6E</td> </tr> </table>			r	A	B	C	D	E	H	L	xx	7E	46	4E	56	5E	66	6E
r	A	B	C	D	E	H	L												
xx	7E	46	4E	56	5E	66	6E												
	d = -128 bis +127 (Zweierkomplementdarstellung) (siehe Kapitel 9)																		

**Wirkung:** Der Inhalt des Speicherplatzes, dessen Adresse sich aus der Summe des Indexregisterwertes und der Verschiebung d ergibt, wird in das Register r geladen. Keine Flag-Beeinflussung.

Mnemonic: LD (IX+d),r      LD (IY+d),r (r-Universalregister  
oder A,  
d-Verschiebung)

Befehlskode:      DD      PD  
(hex)      xx      xx  
                  d      d

r	A	B	C	D	E	H	L
xx	77	70	71	72	73	74	75

d = -128 bis +127 (Zweierkomplementdarstellung.  
siehe Kapitel 9)

Wirkung: Der Inhalt von Register r wird auf den Speicherplatz transportiert, dessen Adresse sich aus der Summe des Indexregisterwertes und der Verschiebung d ergibt. Keine Flag-Beeinflussung.

Beispiel: Die Inhalte der Speicherplätze 4100H bis 4103H sollen entsprechend in die Register A, B, C, D transportiert werden:

Adresse	Befehlskode (hex)	Mnemonic	Kommentar
4000	DD, 21, 00, 41	LD IX, 4100H	IX:=4100H
4004	DD, 7E, 00	LD A, (IX+0)	A:=(4100H)
4007	DD, 46, 01	LD B, (IX+1)	B:=(4101H)
400A	DD, 4E, 02	LD C, (IX+2)	C:=(4102H)
400D	DD, 56, 03	LD D, (IX+3)	D:=(4103H)

Bemerkung: Sämtliche Befehle, die Indexregister verwenden, besitzen als erstes Befehlskodebyte DDH (bei IX) bzw. PDH (bei IY).

#### 4.4.2. Der Stapelzeiger (SP)

Der Stapelzeiger (SP = stack pointer) ist ein 16 Bit-Register und enthält die aktuelle Adresse eines im externen RAM-Speicher befindlichen Stapelspeichers (Stack)

Für die Programmorganisation ist es häufig sehr sinnvoll, wenn neben den relativ wenigen Registerplätzen in der CPU noch ein externer (außerhalb der CPU befindlicher) Speicherbereich existiert, um dessen Organisation und Adressierung sich der Programmierer nur wenig bemühen muß und auf dem all das abgelegt werden kann, was im Augenblick den dringend benötigten und knappen Registerplatz der CPU belastet und erst später wieder benötigt wird (z.B. Programadressen, alte Registerinhalte, zu übergehende Programmparameter). Für diese Zwecke wurden Stapel-speicherorganisationen geschaffen (Speicher dieser Art werden auch als Kellerspeicher oder Stack bezeichnet), die den Programmierer unterstützen.

Eine solche Form des Stapelspeichers, die auch bei der CPU US80 Anwendung fand, ist die des LIFO (engl. für: last in, first out). Damit soll etwas über die Reihenfolge des Zugriffs auf einen solchen Stapelspeicher ausgesagt werden, nämlich:

Die zuletzt eingeschriebenen Daten können als erste wieder entnommen werden.

Bild 4.3. veranschaulicht das Prinzip, das mit dem eines Vorratskellers vergleichbar ist.

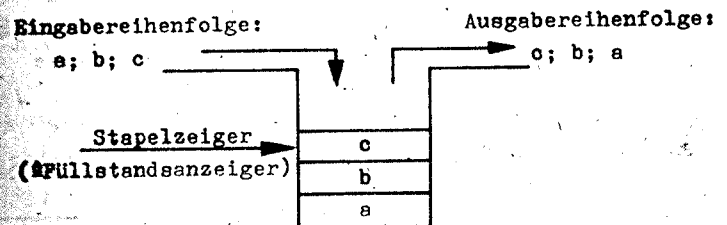


Bild 4.3. Prinzip des LIFO-Stapelspeichers

Bevor man an die zuerst eingekellerten Dinge heran kann, müssen die später eingelagerten Gegenstände (Daten) entnommen werden.

Bei Mikrorechnern mit der CPU U880 ist dieser Stapelspeicher irgendwo im RAM anzuordnen und zwar möglichst auf sehr hohen Adressen, da der Stapel ja "rückwärts" wächst, d.h. mit zunehmender Füllung Speicher mit niedrigeren Adressen belegt. Die Festlegung des Stapelbereiches geschieht durch Laden des Stapelzeigers mit der beabsichtigten höchsten Adresse des Stapelspeichers (Bild 4.4.)

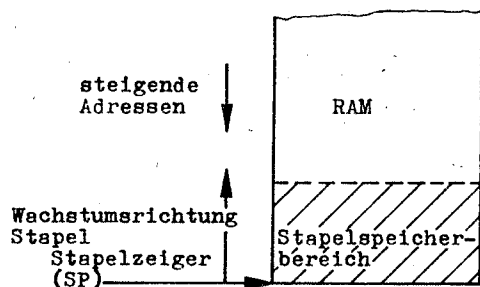


Bild 4.4. Anfangszuweisung Stapelzeiger

Wir können festhalten:

Der Stapelspeicher (Stack) ist ein Bereich im RAM (Schreib-Lese-Speicher) des Mikrorechners, dessen Verwaltung durch die CPU unterstützt wird. Das Register SP (Stapelzeiger) wird anfangs mit der höchsten Adresse des Stapelbereiches geladen und zeigt während der Nutzung stets auf die letzte belegte Adresse des Stapelspeichers.

Der Stapelspeicher hat keine feste Größe, sondern wächst (rückwärts) und schrumpft während der Benutzung (dynamischer Speicherbereich).

Der Stapelzeiger wird normalerweise am Anfang eines Programmes auf die höchste verfügbare RAM-Adresse gestellt.

**Bemerkung:** Bei Versuchen mit dem POLY-COMPUTER ist diese Maßnahme nicht unbedingt erforderlich, da das Monitorprogramm bereits den Stapelzeiger optimal lädt. Soll danach der Stapelzeiger verändert werden, so sollte dies nicht auf Adressen oberhalb 4340H erfolgen, da sonst eventuell Monitorarbeitszellen zerstört werden.

Die nun folgenden Befehle werden zur Verwendung des Stapelspeichers benötigt.

<b>Mnemonic:</b>	LD dd, mn	(dd - Registerpaar mn - 16 Bit-Wert z.B. Adresse) <sup>A</sup>								
<b>Befehlskode:</b>	00dd0001									
<b>binär</b>	←-n-→ ←-n-→	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>BC</td> <td>DE</td> <td>HL</td> <td>SP</td> </tr> <tr> <td>dd</td> <td>00</td> <td>01</td> <td>10</td> </tr> </table>	BC	DE	HL	SP	dd	00	01	10
BC	DE	HL	SP							
dd	00	01	10							
<b>hex</b>	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>BC</td> <td>DE</td> <td>HL</td> <td>SP</td> </tr> <tr> <td>01</td> <td>11</td> <td>21</td> <td>31</td> </tr> </table> <p style="text-align: center;">n n</p>	BC	DE	HL	SP	01	11	21	31	
BC	DE	HL	SP							
01	11	21	31							

**Wirkung:** Lade Registerpaar BC, DE, HL bzw. Spezialregister SP mit dem Wert mn.

Flags werden nicht beeinflusst

<b>Mnemonic:</b>	PUSH qq	(qq - Registerpaar)								
<b>Befehlskode:</b>	11qq0101									
<b>binär</b>		<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>BC</td> <td>DE</td> <td>HL</td> <td>AF</td> </tr> <tr> <td>qq</td> <td>00</td> <td>01</td> <td>10</td> </tr> </table>	BC	DE	HL	AF	qq	00	01	10
BC	DE	HL	AF							
qq	00	01	10							
<b>hex</b>	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>AF</td> <td>BC</td> <td>DE</td> <td>HL</td> </tr> <tr> <td>F5</td> <td>C5</td> <td>D5</td> <td>E5</td> </tr> </table>	AF	BC	DE	HL	F5	C5	D5	E5	
AF	BC	DE	HL							
F5	C5	D5	E5							

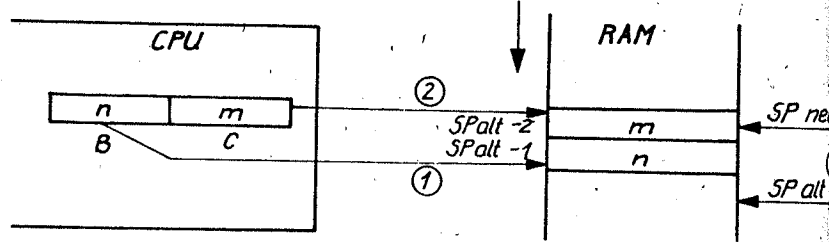
**Wirkung:** Der Inhalt des Registerpaares qq wird im Stapelspeicher abgelegt, das Registerpaar selbst bleibt unverändert.

niederwertiges Register → auf Adresse (SP-2)  
(C, E, L bzw. F)

höherwertiges Register → auf Adresse (SP-1)  
(B, D, H bzw. A)

Anschließend wird Stapelzeiger um 2 verringert  
SP: =SP-2

**Beispiel:** PUSH BC



Mnemonic: POP qq (qq - Registerpaar)

Befehlskode:

binär 11qq0001

	BC	DE	HL	AF
qq	00	01	10	11

hex

AF	BC	DE	HL
F1	C1	D1	E1

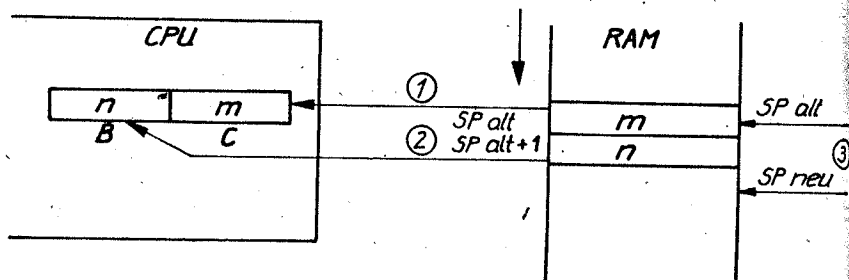
**Wirkung:** Das Registerpaar qq wird mit den zwei zuletzt im Stapelspeicher abgelegten Datenbytes geladen.

niederwertiges Register := Inhalt von Adresse SP (C, E, L, F)

höherwertiges Register := Inhalt von Adresse SP+1 (B, D, H, A,)

Anschließend wird der Stapelzeiger um 2 erhöht  
SP := SP+2

**Beispiel:** POP BC

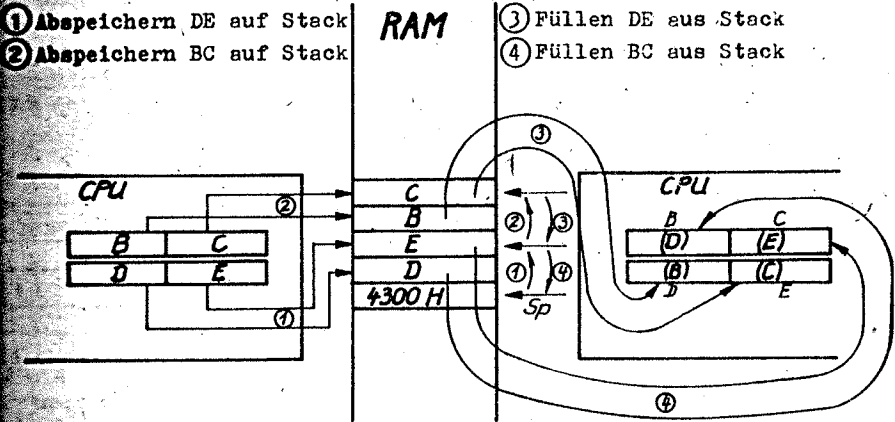


**Programmbeispiel:** Stapelspeicher

Die Inhalte der Registerpaare BC und DE sollen mit Hilfe des Stapelspeichers ausgetauscht werden.

Anstelle eines PAP wollen wir uns den Ablauf im Stapelspeicher verdeutlichen:

- ① Abspeichern DE auf Stack
- ② Abspeichern BC auf Stack
- ③ Füllen DE aus Stack
- ④ Füllen BC aus Stack



Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	31, 00, 43	STAPEL: LD SP, 4300H	; Stelle SP-Register auf RAM-Platz
4003	D5	PUSH DE	; DE auf Stapel
4004	C5	PUSH BC	; BC auf Stapel
4005	D1	POP DE	; Füllen DE
4006	C1	POP BC	; Füllen BC

Das Programm laden wir in den POLY-COMPUTER (MEM) und setzen die Registerpaare BC und DE auf bestimmte Werte (REG).

Das Programm können wir dann im Einzelbefehls- (STEP) und im Einzelzyklusbetrieb (CYCL) abarbeiten und die oben gezeigten Vorgänge beobachten.

Wir füllen beispielsweise das Registerpaar BC mit 1122H:

REG	BC	EXEC	rGbCXXXX
1	1	EXEC	rGbC11XX
2	2	EXEC	rGdEXXXX

und auf die gleiche Weise das Registerpaar DE mit 3344H.

Anschließend führen wir das Programm aus:

```

[STHP] [4] [0] [0] [0] [EXEC]
[EXEC]

```

Auf den Stack-Plätzen 42FE und 42FF befinden sich jetzt die Daten aus dem Registerpaar BC

```

[MEM] [4] [2] [F] [E] [EXEC]
[EXEC]
[STEP] [EXEC]
[EXEC]
[EXEC]

```

Der Registertausch ist vollzogen:

```

[REG] [BC] [EXEC]

```

BC enthält jetzt 3344H

```

[EXEC]
[EXEC]

```

und DE enthält 1122H.

Im Einzelzyklusbetrieb sind die einzelnen Schreib- und Lesezyklen noch besser zu verfolgen, der Leser sollte das selbständig vornehmen.

```

5E4003d5
5E4004C9

```

```

7742FE22
7742FF11
5E4005d1
5E4006C1
5E4007XX

```

```

rG6C3344
rG6C3344
rGdE1122

```

5. Zusammenfassung

Das volle Verständnis der Arbeitsweise des Stapelspeichers und des Stapelzeigers ist Voraussetzung für das nächste Kapitel und sollte deshalb nochmals überprüft werden. Nach dem Studium des vierten Kapitels sollte der Leser auf folgende Fragen antworten können:

- 1) Welche Registerkonfiguration weist die CPU U880 auf?
- 2) Wie erfolgt der Zugriff zum Alternativregistersatz?
- 3) Welche Flags enthält das Flag-Register und was zeigen sie an?
- 4) Welches Datenmuster ist im Flagregister nach der Addition der beiden Binärzahlen 11001100+11110000 enthalten?
- 5) Wie sieht ein einfaches Divisionsprogramm für Binärzahlen aus, wenn man eine Teilbarkeit des Dividenden durch den Divisor ohne Rest voraussetzt und die Division auf eine Addition zurückführt? Stellen Sie den PAP auf, kodieren Sie das Programm und testen Sie es!
- 6) Welche Spezialregister enthält die CPU U880?
- 7) Was ist indexierte Adressierung?
- 8) Was ist ein Stapelzeiger?
- 9) Erläutern Sie die Arbeitsweise eines LIFO-Stapelspeichers!
- 10) Realisieren Sie ein Programm, das mit Hilfe des Stapelspeichers die Daten der Registerpaare BC, DE und HL zyklisch vertauscht
  - BC:=HL
  - DE:=BC
  - HL:=DE !

5. Unterprogrammtechnik und Stapelspeicher

Häufig werden bestimmte Befehlsfolgen mehrfach und an verschiedenen Stellen in einem Programm benötigt. Diese gleiche Befehlsfolge entsprechend oft im Programm aufzuschreiben hätte mehrere Nachteile: Erstens wird Speicherplatz verschwendet, zweitens ist der Aufwand für den Programmierer beim Schreiben und Eingeben des Programmes groß und drittens steigt damit die Fehlerwahrscheinlichkeit. Aus diesen Gründen wird für solche Fälle die Unterprogrammtechnik verwendet. Sie erlaubt, mit nur geringem zusätzlichen Organisationsaufwand, eine mehrfach benötigte Befehlsfolge nur einmal im Speicher abzulegen und von den verschiedensten Programmpunkten aus aufzurufen. Beispielsweise könnte unsere Multiplikationsroutine aus dem Kapitel 4 von verschiedenen Stellen eines Programmes aus, aufgerufen werden müssen (siehe Bild 5.1.).

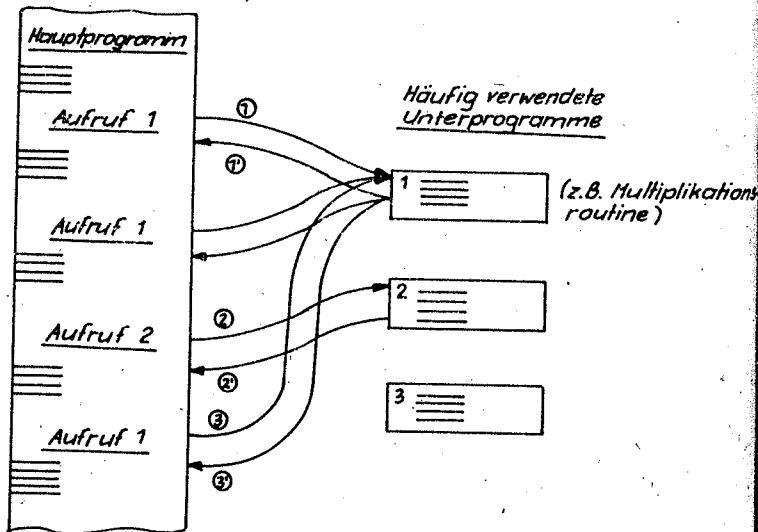


Bild 5.1. Zugriffsschema zu häufig benötigten Unterprogrammen

Zwei wesentliche Probleme sind in diesem Zusammenhang zu lösen:

- 1) Rückkehr aus der aufgerufenen Befehlsfolge an die aufrufende Stelle im Programm.
- 2) Universelle Übergabe von Parametern, Variablen, Ergebnissen usw.

In sich geschlossene Befehlsfolgen, die nach Ausführung zur aufrufenden Adresse verzweigen, werden als Unterprogramme (UP) bezeichnet.

Die sinnvolle Anwendung von Unterprogrammen führt gleichzeitig zu einer Verbesserung der Programmgliederung und damit zur Erhaltung der Übersichtlichkeit.

5.1. Aufruf von Unterprogrammen und Rückkehr zum aufrufenden Programm

Die CPU U880 unterstützt mit ihrem Befehlssatz und ihrer Hardwareorganisation die Handhabung von Unterprogrammen. Bereits beim Aufruf eines Unterprogrammes wird durch Abspeicherung des aktuellen Programmzählerinhaltes im Stapelspeicher die richtige Rückkehr zum aufrufenden Programm vorbereitet (siehe Bild 5.2.). Für den Aufruf von Unterprogrammen wird ein spezieller CALL-Befehl verwendet.

Mnemonic:	CALL	nn	(call - Ruf (nn - Adresse des Unterprogrammes))
Befehlskode: binär	1 1 0 0 1 1 0 1		
	← n →		
	← n →		
hex	CD		
	n		
	n		

**Wirkung;** Das Programm verzweigt (ähnlich wie bei einem Sprungbefehl) zur Adresse nn. Vorher wird jedoch der "alte" Programmzähler (PC) (er enthält die Adresse des dem CALL-Befehl folgenden Befehls im aufrufenden Programm) im Stapelspeicher abgelegt (siehe Wirkung des PUSH-Befehls!) und der Stapelzeiger um zwei erniedrigt.



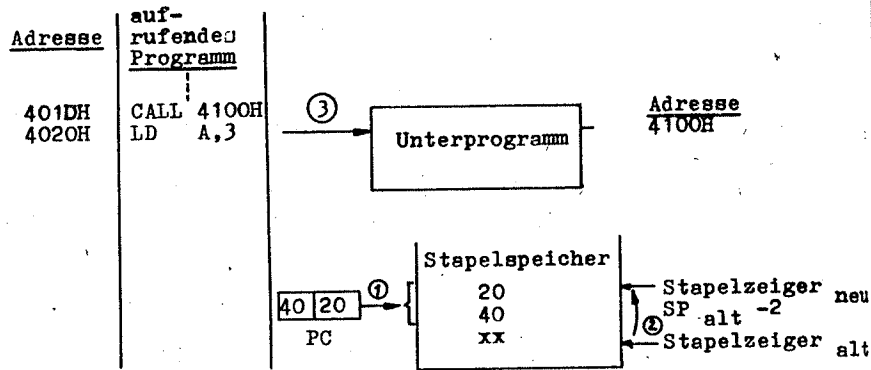


Bild 5.2. Unterprogrammaufruf

Nach dem CALL mn-Befehl werden die Befehle des Unterprogrammes abgearbeitet.

Die Rückkehr aus dem Unterprogramm in das aufrufende Programm erfolgt durch Übernahme von zwei auf der Stapelspeicherspitze befindlichen Bytes in den Programmzähler und der Korrektur des Stapelzeigers um +2. Für diesen Zweck existiert der RET-Befehl (siehe Bild 5.3.).

Mnemonic:	RET	(return = Rückkehr)
Befehlskode:		
binär	1 1 0 0 1 0 0 1	
hex	C 9	
Wirkung:	Verzweigung zu der Adresse, die an der Spitze des Stapelspeichers steht. D.h., der Programmzähler PC wird mit dem Inhalt der Speicherzellen geladen, auf die der Stapelzeiger SP zeigt:	
	PC <sub>L</sub>	— (SP)
	PC <sub>H</sub>	— (SP+1)
	Anschließend wird der Stapelzeiger um zwei erhöht:	
	SP: = SP+2	

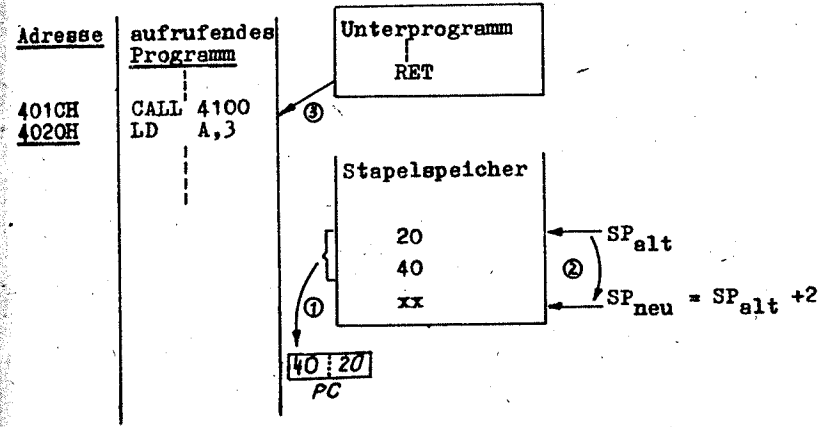


Bild 5.3. Rückkehr aus dem Unterprogramm

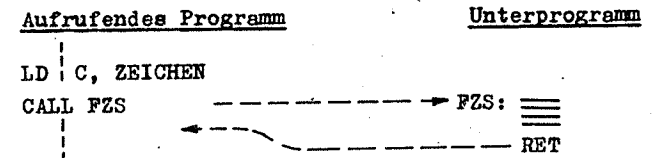
### 5.2. Übergabe von Variablen an Unterprogramme - Rettung der Registerzustände.

Unterprogramme sind in der Regel Aktionsprogramme. Sie verarbeiten übergebene Daten, senden sie an periphere Einrichtungen, lesen von diesen Daten für die CPU ein oder liefern ohne Peripheriekontakte Verarbeitungsergebnisse zurück. In all diesen Fällen sind Daten zwischen aufrufenden und Unterprogrammen zu übergeben. Dafür existieren eine Reihe von Möglichkeiten, von denen wir drei diskutieren wollen.

#### 5.2.1. Variablenübergabe in den CPU-Registern

Im einfachsten Fall werden vor allem die Universalregister, der Akkumulator oder auch die Flags der CPU als Übergabespeicher verwendet. Durch die Übergänge von und zum Unterprogramm werden diese ja nicht verändert.

**Beispiel:** Ein Unterprogramm FZS des Monitors, das genau ein Zeichen zum Fernschreiber sendet, verwendet das Register C zur Übergabe dieses Zeichens.



Vorteilhaft bei dieser Methode ist ihr geringer programmtechnischer Aufwand und damit auch der geringe zeitliche Aufwand. Soweit die Anzahl der zu übergebenden Variablen klein ist und auch keine anderen Gründe dagegen sprechen, sollte diese Methode den anderen vorgezogen werden.

5.2.2. Variablenübergabe auf Speicherplätzen

Größere Datenmengen können nicht in den CPU-Registern übergeben werden. Dazu können Speicherplätze verwendet werden, deren Adressen dem Unterprogramm entweder (a) von vornherein bekannt sind (absolute Adressierung), (b) über einen sogenannten Adreßzeiger, der in einem CPU-Registerpaar angeordnet ist, mitgeteilt oder (c) mit Hilfe des Stapelzeigers (SP) ermittelt werden.

Beispiel: Einem Unterprogramm (VERARB) sollen 9 Datenbytes zur Weiterverarbeitung übergeben werden.

Variante (a): Variablenübergabe auf festen Speicherplätzen

<u>Adresse</u>	<u>Aufrufendes Programm</u>	<u>Unterprogramm</u>
4300H 4301H  4308H	LD (WERT1), A CALL VERARB  WERT1: ; DEFINITION WERT2:  WERT9:	VERARB: LD A, (WERT1) LD A, (WERT9) RET

WERT1...WERT9 sind Symbole für bestimmte Speicherplätze, deren Adressen beiden Programmen bekannt sind.

Nachteil dieser Methode ist die Notwendigkeit, daß bei jedem Unterprogrammaufruf dieselben Speicherplätze zur Parameterübergabe verwendet und damit auch bei jedem Aufruf geändert werden müßten (somit ist beispielsweise eine Übergabe im ROM ausgeschlossen). Das führt u.U. zu hohem Organisationsaufwand, so daß die Variante (b) empfehlenswert ist.

Variante (b): Variablenübergabe auf indiziert adressierten Speicherplätzen

<u>Aufrufendes Programm</u>	<u>Unterprogramm</u>
LD IY, TABELLE ; IY enthält ; Anfangsadresse ; der Variablenliste	VERARB: LD A, (IY) LD A, (IY+8) RET

<u>CALL</u>	<u>VERARB</u>	<u>VERARB:</u>
TABELLE: ; Definitionen	LD A, (IY) ; lade ersten Wert LD A, (IY+8) ; lade neunten Wert RET	LD A, (IY) ; lade ersten Wert LD A, (IY+8) ; lade neunten Wert RET

Das Spezialregister IY (16 Bits!) fungiert als Adreßzeiger für die Parametertabelle, d.h. es enthält die Adresse des ersten Variablentabellenplatzes. Zu den Werten auf den weiteren Tabellenplätzen wird durch Addition eines Indexwertes zu dieser Basisadresse (IY+d) zugegriffen.

Auf ähnliche Weise läßt sich auch der Stapelzeiger SP als Basisadresse für eine Wertetabelle verwenden, worauf im Arbeitsbuch Teil II noch näher eingegangen wird.

5.3. Gestaltung und Schachtelung (Nesting) von Unterprogrammen

Unterprogramme besitzen meist genau einen Eintrittspunkt und einen oder mehrere Austrittspunkte.

Der Eintrittspunkt (Entry point) eines Programmes ist die Startadresse des Programmes (sie muß nicht immer die Adresse des ersten Befehles sein!). Als Austrittspunkte eines Programmes werden die Adressen der Befehle bezeichnet, mit denen das Programm verlassen wird (z.B. RET-Befehle bei Unterprogrammen).

Das Verlassen von Unterprogrammen geschieht normalerweise mit RET-Befehlen.

Häufig wird von Programmen, die Unterprogramme aufrufen, vorausgesetzt, daß nach der Abarbeitung des Unterprogrammes bestimmte oder sogar sämtliche CPU-Register noch die gleichen unveränderten Werte wie beim Aufruf des Unterprogrammes aufweisen. Nicht selten entstehen aus der Nichtbeachtung dieser Tatsache "unerklärliche" Programmfehler. Zur Beschreibung eines UP gehört deshalb stets die Angabe der zerstörten Register. Soll eine Zerstörung von Registern vermieden werden, ist folgende Gestaltung von UPs sinnvoll:

```

UP1:  PUSH  AF      ; Rettung sämtlicher
      PUSH  BC      ; Universalregister, des
      PUSH  DE      ; Akkum. und der Flags
      PUSH  HL      ; auf den Stapelspeicher,
      ; Befehle des UP1:
      ; Jetzt können sämtliche
      ; Register verwendet werden

      ;
      ;
      ;
      POP  HL      ; Rückschreiben der Register-
      POP  DE      ; inhalte entsprechend dem
      POP  BC      ; Stand beim UP-Aufruf
      POP  AF
      RET          ; Verlassen des UP
    
```

Mit Hilfe der PUSH- und POP-Befehle werden zu Beginn des Unterprogrammes sämtliche (oder auch nur ein Teil der) CPU-Registerinhalte in den Stapelspeicher geschrieben. Nach Ausführung des UP wird in umgekehrter Reihenfolge (!) (LIPO-Prinzip des Stapels) das Rückschreiben der CPU-Register vorgenommen, so daß sie wieder denselben Wert wie beim UP-Aufruf besitzen. Im UP selbst können die CPU-Register beliebig verwendet werden. Lediglich bei Manipulationen des Stapelzeigers ist größte Sorgfalt walten zu lassen. Ebenso muß die Anzahl der PUSH-Befehle genau der Anzahl der POP-Befehle entsprechen. Wird das nicht beachtet, so kehrt der Stapelzeiger nach Verlassen des UP nicht wieder in seine Ausgangslage zurück und es kommt zum Über- oder Unterlauf des Stapelspeichers, was schließlich zur Zerstörung von Programmen und Variablen führt.

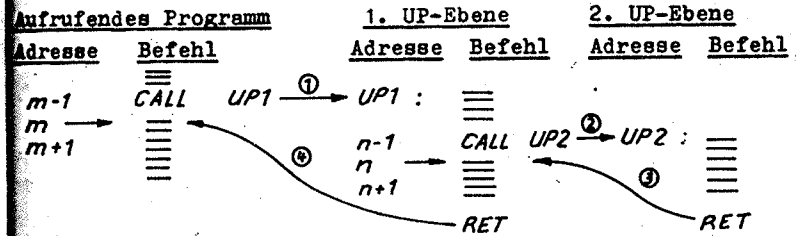
Z.B. würde ein Unterprogramm der Gestalt

```

UP:   PUSH HL
      ;
      RET
    
```

bei jedem Aufruf den Stapelspeicher um zwei Bytes vergrößern, so daß nach 1000 Aufrufender Stapelspeicher 2000 Plätze umfaßt und außerdem der RET-Befehl nicht mehr zur richtigen Adresse sondern zum HL-Inhalt zurückführen würde.

Aufgrund der flexiblen Stapelstruktur können Unterprogramme selbst Unterprogramme aufrufen, d.h. folgende Programmfolgen sind zulässig (Bild 5.4.):



Stapelspeicheraktivitäten:

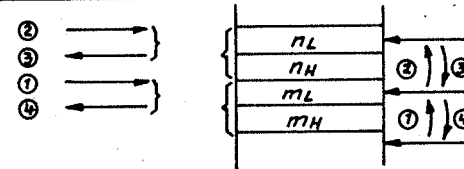


Bild 5.4 Unterprogrammenschachtelung

Damit wird ein UP selbst zum aufrufendem Programm. Je nach der Anzahl der Schachtelungsschritte (-tiefe) unterscheiden wir mehrere Programmebenen (bzw. -niveaus). Prinzipiell ist die Schachtelungstiefe nur durch die Größe des Stapelspeicherbereiches begrenzt. Bis zu einem gewissen Grade (2-4 Ebenen) erhöht die Einführung mehrerer Programmebenen die Übersichtlichkeit des Programmes, wesentlich mehr Ebenen bewirken allerdings das Gegenteil.

5.4. Bedingte Ausführung von CALL und RET

Ebenso wie die Ausführung von Sprungbefehlen vom Zustand der Flags abhängig gemacht werden kann, können bei CALL- und RET-Befehlen Ausführungsbedingungen gestellt werden. Die wesentlichsten sind auch hier

- Ausführung bei
- (a) C-Flag=1 oder
  - (b) C-Flag=0 oder
  - (c) Z-Flag=1 oder
  - (d) Z-Flag=0

Die entsprechenden Befehle dazu sind:

**Bedingter CALL:**

	a	b	c	d
Mnemonic:	CC nn	CNC nn	CZ nn	CHZ nn
Befehlskode: (hex)	DC n n	D4 n n	CC n n	C4 n n
Wirkung:	Aufruf des UP auf Adresse nn, wenn Bedingung erfüllt, sonst Fortsetzung beim nächsten Befehl.			

**Bedingter RET:**

	a	b	c	d
Mnemonic:	RC	RNC	RZ	RNZ
Befehlskode: (hex)	DB	DO	CB	CO
Wirkung:	Verlassen des UP wenn Bedingung erfüllt, sonst Fortsetzung beim nächsten Befehl.			

**Beispiel:** Ein Unterprogramm realisiert eine bestimmte Zeitlänge, indem es den Inhalt eines Registers herunterzählt und bei Erreichen des Wertes Null verlassen wird.

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	06, 05	APR: LD B, 5	; Aufrufendes Programm
4002	0D, 10, 40	CALL UP	; lädt Zählkonstante nach
4005	C3, 00, 40	JMP APR	; Register B
4010	05	UP: DEC B	; Unterprogramm; B:=B-1
4011	08	RZ	; Rückkehr, wenn B=0
4012	C3, 10, 40	JMP UP	; Zyklus

Das Programm ist auf den POLY-COMPUTER im Einzelzyklusbetrieb auszuführen!

**5.5. Die für den Anwender nutzbaren Unterprogramme des Monitors**

Für die Bedienung der Peripherie des POLY-COMPUTERS werden dem Nutzer durch den Monitor eine Vielzahl von Kommandos und Funktionen zur Verfügung gestellt, die für die Programmtestung wichtig sind. Will darüber hinaus der Nutzer in seinen eigenen Programmen die Peripherie (z.B. Tastatur und Displayelemente) in anderer Weise verwenden, sollte er auf die im Monitor vorhandenen Unterprogramme zu deren Ansteuerung zurückgreifen. Er hat damit die Gewissheit, auf dieser untersten Programmebene fehlerfreie Programme anzuwenden und alle spezifischen Forderungen der Peripherie (besonders kritisch beim Display) zu erfüllen. Die Monitorunterprogramme sind so gestaltet, daß keine Einschränkungen für die Nutzung der POLY-COMPUTER-Peripherie eintreten.

Es werden folgende fünf Routinen angeboten:

- KONSOL - einmalige Tastaturabfrage und Displayauffrischung
- ANZDEC - Tabelle der Siebensegmentcodes für Hexaziffern
- PUNKAN - füllt die Speicherzellen für die zwei linken Anzeigestellen, löscht die übrigen
- ZIFANZ - füllt die Speicherzellen von zwei rechten Anzeigestellen und verschiebt die rechten sechs Stellen um zwei nach links
- RDYANZ - Bereitet die Anzeige der Zeichen "READY."

Während diese fünf UPs ständig im ROM des POLY-COMPUTERS enthalten sind, müßten die beiden Fernschreiber Routinen vor ihrer Verwendung in den RAM eingegeben werden (über Tastatur oder Magnetband). Die Programmlisten dieser Programme sowie ausführliche Verwendungshinweise sind im Bedienhandbuch enthalten. In einigen der folgenden Kapitel finden diese UPs in Beispielen Anwendung.

**5.6. Zusammenfassung**

Die Unterprogrammtechnik stellt ein Programmgestaltungsmittel dar, das zu effektiven (speicherplatzsparenden) und gut gegliederten Programmen wesentlich beiträgt. Die gezeigten Möglichkeiten zur Variablenübergabe an UPs sowie allgemein zur Nutzung des Stapelspeichers sind lediglich eine (repräsentative) Auswahl aus einer Fülle von Möglichkeiten.

Folgende Fragen und Aufgaben sollten gewissenhaft beantwortet bzw. gelöst werden:

- 1) Was verstehen wir unter dem Begriff Unterprogramm?
- 2) Welche Vorteile bietet die Unterprogrammtechnik und wann wird sie angewendet?
- 3) Welche Abläufe vollziehen sich bei einem UP-Aufruf?
- 4) Wie wird ein UP verlassen?
- 5) Welche Möglichkeiten zur Übergabe von Variablen an ein UP kennen Sie? Welche ist wann sinnvoll anzuwenden?
- 6) Welche Größe kann der Stapelspeicherbereich beim Einsatz der CPU U880 maximal annehmen? Wie viele UPs können ineinander maximal geschaltet werden?
- 7) Was ist unter dem Begriff "Eintrittspunkt" zu verstehen?
- 8) Welcher Fehler steckt im folgenden UP?
 

PUSH	BC
PUSH	DE
LD	A,5
POP	DE
RET	
- 9) Schreiben Sie ein Unterprogramm, daß die Werte der Register B und C addiert und die Summe ins Register D übergibt und testen Sie dieses Programm auf dem POLY-COMPUTER!

### 6. Logische und Bitbefehle

Mikrorechner werden meist zur Prozeßsteuerung eingesetzt. Bei derartigen Anwendungen ist häufig auf Signale zu reagieren bzw. sind Steuerinformationen zu verarbeiten und auszugeben, die als einzelne Bits in einem Byte auftreten.

Z.B. könnte ein zu überwachender Prozeß ein Statuswort (repräsentiert den aktuellen Zustand des Prozesses) an den Rechner übergeben, dessen Bits folgende Bedeutung zugeordnet ist:

H	E/A	DH	DN	frei	frei	TH	TN
D7	D6	D5	D4	D3	D2	D1	D0
<u>Statuswort</u>							

- H: 0 - normal  
1 - Havarie
- E/A: 0 - Prozeß aus  
1 - Prozeß ein
- DH: 1 - Drehzahl zu hoch
- DN: 1 - Drehzahl zu niedrig
- TH: 1 - Temperatur zu hoch
- TN: 1 - Temperatur zu niedrig.

Der Rechner muß nun in der Lage sein, diese Bitinformationen auszuwerten und wenn nötig, entsprechende Aktionen einzuleiten. Für derartige sowie arithmetische Problemstellungen besitzen Mikroprozessoren eine Reihe von logischen und bitmanipulierenden Befehlen.

#### 6.1. Rotations- und Verschiebepfehle

Rotations- und Verschiebepfehle transportieren Bits eines Bytes (in Registern oder Speicherplätzen) auf eine der benachbarten Bitpositionen. Sie beeinflussen sämtliche Flags.

Während es sich bei Rotationsbefehlen um geschlossene "Bitkreisläufe" handelt (z.B. Rückkopplung von Bit 7 auf Bit 0), realisieren Verschiebepfehle offene Schiebeketten (Bild 6.1.)

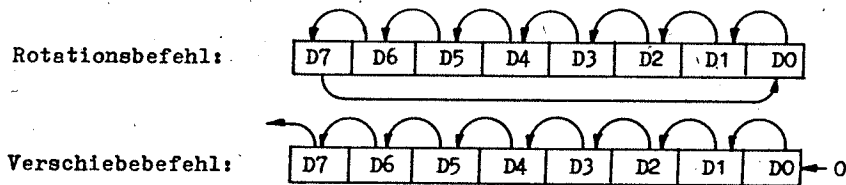


Bild 6.1. Beispiel für Rotations- und Verschiebefehle

Sinnvollerweise wird bei den meisten dieser Befehle das Übertrags-Flip-Flop (Carry-Flag) als erstes oder letztes Bit der Schiebekette einbezogen, so daß die Auswertung der Bits z.B. über bedingte Sprünge erfolgen kann.

### 6.1.1. Rotieren im A-Register

Die folgenden Befehle sind nur im A-Register ausführbar.

Mnemonic: RLCA (Rotieren links in A)

Befehlscode: 07 (hex)

Wirkung:

Der A-Inhalt wird um eine Bitposition nach links geschoben. Der Inhalt des höchstwertigsten Bits gelangt zu Bit 0 und in das Carry-Flag.

Mnemonic: RRCA (Rotieren rechts in A)

Befehlscode: 0F (hex)

Wirkung:

Der A-Inhalt wird um eine Bitposition nach rechts geschoben. Der Inhalt des niederwertigsten Bits gelangt zu Bit 7 und in das Carry-Flag.

**Beispiel:** Im Dezimalsystem bedeutet eine Linksverschiebung einer Zahl (entspricht dem Anhängen einer Null) einer Multiplikation mit 10.

$$210 \longleftarrow 21$$

Eine Rechtsverschiebung ist demzufolge gleichzusetzen mit einer Division durch 10.

$$210 \longrightarrow 21$$

In Binärsystem verhalten sich die Zahlen analog, nur das der

Binär:	11	→	110
Dezimal:	3	x2=	6
Binär:	110	→	11
Dezimal:	6	:2=	3

Eine Verschiebung einer Binärzahl um eine Bitposition nach links, entspricht einer Multiplikation mit 2.  
 Eine Verschiebung einer Binärzahl um eine Bitposition nach rechts, entspricht einer Division durch 2.

Auf diese Weise lassen sich Multiplikationen und Divisionen mit einer Potenz von 2 auf Rotationen bzw. Verschiebungen zurückführen.

Z.B. entspricht einer Multiplikation mit 8 einer dreimaligen Linksverschiebung.

**Festlegung:** Bei Rotations- und Verschiebefehlen bedeutet links in Richtung höherwertiger Bitstellen und rechts in Richtung niederwertiger Bitstellen.

Die nun folgenden Befehle RLA und RRA unterscheiden sich zu RLCA und RRCA nur dadurch, daß das Carry-Flag als neuntes Bit in die Schiebekette eingeschleift wird.

Mnemonic: RLA (Rotiere links in A durch Carry)

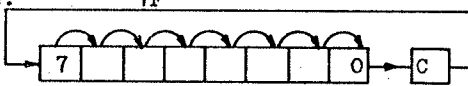
Befehlscode: 17 (hex)

Wirkung:

Der A-Inhalt wird um eine Bitposition nach links geschoben. Das höchstwertigste Bit gelangt in das C-Flag und der C-Flag-Inhalt in das Bit 0.

Mnemonic: RRA (Rotiere rechts in A durch Carry)

Befehlskode: 1F (hex)

Wirkung: 

Der A-Inhalt wird um eine Bitposition nach rechts geschoben. Das niederwertigste Bit gelangt in das C-Flag und der C-Flag-Inhalt in das Bit 7.

Mit Rotierbefehlen dieser Art sind Verschiebungen von Werten größer 8 Bits möglich.

6.1.2. Rotieren in verschiedenen CPU-Registern und in Speicherplätzen

In Verallgemeinerung der im Abschnitt 6.1.1. erläuterten Rotierbefehle existieren Befehle mit prinzipiell gleicher Wirkung, die aber auch auf andere Register und Speicherplätze wirken.

Mnemonic: RLC m (Rotieren links; m=A,B,C,D,E,H,L,(HL),(IX+d),(IY+d))

Befehlskode: 

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
07	00	01	02	03	04	05	06	06	CB	CB
									d	d
									06	06

Wirkung: Genau wie RLCA im angegebenen Ziel

Mnemonic: RRC m (Rotieren rechts; m=A,B,C,D,E,H,L,(HL),(IX+d),(IY+d))

Befehlskode: 

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
0F	08	09	0A	0B	0C	0D	0E	0E	CB	CB
									d	d
									0E	0E

Wirkung: Genau wie RRCA im angegebenen Ziel

Mnemonic: RL m (Rotieren links durch Carry m=A,B,C,D,E,H,L,(HL),(IX+d),(IY+d))

Befehlskode: 

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
17	10	11	12	13	14	15	16	16	CB	CB
									d	d
									16	16

Wirkung: Genau wie RLA im angegebenen Ziel

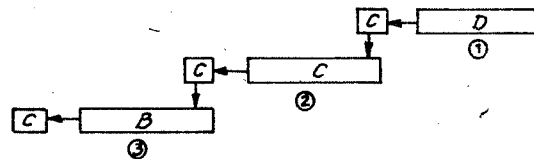
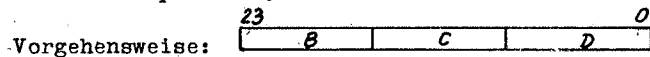
Mnemonic: RR m (Rotieren rechts durch Carry m=A,B,C,D,E,H,L,(HL),(IX+d),(IY+d))

Befehlskode: 

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
1F	18	19	1A	1B	1C	1D	1E	1E	CB	CB
									d	d
									1E	1E

Wirkung: Genau wie RRA im angegebenen Ziel

Beispiel: Eine 24 Bits umfassende Binärzahl (in den Registern B, C, und D untergebracht) soll verdoppelt werden (entspricht einer Linksverschiebung um eine Bitposition).



Die Aufgabe ist mit Hilfe des RL m.- Befehles lösbar. Das jeweils aus dem Register herausgeschobene Bit gelangt bei der nächsten Rotation über das C-Flag auf Bitposition Null des höherwertigen Bytes.

Programm: Verdopplung einer 24 Bit-Binärzahl in B,C,D

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	AF	MULT2: XOR A	; C-Flag:=0
4001	CB, 12	RL D	; linksrot. D
4003	CB, 11	RL C	; linksrot. C
4005	CB, 10	RL B	; linksrot. B

6.1.3. Verschiebepfehle

Der Vollständigkeit halber seien die drei Typen von Verschiebepfehlen noch kurz erläutert.

**Mnemonic:** SLA m (Verschiebung (shift) links arithmetisch, m=A,B,C,D,F,H,L, (HL), (IX+d), (IY+d))

**Befehlscode: (hex)**

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
27	20	21	22	23	24	25	26	26	CB	CB
									d	d
									26	26

**Wirkung:**

C-Flag r, (HL), (IX+d), (IY+d)

Der Speicherplatz- oder Registerinhalt wird um eine Bitposition nach links verschoben. In Bitposition 0 wird stets eine 0 nachgeschoben, Bitposition 7 wird in das C-Flag geschoben.

**Mnemonic:** SRA m (Verschiebung (shift) rechts arithmetisch, m=A,B,C,D,E,H,L, (HL), (IX+d), (IY+d))

**Befehlscode:**

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
2F	2B	29	2A	2B	2C	2D	2E	2E	CB	CB
									d	d
									2E	2E

**Wirkung:**

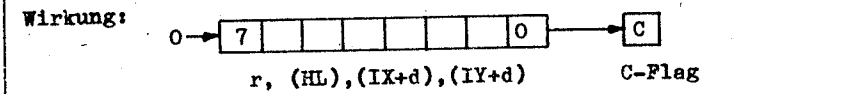
r, (HL), (IX+d), (IY+d) C-Flag

Der Speicherplatz- oder Registerinhalt wird um eine Bitposition nach rechts verschoben. In Bitposition 7 bleibt der Wert erhalten, der Inhalt von Bitposition 0 gelangt in das C-Flag.

**Mnemonic:** SRL m (Verschiebung (shift) rechts logisch, m=A,B,C,D,E,H,L, (HL), (IX+d), (IY+d))

**Befehlscode: (hex)**

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
3F	3E	39	3A	3B	3C	3D	3E	3E	CB	CB
									d	d
									3E	3E



Der Speicherplatz- oder Registerinhalt wird um eine Bitposition nach rechts verschoben. Bitposition 7 wird Null, Inhalt von Bitposition 0 wird ins C-Flag geschoben.

**Beispiel:** Die Binärzahl auf Speicherplatz 4102H ist mit 4 zu multiplizieren und es ist zu überprüfen, ob die Zahl danach größer als in 8 Bits darstellbar geworden ist.

Programm:

Adresse (hex)	Befehlscode (hex)	Mnemonic	Kommentar
4000	DD, 21, 02, 41	MULT4: LD IX, 4102	; Fülle IX mit der Speicheradresse
4004	DD, CB, 00, 26	SLA (IX+0)	; (4102):=2x(4102)
4008	DA, 08, 40	M1: JC M1	; Übertrag 2? ja → Schleife
400B	DD, CB, 00, 26	SLA (IX+0)	; (4102):=2x(4102)
400F	DA, 0F, 40	M2: JC M2	; Übertrag ? ja → Schleife
4012	C3, 12, 40	M3: JMP M3	; Schlußschleife wenn kein Übertrag

Beim Test auf den POLY-COMPUTER ist vor dem Start des Programmes der Speicherplatz 4102 mit einem sinnvollen Binärwert zu füllen. Nach dem Programmstart (GO 4000H) kreist das Programm in einer der drei Schleifen. Mit MON ist die Schleife zu unterbrechen und das Ergebnis kann in Speicherzelle 4102H abgelesen werden. Trat ein Übertrag bereits bei der ersten Verschiebung auf, so wird eine zweite nicht ausgeführt.



6.2. Bitbefehle

Der Zustand einzelner Bits der CPU-Register bzw. Speicherplätze kann mit Hilfe der Bitbefehle getestet oder verändert (Setzen auf 1, Rücksetzen auf 0) werden.

Der Bittest verändert das untersuchte Bit nicht. Der Zustand des Bits wird lediglich in das Z-Flag geladen, welches anschließend (z.B. durch bedingte Sprung-, Ruf- oder Rückkehrbefehle) ausgewertet werden kann.

Mnemonic: Bit b, m (Bittest, b-Bitposition  $0 \leq b \leq 7$ ,  $m=A,B,C,D,E,H,L,(HL), (IX+d), (IY+d)$ )

Befehlskode: (hex)

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
xx	xx	xx	xx	xx	xx	xx	xx	YJ	d	d
									YY	YY

xx = 01 (-b-)(-r-) binär  
yy = 01 (-b-)110 binär

b=Binärwert:  $000 \leq b \leq 111$   
r-Registerkode: 

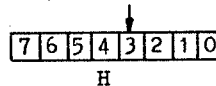
A	B	C	D	E	H	L
7	0	1	2	3	4	5

  
d-Verschiebung

Wirkung: Der zu untersuchende Bitinhalt wird negiert und in das Z-Flag eingeschrieben.

D.h. Bit=0  $\longrightarrow$  Z= $\overline{\text{Bit}}$   
Bit=1  $\longrightarrow$  Z=0

Beispiel: In Abhängigkeit vom Zustand von Bit 3 des Registers H soll das A-Register gelöscht (Bit  $H_3 = 1$ ) oder mit FFH gefüllt werden (Bit  $H_3 = 0$ ).



Programm:

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	CB, 5C	TESTB: BIT 3,H	; Teste Bit $H_3$ , Z= $\overline{H_3}$
4002	CA, 09, 40	JZ T1	; Sprung, wenn Bit $H_3 = 0$
4005	AF	XOR A	; Lösche A-Register
4006	C3, 06, 40	TQ: JMP TO	; Schlußschleife
4009	3E, FF	T1: LD A, 0FFH	; A:=FFH
400B	C3, 06, 40	JMP TO	

Vor dem Programmtest auf dem POLY-COMPUTER ist das H-Register entsprechend zu füllen, z.B. 1. Test H:=8  
2. Test H:=0

Zur Manipulation einzelner CPU-Register- oder Speicherbits stehen die Befehle SET (Bit:=1) und RES (Bit:=0) zur Verfügung.

Mnemonic: SET b, m (Setzen eines Bits auf 1 b-Bitposition,  $0 \leq b \leq 7$ ,  $m=A,B,C,D,E,H,L,(HL), (IX+d), (IY+d)$ )

Befehlskode: (hex)

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
xx	xx	xx	xx	xx	xx	xx	xx	YJ	d	d
									YY	YY

xx = 11(-b-)(-r-) (binär)  
yy = 11(-b-)110

b=Binärwert:  $000 \leq b \leq 111$   
r-Registerkode: 

A	B	C	D	E	H	L
7	0	1	2	3	4	5

  
d-Verschiebung

Wirkung: Das adressierte Bit wird auf 1 gesetzt.

Mnemonic: RES b, m (Rücksetzen eines Bits auf 0; b-Bitposition:  $0 \leq b \leq 7$ ,  $m=A,B,C,D,E,H,L,(HL), (IX+d), (IY+d)$ )

Befehlskode: (hex)

m	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
CB	CB	CB	CB	CB	CB	CB	CB	CB	DD	FD
xx	xx	xx	xx	xx	xx	xx	xx	YJ	d	d
									YY	YY

xx = 10(-b-)(-r-) (binär)  
yy = 10(-b-)110 (binär)

b=Binärwert:  $000 \leq b \leq 111$   
r-Registerkode: 

A	B	C	D	E	H	L
7	0	1	2	3	4	5

  
d-Verschiebung

Wirkung: Das adressierte Bit wird auf 0 rückgesetzt.

Beispiel: Das Bit 0 des Speicherplatzes 4102H soll negiert werden, d.h. wenn das Bit gleich 1 ist, soll es rückgesetzt werden und wenn es gleich 0 ist, soll es gesetzt werden.

**Programm:**

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	21, 02, 41	NEGB: LD HL,4102H	; HL:=Speicheradresse
4003	CB, 46	BIT 0,(HL)	; Test Bit 0 von 4102H
4005	CA, 0D, 40	JZ N2	; Sprung bei Bit 0=0
4008	CB, 86	RES 0,(HL)	; Rücksätzen von Bit 0 (: =0)
400A	C3, 0A, 40	N1: JMP N1	; Schlußschleife
400D	CB, C6	N2: SET 0,(HL)	; Setzen von Bit 0 (: =1)
400F	C3, 0A, 40	JMP N1	; Sprung zur Schlußschleife

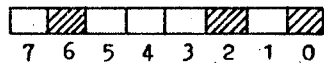
Vor und nach jeder Ausführung dieses Programmes auf den POLY-COMPUTER ist Bit 0 vom Speicherplatz 4102H verschieden.

**6.3. Logikbefehle**

Logikbefehle realisieren die im Abschnitt 1.2.2. vorgestellten Operationen im Binärsystem, also die Funktionen UND, ODER, EXCLUSIV- ODER, NEGATION sowie den Vergleich zweier Binärwerte. Diese Funktionen werden in Steuerprogrammen außerordentlich vielfältig angewendet.

Beispielsweise kann die UND-Verknüpfung zweier Binärwerte zur "Maskierung" einzelner Bits genutzt werden.

**Beispiel:** Für den Start einer Prozeßoperation müssen drei bestimmte Signale (von insgesamt 8 Signalen) inaktiv (Null) sein. Diese 8 Signale werden durch die 8 Bits eines Bytes repräsentiert. Die interessierenden drei Signale befinden sich auf den Bitpositionen 0, 2 und 6.

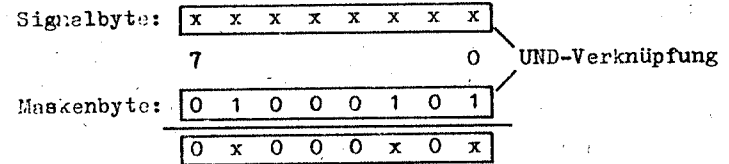


Der Test für den Prozeßstart muß lediglich diese drei Bits prüfen und zwar muß die Bedingung

$$\text{Signal } 0 \wedge \text{Signal } 2 \wedge \text{Signal } 6 = 0$$

erfüllt sein.

Dazu wird über das zu untersuchende Datenbyte eine "Maske" gelegt, die genau diese drei Bitstellen unverändert läßt und die übrigen (nicht interessierenden Signalbits) auf Null setzt.



Bei einer solchen UND-Verknüpfung wird das Z-Flag beeinflusst. Ist es gesetzt (d.h. Ergebnis ist auf allen Bitpositionen gleich Null), so ist die gestellte Bedingung erfüllt.

Nachfolgend werden die fünf logischen Befehlstypen erläutert. Ein Operand befindet sich stets im A-Register! Alle Flags werden beeinflusst!

Mnemonic: AND s (UND-Verknüpfung zwischen A-Register und s  
s=A,B,C,D,E,H,L,n, (HL), (IX+d), (IY+d))

Befehlskode: (hex)

s	A	B	C	D	E	H	L	n	(HL)	(IX+d)	(IY+d)
A7	A0	A1	A2	A3	A4	A5	A6	E6	A6	DD	FD
								n		A6	A6
										d	d

Wirkung: Es wird eine UND-Verknüpfung zwischen dem A-Inhalt und s ausgeführt; das Ergebnis steht im A-Register

$$A := A \wedge s$$

**Beispiel:** Das obige Maskierungsproblem entspricht folgendem Programm: (das Signalbyte sei im Register B)

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	3E, 45	MASK: LD A,45H	; Lade A mit der Maske 45H=01000101B
4002	A0	AND B	; Maskierung, A:=A^B
4003	CA, 09, 40	JZ M1	; Sprung, wenn A=0
4006	C3, 06, 40	MO: JMP MO	; Bedingung nichterfüllt
4009	C3, 09, 40	M1: JMP M1	; Bedingung erfüllt

Mnemonic: OR s (ODER-Verknüpfung zwischen A-Register und s, s=A,B,C,D,E,H,L,n,(HL), (IX+d), (IY+d))

Befehlskode: 

s	A	B	C	D	E	H	L	n	(HL)	(IX+d)	(IY+d)
	B7	B0	B1	B2	B3	B4	B5	P6	B6	DD	FD
								n		B6	B6
										d	d

Wirkung: Es wird eine ODER-Verknüpfung zwischen dem A-Inhalt und s ausgeführt, das Ergebnis steht im A-Register

A : = A v s

Mnemonic: XOR s (EXCLUSIV-ODER-Verknüpfung zwischen A-Register und s, s=A,B,C,D,E,H,L,n,(HL), (IX+d), (IY+d))

Befehlskode: 

s	A	B	C	D	E	H	L	n	(HL)	(IX+d)	(IY+d)
	AF	AB	A9	AA	AB	AC	AD	EE	AE	DD	FD
								n		AE	AE
										d	d

Wirkung: Es wird eine EXCLUSIV-ODER-Verknüpfung zwischen dem A-Inhalt und s ausgeführt, das Ergebnis steht im A-Register

A : = A ^ s

Die XOR-Operation diente uns bisher bereits zum Löschen des A-Registers: XOR A → A:=0.

Darüber hinaus ist sie vorteilhaft zur selektiven Negation einzelner Bits einsetzbar. Dazu wird wiederum als ein Operand eine "Maske" verwendet, die genau an den zu negierenden Bitpositionen eine 1 besitzt und sonst Nullen enthält.

Beispiel: Byte: 

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

Maske: 

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

01010101  
↑ ↑ ↑ ↑  
negiert gegenüber  
Byte

Mnemonic: CMP, S (Vergleiche (compare) A-Inhalt mit s, s=A,B,C,D,E,H,L,n,(HL), (IX+d), (IY+d))

Befehlskode: 

s	A	B	C	D	E	H	L	n	(HL)	(IX+d)	(IY+d)
	BF	B8	B9	BA	BB	BC	BD	FE	BE	DD	FD
								n		BE	BE
										d	d

Wirkung: In der CPU läuft die Operation A-s ab, ohne daß A oder s verändert werden. Entsprechend dem Ergebnis dieser Subtraktion werden die Flags gestellt.

Erläuterung: Das Ergebnis eines Vergleichsbefehls (CP) ist folgendermaßen interpretierbar:

A - s = 0 ⇒ Z - Flag: = 1

d.h. wenn A=s ist Z=1

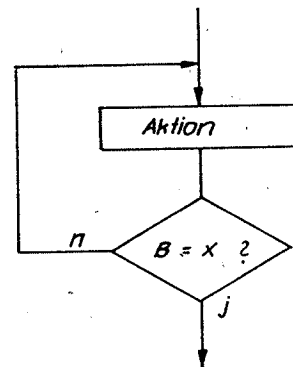
A - s > 0 ⇒ C - Flag: = 0, Z - Flag: = 0

d.h. wenn A > s, dann  $\frac{C}{F} = \frac{0}{0}$

A - s < 0 ⇒ C - Flag: = 1 (es tritt Übertrag auf!)

d.h. wenn A < s, dann C = 1

Beispiel: Ein Zyklus soll abgebrochen werden, wenn eine Variable einen bestimmten Wert (ungleich Null) annimmt. Diese Variable befinde sich im Register B.



In unserem Beispiel besteht die konkrete Aktion lediglich in der Erhöhung des B-Inhaltes um eins.

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	06, 00	VERGL: LD B,0	; Lösche B; B:=0
4002	04	ZYKL: INC B	; B:=B+1
4003	3E, 05	LD A,5	; A:=5
4005	B8	CMP B	; Vgl. A mit B A-B=?
4006	C2, 02, 40	JNZ ZYKL	; A=B?, Sprung wenn A≠B
4009	C3, 09, 40	Z1: JMP Z1	; Stoppschleife

Schließlich seien noch die Negation und die Komplementbildung vorgestellt:

Mnemonic:	CPL	( <u>complement</u> )
Befehlskode:		
binär	0 0 1 0 1 1 1 1	
hex	2 F	
Wirkung:	Die Bits des A-Registers werden in den entgegengesetzten logischen Zustand versetzt	
	A := $\overline{A}$	

Beispiel: A: 0 0 1 1 0 0 1 1  $\xrightarrow{\text{CPL}}$  1 1 0 0 1 1 0 0

Mnemonic:	NEG	( <u>negation</u> )
Befehlskode:	1 1 1 0 1 1 0 1	
binär	0 1 0 0 0 1 0 0	
hex	E D	
	4 4	
Wirkung:	Es wird das sogenannte Zweierkomplement des A-Registers gebildet:	
	A := 0 - A	
	(alle Flags werden beeinflusst)	

Beispiel: A: 0 0 1 1 0 0 1 1  
 NEG 0 0 0 0 0 0 0 0  
 - 0 0 1 1 0 0 1 1  
 1 1 1 0 0 1 1 0 1  
 C =====

Umrechnung: NEG  $\hat{A} \hat{=} \text{CPL } A + 1$

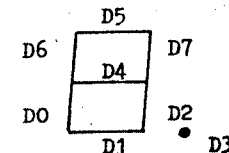
(sowie veränderter Flagzustand)

#### 6.4. Verwendung von Logik- und Bitbefehlen zur Steuerung der Tastatur und des Displays des POLY-COMPUTERS

Als abschließendes Beispiel dieses Kapitels wollen wir gleichzeitig die Anwendung zweier wichtiger UPs des Monitors, nämlich KONSOL und ZIFANZ, kennenlernen.

Dazu werden wir das Multiplikationsprogramm aus Abschnitt 4.2.3. so abändern, daß es direkt über die Tastatur bedient und das Ergebnis direkt auf dem Display abgelesen werden kann.

Wie bereits im Kapitel 5 kurz erwähnt, realisiert KONSOL eine einmalige Tastaturabfrage und eine einmalige Auffrischung des Displays. Der Tastenkode wird in A übergeben und C enthält im Falle einer Zifferntaste den Binärwert (also z.B. 0 0 0 0 1 0 1 0 bei Taste A). Angezeigt wird der Inhalt der Speicherzellen, auf die Registerpaar DE weist. D.h. auf dem linken Anzeigeelement erscheint der Inhalt von der Speicherzelle, deren Adresse im Registerpaar DE enthalten ist, auf dem benachbarten Element der Inhalt der Speicherzelle mit der Adresse (DE+1) usw. Der Nutzer muß also vor Aufruf von KONSOL sowohl DE mit einer geeigneten Speicheradresse füllen und die acht Speicherplätze (für acht Anzeigeelemente) ebenfalls. Jedes der acht Bits eines solchen Speicherplatzes repräsentiert ein Anzeigesegment, das bei Bit=1 leuchtet. Die Zuordnung der Bits ist willkürlich und sieht folgendermaßen aus:



Somit ergibt sich z.B. für die Darstellung der Ziffer 3 der Kode B6H.

Dem Tastaturcode liegt folgendes zugrunde:

Bei  $D_0=1$  wurde eine Funktionstaste betätigt,  
bei  $D_1$  oder  $D_2=1$  wurde eine Zifferntaste betätigt.

Die übrige Kodierung ist willkürlich und durch die Verdrahtung bedingt.

64	74	14	84	21	11
62	72	12	82	81	61
24	54	34	44	71	51
22	52	32	42	41	

Bild 6.2. Tastaturcode (hex)

Die übrigen vier Tasten (RESET usw.) lösen Hardware-Funktionen aus und können vom Programm nicht überprüft werden.

Zur Vereinfachung der Handhabung von KONSOL existiert das Programm ZIFANZ, welches die in den Bits  $D_0 - D_3$  des A-Registers enthaltene Binärziffer (0-F) in den entsprechenden 7-Segmentcode umsetzt und in einen Displaypuffer (adressiert durch Registerpaar HL) transportiert. Darüber hinaus werden bei jedem Aufruf die links von dieser Ziffer stehenden Stellen um eine Stelle nach links verschoben.

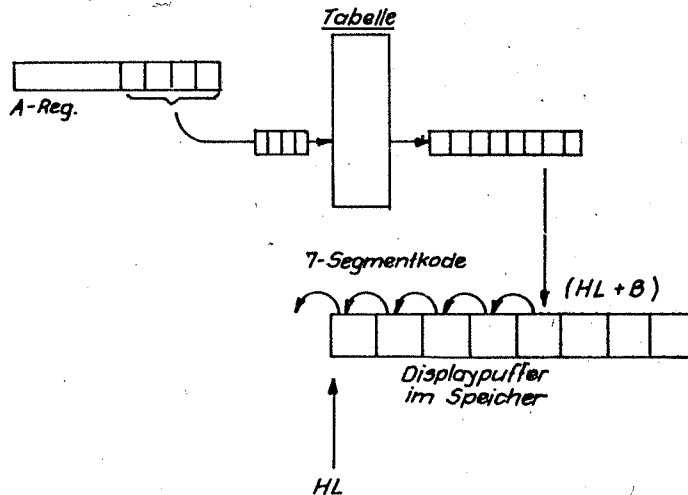
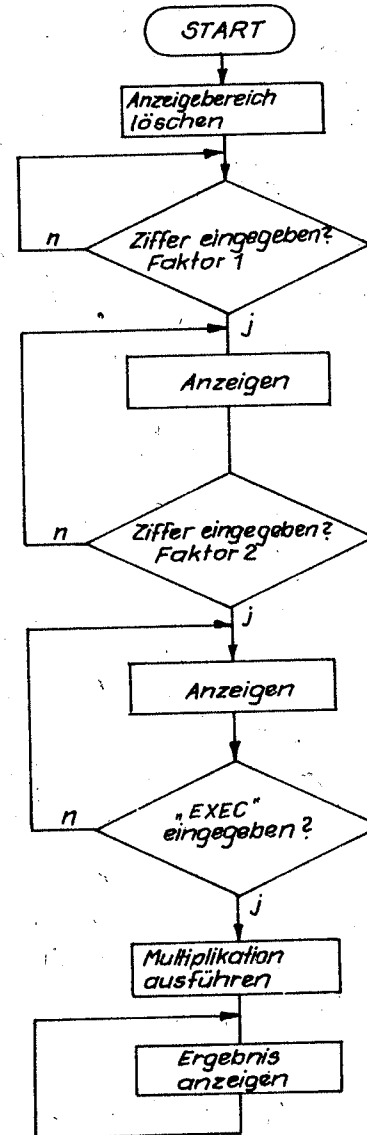


Bild 6.3. Wirkungsweise des Programmes ZIFANZ

In Register B kann angegeben werden, auf welche der acht Pufferplätze diese Hexaziffer geschrieben werden soll ( $B_1=0...7$ ).

Unser Multiplikationsprogramm soll folgenden Ablauf realisieren.



Unser Programm soll also lediglich Faktoren mit einer Ziffer multiplizieren, das Ergebnis kann zwei Ziffern umfassen. Die EXEC-Taste startet die Operation. Da das Anzeige- und Abfrageprogramm KONSOL nur eine einmalige Auffrischung der Anzeige realisiert, ist die Anzeige zyklisch vorzunehmen. Eingaben und Anzeige erfolgen hexadezimal!

Multiplikation unter Nutzung der Monitorroutinen für Tastaturabfrage und Displayanzeige: (Anzeigepuffer 4100H - 4107H)

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	31, 00, 42	MUL: LD SP,4200H	;SP:=4200H für UP-Behandlung
4003	CD, 5C, 40	CALL LOESCH	;Starte UP für Puffer löschen
4006	DD, 21, 10, 41	LD IX,4110H	;Puffer für Faktoren
400A	11, 00, 41	M2: LD DE,4100H	;Zeiger für Displaypuffer
400D	CD, 4E, 01	CALL KONSOL	;Tastaturabfrage
4010	E6, 07	AND 0000011B	;Ausblenden der Tastenbits und Flags stellen!
4012	CA, 0A, 40	JZ M2	;Zyklus, wenn keine Taste gedrückt
4015	E6, 01	AND 0000001B	;Funktionstaste?
4017	C2, 2B, 40	JNZ M3	
401A	79	LD A,C	;Eintragen der Ziffer
401B	DD, 77, 00	LD (IX),A	;im Pufferbereich
401E	DD, 23	INC IX	
4020	06, 07	LD B,7	;B:=Stellenposition
4022	21, 00, 41	LD HL,4100H	
4025	CD, 00, 03	CALL ZIFANZ	
4028	C3, 0A, 40	JMP M2	;Holen nächste Ziffer
402B	DD, 2B	M3: DEC IX	
402D	DD, 7E, 00	LD A,(IX)	;Multiplikation
4030	B7	OR A	
4031	CA, 42, 40	JZ M5	;Faktor = 0 ?
4034	4F	LD C,A	
4035	DD, 2B	DEC IX	
4037	AF	XOR A	
4038	DD, 86, 00	M4: ADD (IX)	;Teilsummenbildung
403B	OD	DEC C	
403C	C2, 38, 40	JNZ M4	
403F	CD, 5C, 40	CALL LOESCH	; Löschen Anzeige
4042	4F	M5: LD C,A	; Anzeige Ergebnis
4043	07	RLCA	

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4044	07	RLCA	
4045	07	RLCA	
4046	07	RLCA	
4047	06, 07	LD B,7	
4049	21,00,41	LD HL,4100H	
404C	CD,00,03	CALL ZIFANZ	;Übergabe höherwertige
404F	79	LD A,C	;Ziffer im Puffer
4050	CD,00,03	CALL ZIFANZ	;Übergabe niederwertige
4053	11,00,41	M6: LD DE,4100H	;Ziffer im Puffer
4056	CD,4E,01	CALL KONSOL	;Anzeige
4059	C3,53,40	JMP M6	
405C	21,00,41	LOESCH: LD HL,4100H	;Löschen Anzeigepuffer (UP)
405F	06,08	LD B,8	
4061	36,00	M1: LD (HL),00	
4063	23	INC HL	
4064	05	DEC B	
4065	C2,61,40	JNZ M1	
4068	C9	RET	

Register IX enthält die Adresse des aktuellen Faktors (zuletzt eingegeben). Es können beliebig viele Faktoren eingegeben werden, es werden aber nur die letzten beiden miteinander multipliziert. Die LOESCH-Routine wird mehrfach benötigt und wurde deshalb als Unterprogramm geschrieben.

Das Programm wird auf Adresse 4000H gestartet. Nach Ausführung des Programms gelangen wir über MON oder RESET wieder in den Monitor. Dem Anwender wird empfohlen, das Programm z.B. auch auf andere Grundrechenarten zu erweitern, und zur Operationsunterscheidung den Tastenkodex der Funktionstasten genauer auszuwerten.

Der Anzeigepuffer (über DE adressiert) kann selbstverständlich auch ohne Zuhilfenahme des Programmes ZIFANZ mit beliebigen Symbolen gefüllt werden.

Zu beachten ist ferner, daß das Programm KONSOL die Register AF, BC, DE und HL zerstört, so daß z.B. DE vor jedem neuen Aufruf neu geladen werden muß. Das Programm KONSOL benötigt für einen Durchlauf ca. 6ms (genaue Angaben siehe Bedienungshandbuch).

### 6.5. Zusammenfassung

Das umfangreiche Repertoire an bitmanipulierenden Befehlen kommt dem Prozeßrechnercharakter des Mikrorechners sehr entgegen. Signalpegel sind somit relativ leicht auswertbar.

#### Fragen und Aufgaben:

1. Worin unterscheiden sich Rotier- und Verschiebepfehle?
2. Realisieren Sie ein Programm zur Multiplikation einer 16-stelligen Binärzahl mit der Konstante 5 unter Anwendung von Rotierbefehlen!
3. Für welche Elementaroperationen ist die EXCLUSIVE-ODER-Operation einsetzbar? Nennen Sie mindestens drei!
4. Worin unterscheiden sich die Operationen Negation und Komplement?
5. Realisieren Sie mit Hilfe der Monitor-Unterprogramme KONSOL und ZIFANZ eine von rechts nach links auf dem Display sich bewegende Schrift beliebigen Inhalts (Laufschrift)!

### 7. Die Anschlußtechnik des Mikrorechners

Nachdem wir uns umfangreich mit der Programmtechnik des Mikrorechners beschäftigt haben, wollen wir uns genauer als im ersten Kapitel mit den Problemen der Schaltungstechnik des Mikrorechners auseinandersetzen.

#### 7.1. Das Buskonzept

##### 7.1.1. Prinzip

Der Mikrorechner verfügt zur Übertragung von Informationen zwischen den Komponenten CPU, Speicher und Peripherie über ein Bussystem, bestehend aus Adreß-, Daten- und Steuerbus. An den Busleitungen sind sämtliche Rechnerelemente parallel (!) angeschlossen.

Der Informationsaustausch erfolgt in der Regel nur zwischen jeweils zwei Komponenten, z.B. zwischen CPU und Speicherbaustein. Die jeweils am Informationsaustausch beteiligten Elemente (Chips) werden anhand der Adresse aktiviert.

Ein spezieller Dekoder (meist ein IC) erzeugt aus den Adreß- und Steuersignalen ein Aktivierungssignal, daß als Chip-Select (CS) oder Chip-Enable-Signal (CE) bezeichnet wird. Für einen Speicherbaustein würde das CS-Signal z.B. aus folgenden Signalen erzeugt werden müssen (Bild 7.1.):

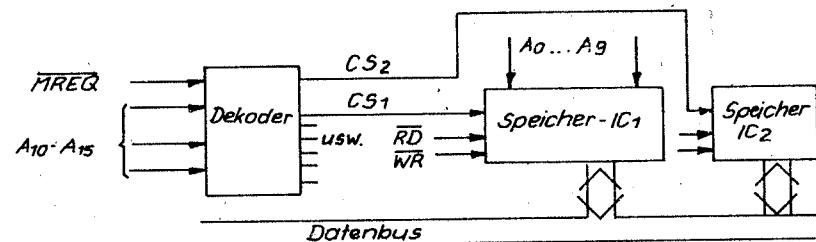


Bild 7.1. Erzeugung von Aktivierungssignalen (CS)

D.h. wenn CS<sub>1</sub> aktiv ist, darf der Speicher-IC<sub>1</sub> Daten ausgeben ( $\overline{RD}$ -aktiv) bzw. er erhält Daten ( $\overline{WR}$  aktiv).

In der übrigen Zeit darf dieser Speicher-IC die Signale auf dem Datenbus nicht beeinflussen.

Praktisch ist dieses Problem am günstigsten mit Hilfe von sogenannten "tri-state" - Ausgangsschaltungen (Dreizustands-...) zu lösen. Das Prinzip zeigt in vereinfachter Form Bild 7.2.

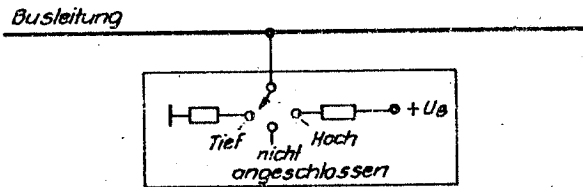


Bild 7.2. Prinzip der tri-state-Ausgänge

Neben den zwei Logikpegeln Tief und Hoch (0 und 1) existiert ein dritter Zustand, der mit einer Abschaltung der Busleitung von diesem IC vergleichbar ist (d.h. Leitung wird hochohmig sowohl gegenüber  $+U_B$  als auch gegen Masse. Diese Abschaltung wird meist durch Deaktivierung des CS-Signals vorgenommen. In diesem dritten Zustand beeinflußt der Schaltkreis die Busleitung nicht mehr.

Sämtliche Schaltkreisausgänge, die auf eine gemeinsame Busleitung arbeiten, müssen tri-state-Verhalten aufweisen!

Wenn beispielsweise an eine Busleitung Schaltkreise ohne tri-state-Verhalten angeschlossen werden sollen, so muß dies stets unter Zwischenschaltung eines tri-state-Treiber-ICs erfolgen (siehe ICs 8216 und 8212 im Systemhandbuch). Heute werden meist zur Realisierung von Busstrukturen derartige tri-state-fähige ICs eingesetzt. Die Schaltkreise der U880-Familie (CPU, PIO, SIO, CTC) sowie die meisten Speicher-ICs besitzen tri-state-Verhalten. Erwähnt werden sollte aber, daß auch durch konsequenten Einsatz von Offen-Kollektor-Ausgängen (kein Arbeitswiderstand) sowie durch andere Methoden Busstrukturen erreichbar sind.

### 7.1.2. Elektrische Bedingungen für den Busanschluß

Schaltkreisausgänge sind nur begrenzt belastbar, d.h. die Anzahl der an einen IC-Ausgang anschließbaren IC-Eingänge ist begrenzt. In den Datenblättern der ICs (z.B. im Systemhandbuch des POLY-COMPUTERS) sind unter der Rubrik "Statische Kennwerte" die maximalen Ströme, die ein Ausgang abgeben bzw. aufnehmen kann (bei Gewährleistung der Logikpegeltoleranzen) angegeben. Darüber hinaus sind vor allem bei MOS-Schaltkreisen (wie z.B. CPU U880) die maximal anschließbare Lastkapazität interessant, die wiederum das dynamische Verhalten beeinflusst.

Typische Werte sind (für U880-Familie):

Ausgangspegel:	0...0,4V	max. 1,8 mA
	2,4V	max. 250 $\mu$ A
Eingangskapazität:	5pF	
max. anschließbare Kapazität:	50pF	
Leckstrom bei tri-state:	10 $\mu$ A	

Die Anzahl derartiger ICs, die direkt parallel an einen Bus angeschlossen werden dürfen, ist somit aufgrund der Lastkapazität auf etwa 10 begrenzt.

Ein Vergleich des Stromangebots von derartigen NMOS-Ausgängen mit dem Strombedarf von TTL-Bausteinen (z.B. D100: Tief-1,6 mA, Hoch-250  $\mu$ A) liefert die ernüchternde Tatsache, daß z.B. ein CPU-Ausgang maximal einen (!) TTL-Eingang ausreichend versorgen kann. Aus diesem Grund spielen Treiberschaltkreise für die Mikrorechner-technik (z.B. 8216, siehe Systemhandbuch) eine wesentliche Rolle. Sie vergrößern das Stromangebot der Ausgänge z.T. erheblich. Im POLY-COMPUTER (siehe Bedienhandbuch) wurden TTL-ICs als Bustreiber eingesetzt, die z.B. den Strom für die Leuchtdioden des Busanalysators liefern (Bild 7.3.).

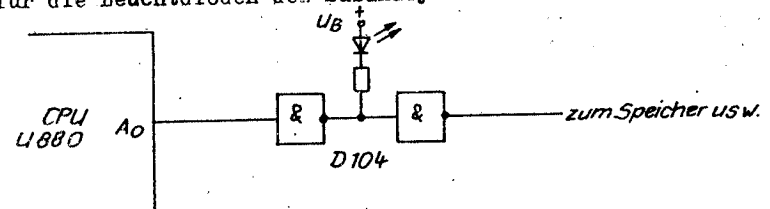


Bild 7.3. Bustreiber im POLY-COMPUTER



### 7.1.3. Der Systemsteckverbinder des POLY-COMPUTERS

Der POLY-COMPUTER ist für externe Erweiterungen des Speichers und der Peripherie vorbereitet. Dazu sind sämtliche Bussignale (Adreß-, Daten- und Steuerbus), die Betriebsspannungen sowie einige Sondersignale auf einem 58-poligen Steckverbinder an der Anschlußseite verfügbar. Die Belegung sowie zusätzlich zu beachtende Bedingungen sind im Bedienungshandbuch enthalten. Besonders zu berücksichtigen sind die maximal zulässigen Ausgangslasten (maximal eine TTL-Last) sowie die begrenzten Reserven des Netzteilens.

### 7.2. Die zeitliche Steuerung der CPU (timing)

#### 7.2.1. Das Taktdiagramm

Die Operationen der Bausteine der U880-Familie (CPU, CTC, SIC) laufen nach einem festen zeitlichen Regime ab, dessen kleinste Einheit eine Taktperiode ist. Der Rechnertakt ( $\phi$ ) wird meist von einem Quarzgenerator geliefert.

Die maximal zulässige Taktfrequenz für das U880-Bausteinsortiment beträgt 2,5 MHz. Im POLY-COMPUTER finden Anfalltypen Anwendung, die bis max. 1 MHz einsetzbar sind. Es wurde eine Taktfrequenz von 921,6 KHz gewählt (dieser etwas "krumme" Wert ergibt bei Anwendung des CTC als Teiler häufig benötigte Frequenzen und Zeitwerte). Ohne Berücksichtigung künstlich verlängerter Maschinenzyklen (durch Aktivieren von WAIT) bilden drei bzw. vier Takte einen Maschinenzyklus (Bild 7.4.). Mehrere solcher Maschinenzyklen (siehe Kapitel 2) bilden schließlich einen vollständigen Befehlszyklus.

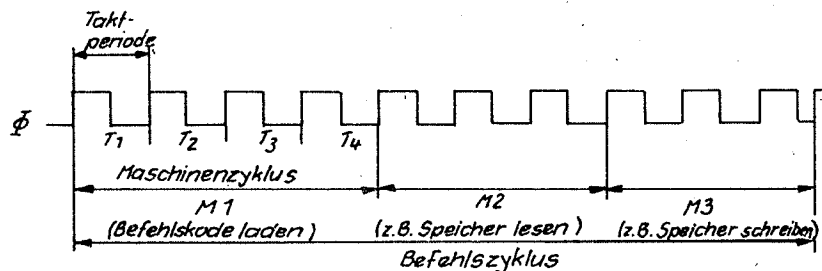


Bild 7.4. Beispiel für CPU-Zeitablauf

An dieser Stelle sollen die drei wesentlichen Zyklustypen nochmals anhand ihres Taktdiagramms erläutert werden.

### M1 - Zyklus (Befehlskode laden)

Der M1-Zyklus wird ein- bzw. zweimal zu Beginn jedes Befehls ausgeführt. Er besitzt aufgrund seiner Doppelfunktion (Befehlskode laden und Auffrischen dynamischer Speicher) stets mindestens vier Taktperioden (Bild 7.5.).

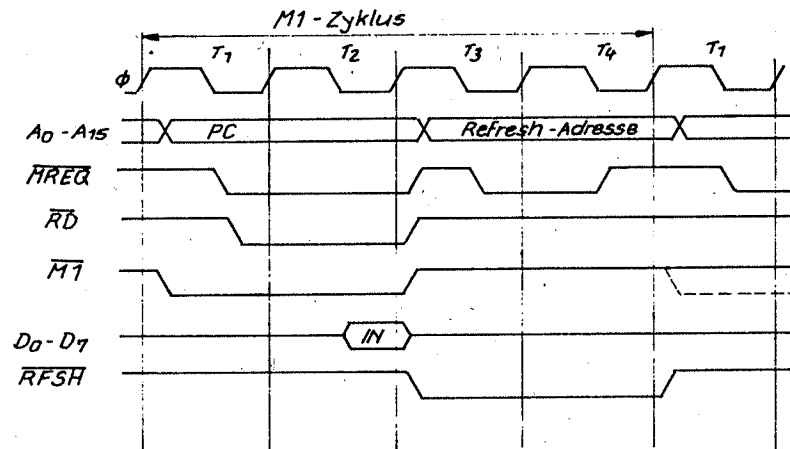


Bild 7.5. Typischer M1-Zyklus (Befehlskode laden)

Zu Beginn von  $T_1$  werden bereits der Programmzählerinhalt (PC) auf dem Adreßbus ( $A_0 - A_{15}$ ) sowie das Signal  $\bar{M1}$  ausgegeben. In der zweiten Hälfte von  $T_1$  folgen die Signale  $\bar{MREQ}$  und  $\bar{RD}$ , die zur Aktivierung des adressierten Speichers eingesetzt werden können, der den gesuchten Befehlskode enthält ( $\bar{MREQ}$ -Speicheranforderung,  $\bar{RD}$ -Lesen). Bis zum Ende von  $T_2$  hat dieser Speicher Zeit, das gewünschte 8Bit-Datenmuster (Befehlskode) auf dem Datenbus bereitzustellen (d.h. bei der Taktfrequenz von 921,6 KHz sind das etwa 1,5  $\mu s$ ). Danach wird das auf dem Datenbus befindliche Byte gelesen und als Befehl interpretiert, gleichgültig, ob es das richtige ist oder nicht. Sollte der angeschlossene Speicher länger für die Bereitstellung benötigen, so können mit Hilfe der WAIT-Leitung zusätzliche Taktperioden eingefügt werden (siehe E/A-Zyklus). Der auf dem Busanalysator des POLY-COMPUTERS im Einzelzyklusbetrieb ablesbare Zustand entspricht dem am Ende von  $T_2$  jedes Zyklus. Mit Beginn von  $T_3$  ist der eigentliche Befehlscodevorgang abgeschlossen.

In  $T_3$  und  $T_4$  läuft ein weiterer Speicherlesevorgang, der lediglich beim Einsatz sogenannter dynamischer RAM-Bausteine Anwendung findet.

Als dynamische RAM (DRAM) werden Halbleiterspeicherbausteine bezeichnet, die elektrische Ladungen zur Informationsspeicherung verwenden. Durch sogenannte Leckströme sind diese Ladungen relativ kurzlebig, so daß der Informationsgehalt in kurzen Abständen regeneriert werden muß (ca. aller 2 ms). Diese Regenerierung erfolgt durch Lesen der Information.

Dazu wird während  $T_3$  und  $T_4$  jedes M1-Zyklus der Inhalt des Refresh-Registers (R) der CPU (Kapitel 4) auf den Adreßbus ausgegeben (Bits  $A_0 - A_6$ ) und die Steuersignale  $\overline{MREQ}$  und  $\overline{RFSH}$  aktiviert. Dieses R-Register wird zyklisch weitergezählt, so daß bis zu einer bestimmten DRAM-Größe die Regenerierung sämtlicher DRAM-Zellen innerhalb von höchstens jeweils 2 ms abgeschlossen ist.

Speicherzugriffszyklus (Lesen / Schreiben)

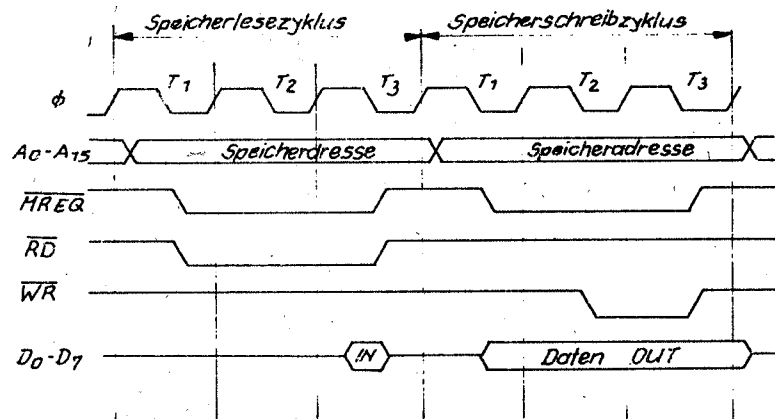


Bild 7.6. Typische Speicherzugriffszyklen (Lesen/Schreiben)

Beim Speicherlesezyklus steht kurz nach Beginn von  $T_1$  bis nach  $T_3$  die Speicheradresse zur Verfügung (auf Adreßbus). Ab der zweiten Hälfte von  $T_1$  sind die Steuersignale  $\overline{MREQ}$  und  $\overline{RD}$  aktiv. Diese Signale ( $A_0 - A_{15}$ ,  $\overline{MREQ}$ ,  $\overline{RD}$ ) können zur Auswahl des Speicherplatzes verwendet werden. Vor der fallenden Flanke von  $\phi$  in  $T_3$  müssen die Daten vom Speicher stabil auf dem Datenbus anliegen.

Der Speicherlesezyklus verläuft analog, nur daß diesmal das Steuersignal  $\overline{WR}$  (Schreiben) anstelle von  $\overline{RD}$  (Lesen) aktiviert wird. Während der gesamten Gültigkeitsphase von  $\overline{WR}$  sind die Ausgabedaten auf dem Datenbus stabil und können übernommen werden.

Ein-/Ausgabezyklus

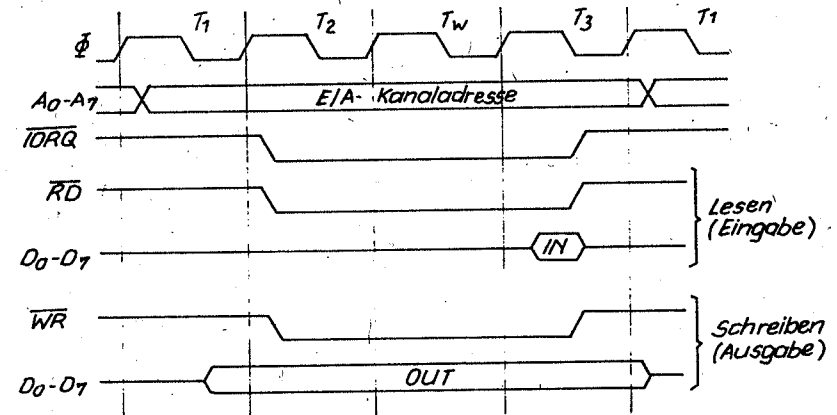


Bild 7.7. Typischer Ein-/Ausgabezyklus

Während Ein- bzw. Ausgabezyklen (z.B. von oder zu einer PIO) wird in  $T_1$  von der CPU die Kanaladresse auf die unteren Adreßbits ( $A_0 - A_7$ ) ausgegeben und liegt während des gesamten weiteren Zyklus an. Das Auswahlsignal für einen E/A-Zugriff ( $\overline{IORQ}$ ) kommt erst mit Beginn von  $T_2$  (also wesentlich später als das vergleichbare  $\overline{MREQ}$  bei Speicherzugriff), so daß bei sämtlichen E/A-Zyklen zur Realisierung einer nicht zu geringen Zugriffszeit ein zusätzlicher Zustand  $T_w$  (Wartezustand) nach  $T_2$  eingeschoben wird. Damit hat die Peripherie eine Taktperiode länger Zeit, auf CPU-Anforderungen zu reagieren. In diesem  $T_w$ -Zustand bleiben sämtliche CPU-Ausgabesignale unverändert. Eingabedaten (von der Peripherie) müssen vor der fallenden Taktflanke in  $T_3$  stabil sein. Ausgabedaten (zur Peripherie) sind bereits ab Mitte  $T_1$  gültig und können mit einer der Flanken des  $\overline{WR}$ -Signals getaktet werden. Beim Entwurf von Eingabebrettern bzw. Ausgaberegistern sind bei zeitkritischen Anwendungen auch die Laufzeiten der Dekoder usw. zu beachten!

### 7.2.2. Der Anschluß von Speicherschaltkreisen

Speicherschaltkreise sind an die Busstruktur eines Mikrorechners meist bereits angepaßt, so daß kleinere Systeme ohne spezielle Treiberschaltungen aufgebaut werden können.

Im POLY-COMPUTER werden ROM Schaltkreise des Types U505 eingesetzt. Sie besitzen eine Speicherkapazität von 1KBytes=1024 Bytes. Jedes Byte ist adressierbar, der IC besitzt daher acht Datenausgänge und 10 ( $2^{10}=1024$ ) Adreßeingänge. Ein  $\overline{CS}$  (chip-select)-Auswahleingang aktiviert den IC sowie seine Datenausgänge (tri-state). Die beiden Anschlüsse für die Betriebsspannung (+5V) vervollständigen die Pinbelegung (Pin-Anschlußbein). Zum Aufbau eines Speichers mit 8Bit=1Byte Aufraufbreite wird demnach nur ein Schaltkreis benötigt.

Den Anschluß eines solchen Speicher-ICs an eine CPU zeigt Bild 7.8.

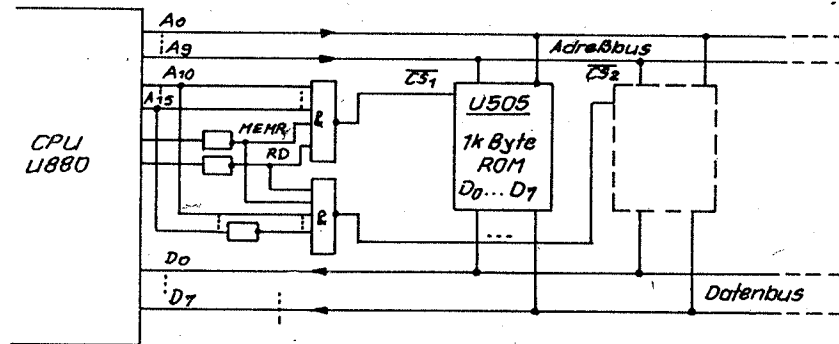


Bild 7.8. CPU-Anschluß von ROM-Schaltkreisen

Adreß- und Datenbus sind unmittelbar anschließbar. Die  $\overline{CS}$ -Leitung soll nur dann aktiviert werden, wenn

- (1) zu einem Speicher zugegriffen wird ( $\overline{MREQ}$ ),
- (2) von einem Speicher gelesen wird ( $\overline{RD}$ ),
- (3) der richtige Adreßraum angewählt wurde ( $A_{10}-A_{15}$ ).

In Bild 7.5. wird  $\overline{CS}_1$  aktiv, wenn  $\overline{MREQ}$  und  $\overline{RD}$  und  $A_{10}-A_{15}$  aktiv sind, d.h. dieser Speicher -IC belegt die Adressen  $A_{10}=...=A_{15}=1 \rightarrow FCOOH$  bis  $FFFFH$ .  $\overline{CS}_2$  dagegen wird nur aktiv, wenn  $A_{15}=0$ , alle andere aber wie bei  $\overline{CS}_1$  zutrifft, d.h.  $\rightarrow 7COOH$  bis  $7FFFH$ .

RAM-Speicher werden prinzipiell gleichartig angeschlossen, lediglich die Unterscheidung zwischen Schreiben und Lesen wird noch mittels der Signale  $\overline{RD}$  oder  $\overline{WR}$  zusätzlich getroffen.

Über die Anschlußbedingungen für Ein- und Ausgabearrichtungen (Peripherie) gibt Kapitel 8 Auskunft.

### 7.3. Zusammenfassung

Das Buskonzept des Mikrorechners schafft die Voraussetzungen für eine relativ problemarme Konfigurierbarkeit und damit die enorme Flexibilität in der Anwendung. Trotzdem erwachsen aus der Nichtbeachtung der Verträglichkeit von Signalen sowie ihrer Belastungsgrenzen nicht selten "unerklärliche" meist dynamische Fehlererscheinungen, deren Ursachendann auf anderen Gebieten gesucht werden. Eine strikte Einhaltung und Beachtung der statischen und dynamischen Kenn- und Grenzwerte der Schaltkreise (siehe Systemhandbuch) ist deshalb Voraussetzung für das zuverlässige Funktionieren mikroelektrischer Baugruppen.

#### Fragen und Aufgaben:

1. Welche Forderungen werden an Bauelemente bzw. Baugruppen gestellt, die an einen gemeinsamen Bus angeschlossen werden sollen?
2. Wie verhalten sich tri-state-Ausgänge?
3. Was sind statische, dynamische und Grenzwerte von elektronischen Bauelementen?
4. Was ist eine "TTL-Last"?
5. Unterscheiden sich die Speicherzugriffszeiten während eines M1- und eines Speicherlesezyklus?
6. Wie unterstützt die CPU U880 die Verwendung dynamischer RAM-Bausteine?
7. Wie müßte ein Dekoder aufgebaut sein, der für einen Schaltkreis U505 den Adreßraum 400H-7FFFH reserviert?

### 8. Techniken der Ein- und Ausgabe (E/A)

#### 8.1. Grundlagen der Ein-/Ausgabe-Technik

##### 8.1.1. Einführungsbeispiel E/A-Befehle

Wie aus der Rechnergrundstruktur (Abschnitt 1.1) zu erkennen ist, bilden die Ein- und Ausgabeeinrichtungen einen wichtigen Grundbestandteil jedes Rechners. Zu diesen Einrichtungen (auch Peripherie des Rechners genannt) gehören nicht nur solche datenverarbeitungstypischen Geräte wie Tastaturen, Drucker, Bildschirme, Lochbänderleser und -stanzer. Für die Mikrorechner-technik sind E/A-Einheiten zu einem auszuwertenden oder zu steuernden Prozeß typisch wie beispielsweise Lichtschranken, Leistungsschalter, Analogsensoren (z.B. Temperaturfühler) über Analog/Digital-Wandler u.a. Der Informationsaustausch zwischen dem Rechner und seiner Umwelt (der Peripherie) erfolgt in Mikrorechnersystemen meist über hochintegrierte, programmierbare Ein-/Ausgabe-Schaltkreise. Im Schaltkreissystem U880 stehen der Parallel-E/A-Schaltkreis U855, der serielle E/A-Schaltkreis U856 und der Zähler-/Zeitgeberschaltkreis U857 zur Verfügung. "Programmierbar" bedeutet, daß die Schaltkreise sehr universell sind (um die Zahl der notwendigen Typen gering zu halten) und daß sie erst durch Ausgabe von Steuerwerten von Seiten der CPU auf eine bestimmte Betriebsweise festgelegt werden. Gelegentlich werden aber auch einfachere nicht programmierbare E/A-Schaltungen verwendet.

Zum besseren Verständnis der folgenden Abschnitte und Experimente wollen wir eine sehr einfache Ein-/Ausgabe-Aufgabe betrachten: Es soll die Stellung von drei Schaltern eingelesen werden. Weiterhin soll eine Ausgabe auf zwei Lampen erfolgen in der Art, daß Lampe A leuchtet, wenn kein Schalter in Stellung EIN ist und Lampe B leuchtet, wenn alle Schalter in Stellung EIN sind (siehe Bild 8.1). Zunächst müssen wir die Schalterstellung dem Mikrorechner (dessen CPU) zugänglich zu machen. Da die Schalterstellungen vom Wesen her Daten sind, mit denen der Rechner arbeiten soll, liegt der Gedanke nahe, sie an den Datenbus anzuschließen. Ein direkter Anschluß würde jedoch die Arbeit des Rechners unmöglich machen, da der Datenbus auch andere Daten als die Schalterstellungen übertragen muß. Die Lösung dieses Problems besteht darin, daß die Schalter über eine Torschaltung an den Datenbus angeschlossen werden, die die den Schalterstellungen entsprechenden Logikpegel nur auf eine besondere Anweisung, einen Eingabebefehl, hin auf den Datenbus durchschaltet.

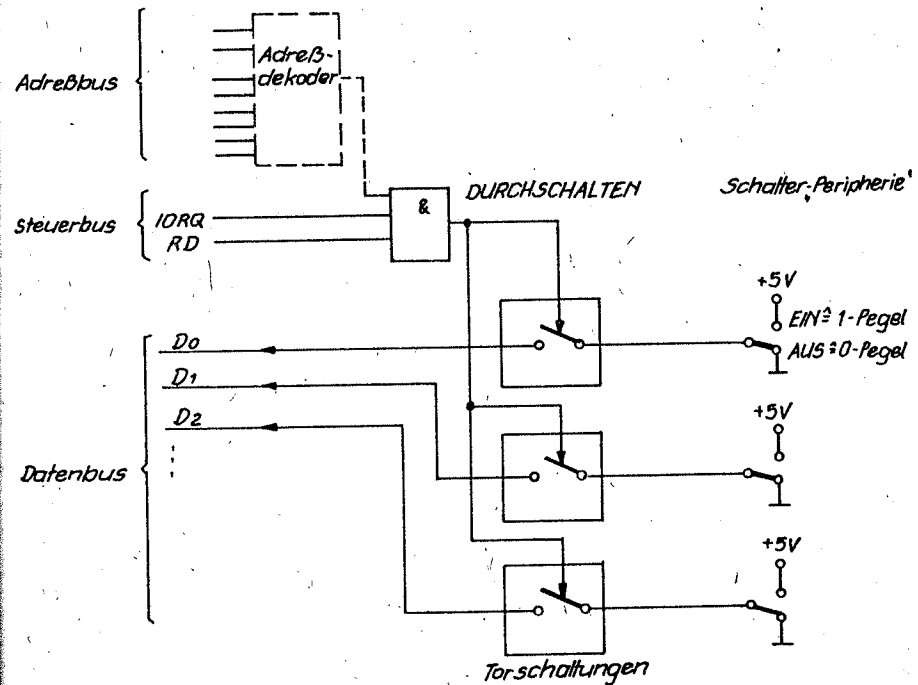


Bild 8.1. Einfache E/A-Torschaltung

Wenn der Prozessor einen Eingabebefehl ausführt, aktiviert er seine Steuersignale IORQ (E/A-Zugriff) und RD (Lesen). Diese beiden Signale werden UND-verknüpft, d.h. das Signal DURCHSCHALTEN wird aktiv, wenn die beiden genannten Steuersignale aktiv sind. Es wird damit bewirkt, daß die Schalter nur bei Ausführung eines Eingabebefehls auf den Datenbus durchgeschaltet werden.

Zu diesem einfachen Vorstellungsmodell sind einige Bemerkungen notwendig: Die Steuersignale werden vom Mikroprozessor negiert (0-aktiv) erzeugt, d.h.  $\overline{IORQ}$ ,  $\overline{RD}$ . Die betrachteten Signale IORQ und RD sind also durch Negation der entsprechenden Signale des Prozessors entstanden.

Die Torschaltungen werden natürlich nicht durch mechanische Schalter, sondern durch logische Verknüpfungen realisiert, wichtig ist dabei, daß die Schaltung im inaktiven Zustand zum Datenbus hin hochohmig ist (tri-state-Schaltung).

Mit der bisher beschriebenen Anordnung könnten wir maximal 8 Schalter, nämlich an jede Datenbusleitung einen, anschließen. Meist ist das nicht ausreichend. Eine größere Zahl von Eingabeleitungen läßt sich anschließen, wenn die Eingabeeinrichtungen adressiert werden. Bei der Ausführung eines Eingabebefehls führen die acht niederwertigen Leitungen des Adreßbusses (A0-A7) die angesprochene Eingabeadresse. Diese Adresse wird einem Dekoder zugeführt, der seinen Ausgang nur dann aktiviert, wenn eine ganz bestimmte Adresse angelegt wurde. Dieses Signal wird in die UND-Verknüpfung zur Gewinnung des Signals DURCHSCHALTEN einbezogen und bewirkt, daß die Schalterstellungen nur bei einem Eingabebefehl mit einer ganz bestimmten Adresse auf den Datenbus durchgeschaltet werden. Ein häufig verwendeter Eingabebefehl hat folgendes Aussehen:

Mnemonic:	IN	n	(Input - Eingabe) (n-Eingabeadresse)
Befehlskode:			
binär	1101	1011	
hex	← n	→	
	DB		
		n	
Wirkung:	Das Datenmuster am Eingabetor wird in das A-Register transportiert.		
Flagbeeinflussung:	keine		

Beachtenswert ist, daß die Eingabeadressen im Gegensatz zu einer Speicheradresse in nur einem Byte dargestellt werden, es gibt also  $2^8=256$  Eingabeadressen (d.h. 256 Eingabeeinheiten sind prinzipiell möglich). Das folgende kleine Experiment auf dem POLY-COMPUTER soll den Ablauf des Befehles demonstrieren:

- Wir schreiben einen Eingabebefehl in den RAM-Speicher mit Hilfe der Funktion **MEM**:

Adresse	Dateninhalt	
4000H	DBH	Operationskode
4001H	33H	Eingabeadresse 33

- Anschließend wird der Akkumulator mit einem definierten Inhalt ≠ FFH mit der Funktion **REG** geladen, z.B.

Register	Dateninhalt
AF	55

- Den Eingabebefehl lassen wir im Maschinenzklusbetrieb ausführen:

Taste **MCYCL** (Lampe MCYCL muß aufleuchten)

Eingabe: **STEP**, **4000**, **EXEC**

Wir sehen auf dem Busanalysator, daß als erstes der Operationskode des Befehls gelesen wird:

Daten: 1 1 0 1 1 0 1 1 B ≙ DBH von Adresse 4000H;  $\overline{M1}$ ,  $\overline{MREQ}$ ,  $\overline{RD}$  sind aktiv.

Nach Betätigen der Taste **CYCL** wird die Eingabeadresse gelesen:

Daten: 0 0 1 1 0 0 1 1 B ≙ 33H von Adresse 4001H;  $\overline{MREQ}$ ,  $\overline{RD}$  sind aktiv.

Nach einer weiteren Betätigung der Taste **CYCL** wird der eigentliche Eingabezyklus ausgeführt; erkennbar daran, daß Steuersignale  $\overline{IORQ}$  und  $\overline{RD}$  aktiv sind. Die Adreßleitungen A7 bis A0 führen die Eingabeadresse. Die Datenleitungen führen die Belegung 1 1 1 1 1 1 1 1 B ≙ FFH (unter der angegebenen Adresse ist nichts angeschlossen, die Datenbustreiberschaltung bewirkt, daß alle Datenleitungen 1-Pegel führen). Aufmerksamen Beobachtern wird nicht entgehen, daß die höherwertigen Adreßleitungen A15 - A8 ebenfalls eine ganz bestimmte Belegung haben, nämlich den von uns eingegebenen Akkumulatorinhalt 0 1 0 1 0 1 0 1 B ≙ 55H. Dies ist eine spezielle Eigenschaft des von uns verwendeten Eingabebefehls, die nur für wenige Spezialanwendungen von Bedeutung ist.

Als nächstes kehren wir mit der Taste **MON** in das Monitorprogramm zurück, wir können uns nun mit der Funktion **REG** von der Ausführung der Operation überzeugen. Wir stellen fest, daß das Akkumulatorregister erwartungsgemäß nicht mehr mit dem alten Wert, sondern mit dem eingelesenen Wert FFH geladen ist.

Wir wollen uns nun der Ausgabe zuwenden. Wir betrachten wiederum zuerst den Fall, daß die Lampen (über eine entsprechende Treiberschaltung) direkt an den Datenbus angeschlossen werden. Genau dies ist der Fall bei der Datenbusanzeige (Busanalysator) im POLY-COMPUTER. Wir können feststellen, daß sich im normalen Rechenbetrieb ein mehr oder weniger gleichmäßiges Leuchten ergibt, da auf dem Datenbus in schneller Folge verschiedene Daten vorliegen. Eine Torschaltung wie bei der Eingabe würde zwar bewirken, daß nur ganz bestimmte Daten zu unseren Lampen gelangen, die richtige Datenbelegung wäre aber nur für den kurzen Moment des Ausgabezyklus vorhanden.

Wir benötigen eine Speicherung der ausgegebenen Informationen. Diese kann mit einem Flip-Flop erfolgen, dessen Funktionsweise wir im Abschnitt 1.2.1. kennengelernt haben.

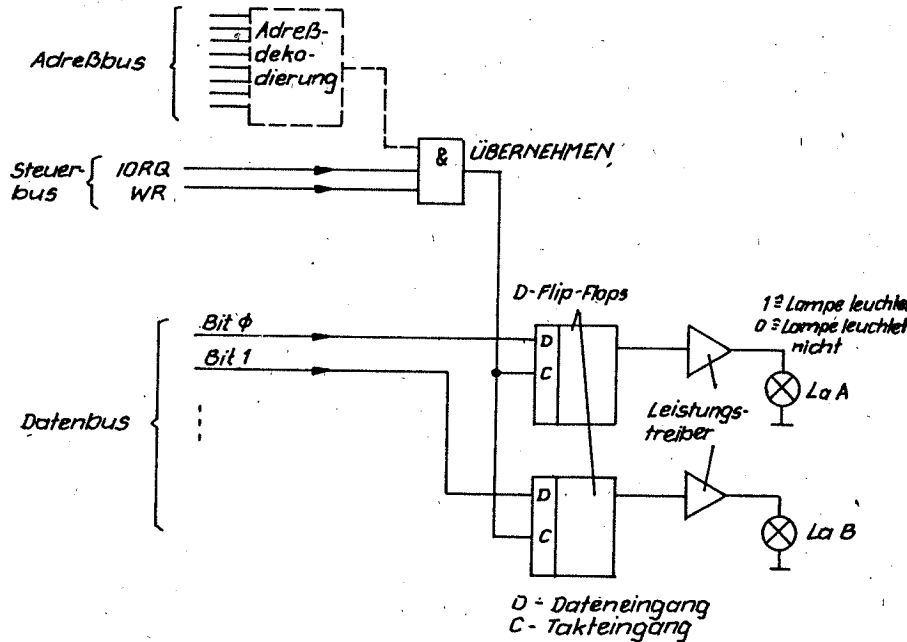


Bild 8.2. Einfaches Ausgaberegister

Wir benutzen sogenannte D-Flip-Flops (D für delay = Verzögerung), die durch folgende Eigenschaften charakterisiert werden:  
 Wenn der Takt C aktiv wird (1-Pegel), wird die am D-Eingang anliegende Information übernommen und erscheint am Ausgang. Wenn der Takt C wieder inaktiv wird, bleibt die übernommene Information am Ausgang bestehen, die am D-Eingang anliegende Belegung hat keinen Einfluß mehr auf die gespeicherte Information und den Ausgang. Die dargestellte Ausgabereinrichtung (Bild 8.2.) funktioniert nun so, daß die D-Eingänge der Flip-Flops am Datenbus des Rechners angeschlossen sind. An die Takteingänge ist das Signal ÜBERNEHMEN angeschlossen. Dieses Signal wird nur durch einen Ausgabebefehl des Prozessors aktiviert. Bei der Ausführung eines Ausgabebefehls werden die Steuersignale  $\overline{IORQ}$  (Ein-/Ausgabezugriff) und  $\overline{WR}$  (Schreiben) aktiviert, die UND-Verknüpfung dieser Signale erzeugt unser Signal ÜBERNEHMEN.

Auf diese Weise wird erreicht, daß die Lampen entsprechend der bei einem Ausgabebefehl anliegenden Daten leuchten oder nicht. Der Zustand der Lampen bleibt wegen der Speicherwirkung der Flip-Flops bis zum nächsten Ausgabebefehl erhalten! Um mehr als 8 binäre Ausgabegeräte (wie z.B. Lampen) anschließen zu können, ist wieder eine Adressierung notwendig, d.h. bei jedem Ausgabebefehl wird eine bestimmte Belegung der niederwertigen Adreßbits A7 - A0, die Ausgabeadresse, erzeugt. Diese kann in die Verknüpfung zur Erzeugung des Signals DURCHSCHALTEN einbezogen werden und so ermöglichen, daß bei einem Ausgabebefehl nur genau eine von mehreren Ausgabereinrichtungen angesprochen wird. Der gebräuchlichste Ausgabebefehl lautet:

Mnemonic:	OUT n	(output - Ausgabe)
Befehlscode:		(n-Ausgabeadresse)
binär	1 1 0 1 0 0 1 1	
hex	← n →	
	D3	
	n	
Wirkung:	Der A-Registerinhalt wird zur Ausgabereinrichtung mit der Adresse n transportiert.	
Flagbeeinflussung:	keine.	

Ermitteln Sie, den genauen Ablauf des Befehls mit Ihrem POLY-COMPUTER 880 wie für die Eingabe beschrieben! Wählen Sie dabei eine Ausgabeadresse < 80H, da die Ausgabeadressen ≥ 80H zum Teil vom MONITOR benutzt werden und sie durch Ausgaben eventuell dessen Funktion stören könnten!

Die Ein-/Ausgabereinrichtungen (Ein-/Ausgabegeräte) werden auch als Kanäle oder Ports bezeichnet, die entsprechenden Adressen als Kanal- oder Portadressen.

Kommen wir nun wieder zu der eingangs genannten Anwendung zurück, für die wir nun alle Voraussetzungen kennengelernt haben. Der Programmablaufplan ist in Bild 8.3. dargestellt.

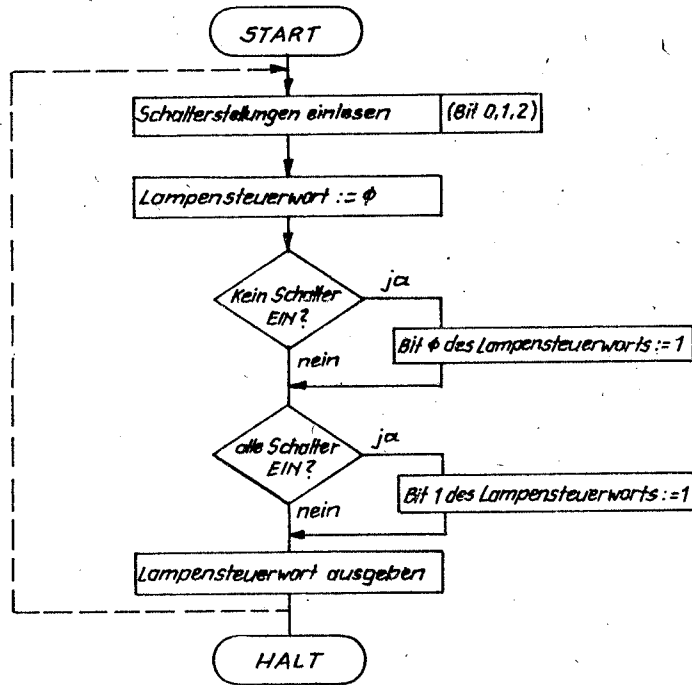


Bild 8.3. Programmablaufplan zum E/A-Beispiel

Wir müssen eine rechnerinterne Darstellung der zu den Lampen auszugebenden Daten einführen, die wir Lampensteuerwort nennen. Damit eine ständige Aktualisierung des Lampenzustandes entsprechend der Schalterstellung erfolgt, darf das Programm nicht bei **HALT** enden, sondern muß der unterbrochenen Linie folgen und zyklisch ablaufen. Wir können uns an diesem einfachen Beispiel eine sehr wichtige Eigenschaft jeder programmgesteuerten Verarbeitung veranschaulichen:

Die ausgegebenen Daten (Lampenleuchtzustand) folgen erst mit einer gewissen Verzögerung (der Laufzeit des Verarbeitungsprogramms) den Eingabedaten (Schalterstellungen). Außerdem muß eine bestimmte Eingabedatenbelegung mindestens für die Laufzeit eines Programmzyklus anliegen, damit sie auch wirklich verarbeitet wird und die Ausgabedaten beeinflußt.

Wir stellen nun das Rechnerprogramm auf, das den angegebenen Programmablauf realisiert; dabei nehmen wir an, daß die Ein- und Ausgabeadressen 0 und 1 sind.

Mnemonic	Kommentar
EATEST: IN 00	;Schalterstellungen in Bit 0,1,2 von A
AND 07	;Bits 3-7 von A mit 0 belegen (nicht benutzt)
LD B,00	;Lampensteuerwort in B
CMP 00	;Übereinstimmung, wenn <u>kein</u> Schalter EIN
JPNZ EA1	
SET 0,B	;Bit 0 des Lampensteuerwortes 1 setzen
EA1: CMP 07	;Übereinstimmung, wenn alle Schalter EIN
JPNZ EA2	
SET 1,B	; Bit 1 des Lampensteuerwortes 1 setzen
EA 2: LD A,B	
OUT 01	;Lampensteuerwort ausgeben
JMP EATEST	;Programm zyklisch durchlaufen

Da das Programm EATEST aufgrund der am Grundgerät nicht vorhandenen Peripherie (Lampen, Schalter usw.) nicht zur Ausführung gelangen kann, wurde auf eine Befehlskodierung verzichtet.

8.1.2. Weitere E/A-Befehle der CPU U880

Die CPU U880 hat eine Reihe weiterer Ein-/Ausgabebefehle, die in manchen Fällen günstigere Programmlösungen erlauben.

Mnemonic: IN r (r - Register A,B,C,D,E,H,L)

Befehlscode:binär 11 101 101  
01 -r- 000

hex

IN A	IN B	IN C	IN D	IN E	IN H	IN L	INF
ED 78	ED 40	ED 48	ED 50	ED 58	ED 60	ED 68	ED 70

Wirkung: Das Datenmuster von dem durch Register C adressierten Eingabetor wird in das im Befehl ausgewählte Register geladen: r:=(c) ; der Befehl INF stellt nur die Flag

Flagbeeinflussung:

CY	Z	P/V	S	N	H
•	↑	P	↑	0	↑

Mnemonic: OUT r

Befehlscode:binär 11 101 101  
-r-

hex

OUT A	OUT B	OUT C	OUT D	OUT E	OUT H	OUT L
ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

Wirkung: Die Daten aus dem im Befehl angegebenen Register werden dem durch das Register C adressierten Kanal ausgegeben: (c) :=r

Flagbeeinflussung: keine

Die Besonderheit dieser beiden Befehle besteht darin, daß die Ausgabeadresse nicht als Konstante im Programm auftritt, sondern der Inhalt eines Registers ist. Die acht höherwertigen Adreßbits führen bei diesen Befehlen den Inhalt des Registers B.

Weitere E/A-Befehle gestatten die Ein-/Ausgabe von ganzen Datenblöcken mit einem Befehl, sie werden im Kapitel 10 kurz erläutert.

8.2. CPU-gesteuerte Ein-/Ausgabe durch Abfrage

Bei dem im vorigen Abschnitt betrachteten Beispiel waren die Zeitpunkte, zu denen eine Eingabe bzw. eine Ausgabe erfolgt, von der Peripherie her betrachtet ziemlich willkürlich und ohne Bezug zu den Ereignissen in der Peripherie.

Oft ist eine derartige Vorgehensweise nicht möglich, es muß ein bestimmter Zeitbezug, auch als Synchronisation bezeichnet, zwischen der Arbeit des Rechners und der Peripherie gesichert werden.

Um dies zu veranschaulichen, betrachten wir wieder ein einfaches Beispiel:

Ein Rechnerprogramm rechnet gewisse Werte aus, die auf einem angeschlossenen Drucker ausgedruckt werden sollen. Es kann nun der Fall eintreten, daß die Ergebnisse schneller anfallen, als sie der (mechanische) Drucker drucken kann. Es muß daher ein Rückmeldungssignal vom Drucker zum Rechner eingeführt werden, das die Bereitschaft des Druckers zur Übernahme eines neuen Zeichens anzeigt. Eine einfache Art der Berücksichtigung dieses Rückmeldungssignals besteht darin, daß es vom Programm vor jeder beabsichtigten Ausgabe abgefragt wird.

Falls die Bereitschaft der Peripherie nicht gegeben ist (z.B. das vorhergehende Zeichen ist noch nicht vollständig verarbeitet), so wird dieses Bereitschaftssignal wiederholt abgefragt, bis sich die Peripherie bereit meldet.

Die CPU arbeitet also eine Programmschleife ab, die im wesentlichen aus einer Abfrage des Bereitschaftssignals der Peripherie besteht (Bild 8.4.).

Das eben vorgestellte Prinzip läßt sich sehr universell anwenden und ist auch oftmals die zweckmäßigste Lösung.

Es ist auch auf Eingabeoperationen anwendbar; bei einer Tastatureingabe würde die Bereitmeldung beispielsweise dann eintreten, wenn eine Taste gedrückt wurde, der Tastenkode steht dann zur Verarbeitung zur Verfügung.



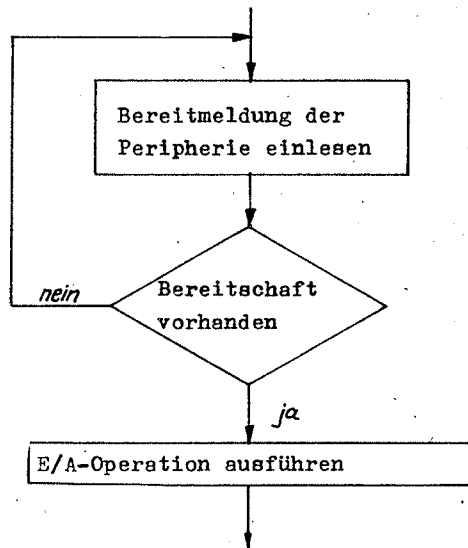


Bild 8.4. Programmablaufplan E/A mit Abfrage durch CPU

### 8.3. Ein-/Ausgabesynchronisierung durch Verlängerung der E/A-Zyklen

Neben der beschriebenen programmierten Auswertung von Bereitschaftsignalen der Peripherie gibt es noch eine weitere Möglichkeit: Der Prozessor führt den E/A-Befehl ohne Rücksicht auf den Bereitschaftszustand der Peripherie aus. Während des E/A-Zyklus wird jedoch die Leitung WAIT am Prozessor aktiviert (auf 0-Pegel geschaltet), wenn die Peripherie noch nicht bereit ist. Der E/A-Zyklus wird dadurch so lange ausgedehnt, bis die Bereitschaft der Peripherie vorhanden ist, die eine Deaktivierung von WAIT bewirken muß. Die programmierte Abfrage der Bereitschaft wird also ersetzt durch die Erzeugung eines WAIT-Signals aus der Bereitmeldung (durch Schaltungen - "Hardware"). Diese Methode scheint sehr einfach zu sein, sie hat jedoch einige Nachteile: Es kann nur ein peripheres Gerät bedient werden, während der Wartezeit (WAIT=0) werden vom Prozessor keine Verarbeitungen ausgeführt, auch auf Unterbrechungen erfolgt keine Reaktion.

Zum anderen werden die Vorteile der flexiblen programmierten Logik ungenügend ausgenutzt, da statt des Abfrageprogramms (einige Byte in einem ROM) eine spezielle, mehr oder weniger aufwendige, fest verdrahtete Logik zur Gewinnung des WAIT-Signals benötigt wird. Aus diesen Gründen wird diese Methode meist nur angewendet, wenn eine hohe E/A-Übertragungsgeschwindigkeit (E/A-Operationen je Zeiteinheit) notwendig ist, die anders nicht erreichbar ist.

### 8.4. Unterbrechungsgesteuerte Ein-/Ausgabe

#### 8.4.1. Das Unterbrechungsprinzip

Das im Abschnitt 8.2. vorgestellte Prinzip der CPU-gesteuerten Ein-/Ausgabe durch Abfrage hat in einigen Anwendungen wesentliche Nachteile: Während des Wartens auf ein Signal der Peripherie führt der Prozessor keine Rechenarbeit aus und während des Ablaufs von Verarbeitungsprogrammen erfolgt keine Reaktion auf Anforderungen von der Peripherie. Besonders der **letztgenannte** Punkt ist dann wesentlich, wenn der Rechner sehr kurzfristig auf Anforderungen seiner Umwelt reagieren und außerdem umfangreiche Verarbeitungsprogramme ausführen muß. Dieses Problem ist durch Einführung des Unterbrechungsprinzips lösbar. Vom Peripheriegerät kann durch Aktivierung eines besonderen Signals am Mikroprozessor veranlaßt werden, daß der Prozessor die Abarbeitung des gerade laufenden Programms unterbricht und ein Programm beginnt, das das Peripheriegerät behandelt. Wenn das erfolgt ist, kann die Abarbeitung des unterbrochenen Programms fortgesetzt werden.

Am Mikroprozessor U880, wie er im POLY-COMPUTER eingesetzt ist, gibt es zwei Anschlüsse für Unterbrechungssignale; für nicht-maskierbare Unterbrechungen (NMI) und für maskierbare Unterbrechungen (INT). In der Literatur wird für Unterbrechung auch häufig der Begriff Interrupt verwendet.

#### 8.4.2. Nichtmaskierbare Unterbrechung - NMI

(Non-Maskable Interrupt)

Das Signal NMI ist  $\emptyset$ -aktiv, es liegt also im Ruhezustand auf 1-Pegel. Wenn es auf  $\emptyset$  gesetzt wird, führt der Mikroprozessor den gerade bearbeiteten Befehl noch fertig aus und führt anschließend einen Sprung zur Adresse 66H aus. Dort muß das Unterbrechungsbehandlungsprogramm beginnen. Vorher erfolgt noch die Abspeicherung des (alten) Programmzählers (PC) auf den Stapelspeicher (Stack).

Der Prozessor verhält sich also ähnlich wie bei Ausführung des Befehls CALL 66H, lediglich der genaue zeitliche Ablauf ist etwas anders:

Der Prozessor führt einen NMI-Annahmezyklus aus, der zeitlich wie ein normaler M1-Zyklus (siehe Kapitel 7) abläuft, der Datenbus, inhalt wird aber vom Prozessor nicht verwendet. Anschließend erfolgen zwei Speicherschreibzyklen zur Abspeicherung des höher- und niederwertigen Bytes des alten Programmzählers (PC) unter der durch den Stapelzeiger vorgegebenen Adresse (wie beim Unterprogrammaufruf). Als nächstes führt der Prozessor einen M1-Zyklus mit der Adresse 66H aus und liest damit den ersten Befehl des Unterbrechungsbehandlungsprogramms. Diese Unterbrechungsart wird nichtmaskierbar genannt, da sie nicht durch Programmbefehle zu unterdrücken ist, d.h. in jedem Fall ausgeführt wird.

Am POLY-COMPUTER ist eine Taste MON vorhanden, mit der das Signal NMI am Prozessor aktiviert werden kann. Daraufhin führt der Prozessor einen Sprung zur Adresse 66H aus und dort beginnt das bereits wohlbekanntes Monitorprogramm des Rechners. Dieses stellt also eine Unterbrechungsbehandlung dar, wobei die "Unterbrechungsursache" die Betätigung der Taste MON ist. Wie an der Anzeige NMI zu sehen ist, wird die Leitung auch nach dem Loslassen der Taste MON im aktiven Zustand gehalten, die Anzeige NMI hat deshalb auch die Bedeutung "Monitorprogramm läuft".

Für die Auslösung der Unterbrechung ist dies aber nicht erforderlich, es genügt ein ┘-Impuls mit bestimmter Mindestlänge am NMI-Eingang der CPU. Die NMI-Annahme kann mit dem POLY-COMPUTER nicht im Maschinenzklusbetrieb ausgeführt werden, da das Betätigen der Taste MON die Nebenfunktion der Beendigung des Maschinenzklusbetriebes hat.

### 8.4.3. Maskierbare Unterbrechung INT (Interrupt)

Das Signal INT ist 0-aktiv, es liegt im Ruhezustand auf 1-Pegel. Wenn 0-Pegel angelegt wird, führt der Prozessor einen Unterbrechungsannahmezyklus aus. Seine konkreten Aktivitäten hängen dabei von der eingestellten Unterbrechungsbetriebsart ab (nächster Abschnitt). Wie das Attribut maskierbar ausdrückt, ist diese Art der Unterbrechung erlaubbar und sperbar.

Zur Beeinflussung des Unterbrechungserlaubniszustands gibt es die Befehle:

	Mnemonic	hex. Kodierung
Unterbrechung erlauben <u>Enable Interrupts</u>	EI	FB
Unterbrechung sperren <u>Disable Interrupts</u>	DI	F3

Der jeweilige Zustand wird vom Prozessor in einem sogenannten Unterbrechungserlaubnis-Flip-Flop (IFF) gespeichert. Praktisch sind im Prozessor U880 sogar zwei Flip-Flops (IFF1 und IFF2) vorhanden. Das ist notwendig, um den Unterbrechungserlaubniszustand nach NMI-Behandlungen wieder richtig herstellen zu können. IFF1 und IFF2 werden normalerweise gemeinsam gestellt, wobei IFF1 als augenblicklicher Erlaubniszustand ausgewertet wird. Bei Ausführung einer NMI-Behandlung wird IFF1 auf 0 gesetzt, um maskierbare Unterbrechungen auf jeden Fall zu sperren. Am Ende der NMI-Behandlung soll der ursprüngliche Erlaubniszustand (der in IFF2 noch gespeichert ist) wieder hergestellt werden. Dazu dient der Befehl:

	Mnemonic	hex. Kodierung
Rückkehr von <u>NMI</u> -Behandlung ( <u>Return from NMI</u> )	RETN	ED 45

Dieser Befehl bewirkt neben dem Holen des Programmzählerstandes des unterbrochenen Programms aus dem Stapelspeicher (wie bei RET) ein Laden von IFF1 mit der Belegung von IFF2, womit der eingangs genannte Zweck erreicht wird. Zur Rückkehr aus INT-Behandlungen gibt es den speziellen Befehl

	Mnemonic	hex. Kodierung
Rückkehr von Unterbrechung ( <u>Return from Interrupt</u> )	RETI	ED 4D

dessen Wirkung wie die des RET-Befehls ist, der Unterbrechungserlaubniszustand wird nicht verändert. Der Zweck dieses Befehls wird in Abschnitt 8.8. erläutert.

Für die praktische Arbeit ist es wichtig zu wissen, daß Unterbrechungen nach dem Rücksetzen und nach Annahme einer Unterbrechung gesperrt sind.

Bevor Unterbrechungsbehandlungen arbeiten, muß also der Befehl EI ausgeführt worden sein, ebenso spätestens am Ende jeder Unterbrechungsbehandlung, um weitere Unterbrechungen zu zulassen. Die folgende Tabelle zeigt noch einmal in der Zusammenstellung alle auf IPF1, IPF2 einwirkenden Ereignisse:

Aktion	IPF1	IPF2
CPU-Reset	0	0
DI	0	0
EI	1	1
Annahme von INT	0	0
Annahme von NMI	0	.
RETN	IPF2	.

IPF2 → IPF1

. bedeutet: keine Veränderung

#### 8.4.4. Unterbrechungsbetriebsarten und deren Abläufe

Die Reaktion auf eine maskierbare Unterbrechung (INT) hängt von der Unterbrechungsbetriebsart ab. Diese ist durch folgende Befehle einstellbar:

	Mnemonic	hex. Kodierung
Unterbrechungsbetriebsart 0	IM0	ED 46
Unterbrechungsbetriebsart 1	IM1	ED 56
Unterbrechungsbetriebsart 2	IM2	ED 5E

Nach dem Rücksetzen des Prozessors (Einschalten des Gerätes bzw. Taste RESET wurde betätigt) ist die Unterbrechungsbetriebsart 0 (IM 0) voreingestellt.

In jeder Unterbrechungsbetriebsart führt der Prozessor einen Unterbrechungsannahmezyklus aus, der sich von allen anderen Zyklustypen dadurch unterscheidet, daß die Steuersignale M1 und IORQ aktiv sind.

Dieser spezielle Zyklus hat die Aufgabe, dem Unterbrechungsverursacher die Anerkennung der Unterbrechung zu signalisieren (das Unterbrechungssignal kann wieder inaktiv werden); außerdem kann die Unterbrechung verursachende Schaltung ein bestimmtes Datenmuster auf den Datenbus legen, anhand dessen sie identifiziert (von anderen Unterbrechungsquellen unterschieden) werden kann. Die Unterbrechungsbetriebsarten unterscheiden sich nun in der Interpretation der bei der Unterbrechungsannahme gelesenen Daten.

#### Unterbrechungsbetriebsart 0 (IM 0):

Das Bitmuster auf dem Datenbus wird wie ein Maschinenbefehl aus dem Speicher abgearbeitet. Damit zu einem Unterbrechungsbehandlungsprogramm verzweigt wird, benutzt man meist einen Kurzuruf-Befehl (RST). Dies ist ein Einbytebefehl, der einen Unterprogrammaufruf zu einer von acht festen Adressen (0, 8, 10H, 18H, 20H, 28H, 30H, 38H) ausführt. Der alte Programmzähler wird dabei im Stapelspeicher abgelegt (wie bei CALL).

#### Unterbrechungsbetriebsart 1 (IM 1)

Das Bitmuster auf dem Datenbus hat keine Bedeutung für den weiteren Ablauf. Es folgt ein Abspeichern des Programmzählers im Kellerspeicher und ein Sprung zu der festen Adresse 38H; Die Aktivitäten sind also wie bei Ausführung des Befehls RST 38H. Diese Unterbrechungsbetriebsart ist vorteilhaft anwendbar bei einfachen Systemen mit nur einer Unterbrechungsquelle. Die Anwendung dieser Betriebsart wird im POLY-COMPUTER dadurch unterstützt, daß auf der Adresse 38H der Befehl JP 4000 steht und der Prozessor damit nach Annahme einer Unterbrechung (in IM 1) an den Anfang des RAM-Bereiches springt, wo sich ein Unterbrechungsbehandlungsprogramm des Anwenders befinden kann.

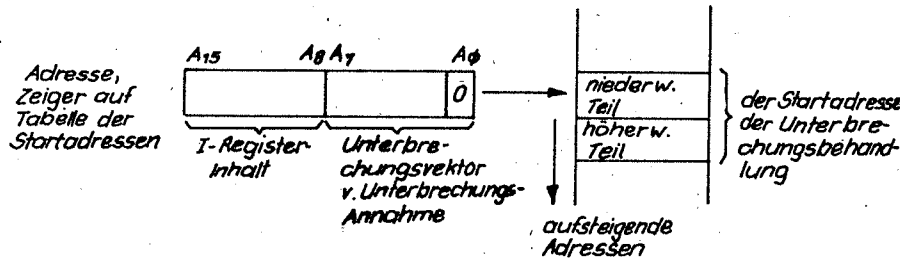
#### Unterbrechungsbetriebsart 2 (IM 2)

Dies ist die leistungsfähigste Betriebsart, da sie die Unterscheidung von praktisch beliebig vielen Unterbrechungsquellen erlaubt und Unterbrechungsbehandlungsprogramme auf beliebigen Adressen angesprungen werden können. Der Datenbusinhalt während des Unterbrechungsannahmezyklus wird als Unterbrechungsvektor bezeichnet und bildet die niederwertigen 8 Bits einer Adresse.

Die höherwertigen 8 Bit dieser Adresse werden vom I-Register des Prozessors geliefert.

Die so erhaltene Adresse bildet den "Zeiger" zu einer Tabelle, die die Startadressen der Unterbrechungsbehandlungsprogramme enthält. Die erhaltene Adresse beinhaltet den niederwertigen Teil, die nächst höhere Adresse den höherwertigen Teil der Startadresse. Das niederwertigste Bit des Unterbrechungsvektors muß 0 sein, die Startadrentabelle also auf einer geraden Adresse beginnen.

Diese doch nicht ganz einfachen Zusammenhänge werden dem Leser bei Experimenten in den folgenden Abschnitten ausführlich erläutert.



### 8.5. Direkter Speicherzugriff

Neben den bereits beschriebenen Methoden der Ein-/Ausgabe mit Abfragen und mit Unterbrechungen soll hier eine weitere kurz vermittelt werden.

Der Grundgedanke des direkten Speicherzugriffs (Kurzbezeichnung DMA: Direct Memory Access) besteht darin, daß die CPU an der E/A-Operation nicht mehr beteiligt ist. Mit Hilfe einer zusätzlichen Schaltung (DMA-Steuerschaltung) werden Daten direkt vom Speicher zur E/A-Einheit oder umgekehrt transportiert.

Wenn nur ein Bussystem vorhanden ist, kann nur entweder die CPU oder die DMA-Steuerschaltung arbeiten, die jeweils nicht aktive Einheit bringt sämtliche Busleitungen in einen hochohmigen Zustand. Bei der von uns benutzten CPU U880 kann der hochohmige Zustand der Adreß- und Datenleitungen und der Steuersignale MREQ, IORQ, RD, WR durch 0-Pegel am Eingang BUSRQ (Busanforderung - Bus Request) herbeigeführt werden. Die erfolgte Freigabe der Busse wird von der CPU durch 0-Pegel am Ausgang BUSAK (Anerkennung der Busanforderung - Bus Request Acknowledge) zurückgemeldet. Der direkte Speicherzugriff wird meist benutzt, wenn Daten in sehr schneller Folge anfallen und die anderen Methoden nicht mehr anwendbar sind (Bild 8.5.).

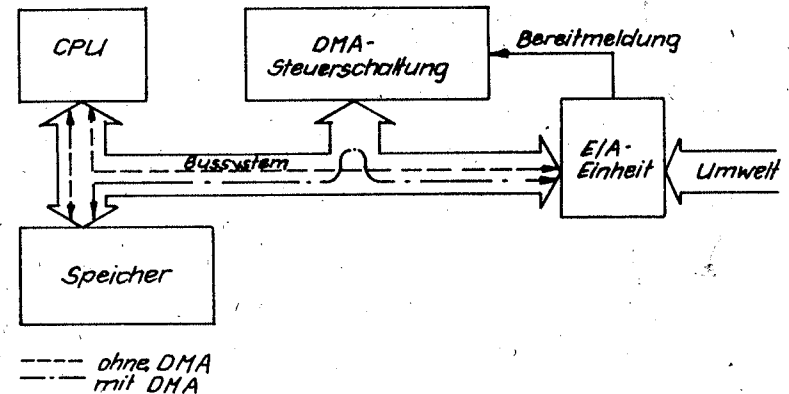


Bild 8.5. Prinzip des direkten Speicherzugriffs (DMA)

### 8.6. Der Parallel-Ein-/Ausgabeschaltkreis U855 (PIO)

#### 8.6.1. Überblick

Der Schaltkreis U855 (PIO) besitzt wie jeder Ein-/Ausgabeschaltkreis zwei Gruppen von Anschlüssen (zwei Schnittstellen). Das sind auf der einen Seite die Anschlüsse zum Bussystem des Rechners, auf der anderen Seite die zur angeschlossenen Peripherie. Der Schaltkreis hat zwei Kanäle (Ports), bezeichnet mit A und B, d.h. 2 x 8 Anschlüsse zur Peripherie. Weiterhin gehören zu jedem Kanal noch je ein Eingang  $\overline{STB}$  (Rückmeldesignal von der Peripherie) und ein Ausgang RDY (Bereitsignal zur Peripherie). Diese Anschlüsse unterstützen einen Quittungsbetrieb und werden bei der Beschreibung der Betriebsarten näher erläutert.

Zum Bussystem des Rechners hin besitzt der Schaltkreis Anschlüsse für den Datenbus, einige Steuersignale sowie drei Auswahlssignale. Eines davon wählt den angesprochenen Kanal (A oder B), ein zweites signalisiert den Schaltkreis, ob Daten zur Aus-/Eingabe oder ein Steuerwort zur Betriebsartenauswahl anliegen. Diese beiden Auswahlssignale werden üblicherweise mit Adreßleitungen verbunden und bewirken, daß der Schaltkreis 4 E/A-Adressen belegt:

- Kanal A - Daten
- Kanal A - Steuerworte
- Kanal B - Daten
- Kanal B - Steuerworte

Das dritte Auswahlssignal  $\overline{CE}$  (Chip Enable) dient der Auswahl des Schaltkreises (bei mehreren vorhandenen E/A-Schaltkreisen). Es ist mit der Adreßdekodierung verbunden.

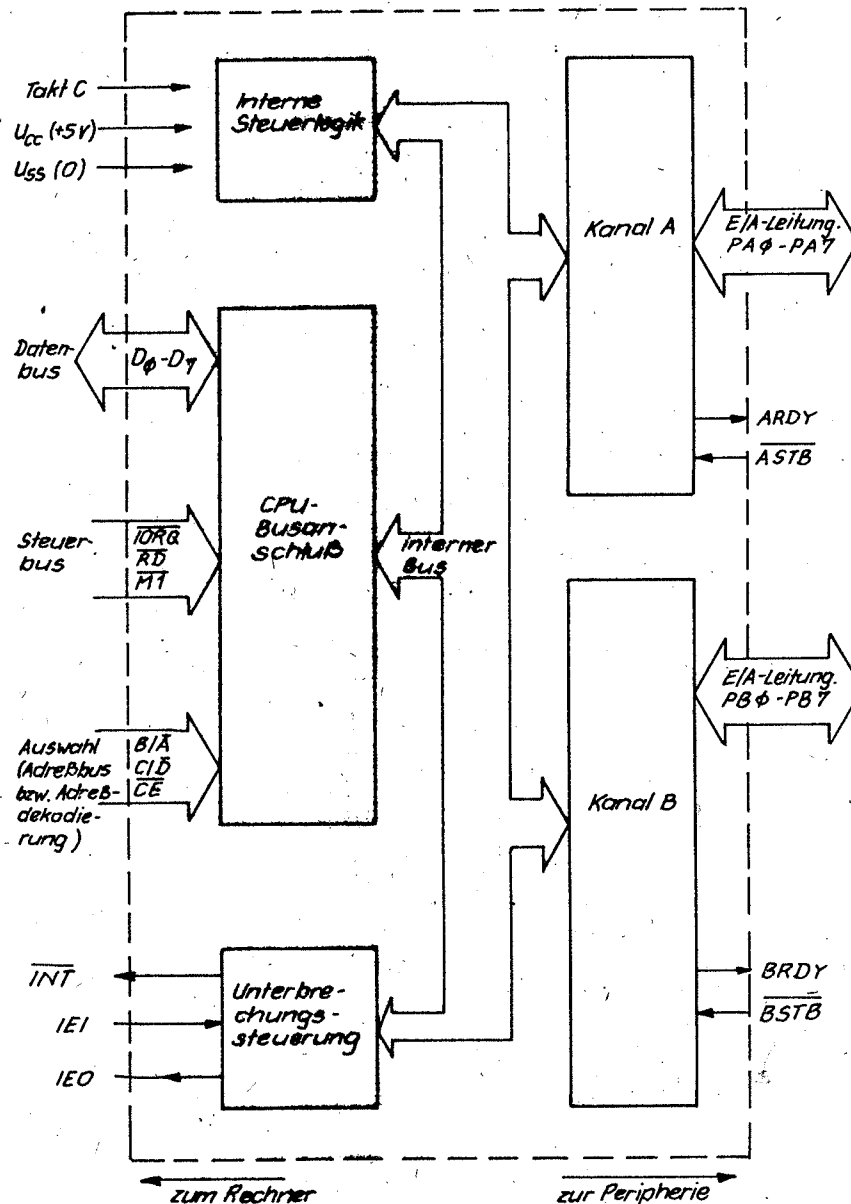


Bild 8.6. Grobstruktur des Schaltkreises U855 (PIO)

Die beiden Kanäle können unabhängig voneinander für eine von vier möglichen Betriebsarten programmiert werden:

#### Betriebsart 0 - Byte-Ausgabe

Die 8 Anschlüsse des Kanals wirken als Ausgänge, die Synchronisierung mit den angeschlossenen peripheren Einrichtungen erfolgt mit Hilfe der Quittungslogik des Schaltkreises. Diese Quittungslogik ist vor allem zur Unterstützung eines unterbrechungsgesteuerten Betriebes gedacht.

Der Funktionsablauf ist wie folgt:

Die Daten werden mit einem Ausgabebefehl der CPU zum PIO-Schaltkreis ausgegeben. Sie werden im Ausgaberegister des Schaltkreises gespeichert und erscheinen an den Kanalanschlüssen. Sobald die neuen Daten dort gültig sind, wird das Bereitsignal RDY (Ready) zur Peripherie aktiv, d.h. geht auf 1-Pegel. Wenn die angeschlossene Peripherie die Daten übernommen hat, so signalisiert sie das, indem sie das Rückmeldesignal  $\overline{STB}$  (Strobe) auf 0 und anschließend wieder auf 1-Pegel legt. Die ansteigende Flanke am  $\overline{STB}$ -Eingang des PIO-Bausteins bewirkt, daß das Bereitsignal RDY wieder 0-Pegel annimmt und daß eine Unterbrechung ausgelöst wird, wenn dies durch Programmierung des Bausteins erlaubt wurde. Eine Unterbrechung wird also immer dann ausgelöst, wenn die Peripherie die ausgegebenen Daten abgeholt hat und das nächste Datenbyte übernehmen kann. Zweckmäßigerweise übernimmt das Unterbrechungsbehandlungsprogramm die Ausgabe des nächsten Datenbytes.

#### Betriebsart 1 - Byte-Eingabe

Die 8 Anschlüsse des Kanals wirken als Eingänge. Die von der angeschlossenen Peripherie angelegten Daten werden mit der steigenden Flanke des Rückmeldesignals  $\overline{STB}$  in das Dateneingaberegister übernommen. Daraufhin wird das Bereitsignal (RDY) auf 0-Pegel gelegt, um der Peripherie zu signalisieren, daß vorläufig keine neuen Daten übernommen werden können. Außerdem wird eine Unterbrechung ausgelöst, die dem Rechner den Eingang neuer Daten signalisiert. Das Unterbrechungsbehandlungsprogramm holt dann die Daten mit einem Eingabebefehl ab. Dieses Abholen bewirkt, daß das Bereitsignal RDY wieder auf 1-Pegel gesetzt wird und damit der angeschlossenen Peripherie die Bereitschaft zur Übernahme neuer Daten signalisiert.

#### Betriebsart 2 - Byte Ein-/Ausgabe, Zweirichtungsbetrieb

Diese Betriebsart kombiniert Byte-Eingabe und Byte-Ausgabe. Die 8 Anschlüsse des Kanals wirken in Abhängigkeit von den Rückmeldesignalen:

Die Betriebsart 2 ist nur für Kanal A möglich, da alle vier Quittungssignale des Schaltkreises benötigt werden. Kanal B muß in diesem Fall in der Betriebsart 3 - Bit-Ein/Ausgabe betrieben werden, da nur diese Betriebsart keine Steuersignale benötigt. Das Quittungssignalpaar ARDY,  $\overline{ASTB}$  steuert die Byte-Ausgabe, das Paar BRDY,  $\overline{BSTB}$  die Byte-Eingabe. Das Signalspiel ist wie bei den Betriebsarten 0 und 1 beschrieben. Die Datenausgabe erfolgt nur während  $\overline{ASTB}=0$ , andernfalls sind die Kanalanschlüsse hochohmig, d.h. als Eingänge zu betrachten.

#### Betriebsart 3 - Bit-Ein-/Ausgabe

Diese Betriebsart entspricht weitgehend der im Abschnitt 8.1.1. beschriebenen einfachen Ein-/Ausgabetechnik. Die Quittungssignale RDY,  $\overline{STB}$  werden nicht benutzt. Durch ein Steuerwort wird definiert welche der 8 Leitungen als Eingang und welche als Ausgang wirken sollen. Wenn mit einem Ausgabebefehl ein Datenbyte zum Kanal ausgegeben wird, so wird dieses in das Ausgaberegister übernommen und dort gespeichert. Bei den als Ausgang definierten Bits bzw. Leitungen wird der entsprechende Logikpegel zum Peripherieanschluß durchgeschaltet. Beim Lesen von einem Kanal werden die augenblicklichen Pegel der als Eingänge definierten Leitungen auf den Datenbus durchgeschaltet. Für die als Ausgänge definierten Leitungen erscheint beim Lesen der Inhalt des Ausgaberegisters. Diese Betriebsart ist auch zweckmäßig, wenn ein Byte-Ein- oder Ausgabe ohne die Notwendigkeit eines Quittungsbetriebes mit RDY/ $\overline{STB}$  realisiert werden soll.

Es kann programmiert werden, daß ein bestimmter Logikpegel (0 oder 1) an einzelnen Kanalleitungen oder auch UND/ODER-Verknüpfungen von Kanalleitungen eine Unterbrechung auslösen.

8.6.2. Programmierung der PIO

In diesem Abschnitt soll der Anwender kennenlernen, wie sich die Betriebsart und weitere Betriebseigenschaften des Schaltkreises einstellen lassen.

Falls mit Unterbrechungen durch den Schaltkreis gearbeitet werden soll (IM 2), so ist zweckmäßigerweise zuerst der Unterbrechungsvektor in den Schaltkreis zu laden. Hierzu wird dieser mit einem Ausgabebefehl zur Steuerwortadresse des Schaltkreises ausgegeben:

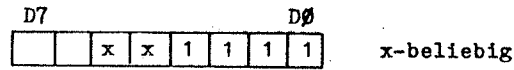
```
LD  A,INTVEC
OUT STEUAD
```

INTVEC-Unterbrechungsvektor=niederwertiger Teil der Ubr.-Tabellenadresse

STEUAD-Steuerwortadresse (im POLY-COMPUTER: Datenadresse +1)

Der Unterbrechungsvektor wird von anderen Steuerworten dadurch unterschieden, daß sein niederwertigstes Bit D0=0 ist.

Nun geben wir das Betriebsartensteuerwort aus. Es hat die Form



binäre Betriebsartennummer (z.B. 11B<sub>2</sub>) für Betriebsart 3)

Damit man nicht immer das hexadezimale Steuerwort aus der binären Darstellung ermitteln muß, sind hier die Steuerworte zusammengestellt (Achtung, wegen der nicht benutzten Bits sind auch einige andere Kodierungen zulässig)

Betriebsart	hex. Betriebsartensteuerwort
0 - Byte-Ausgabe	0F
1 - Byte-Eingabe	4F
2 - Byte-E/A	8F
3 - Bit-E/A	CF

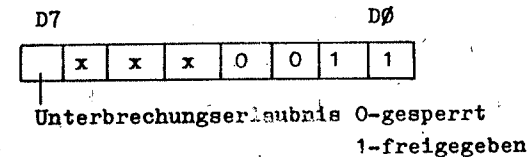
Bei der Betriebsart 3 (Bit-E/A) muß jetzt ein weiteres Steuerwort folgen, das definiert, welche Leitungen als Eingänge und welche als Ausgänge wirken sollen, dabei gilt die Zuordnung:

- 0-Ausgang
- 1-Eingang

Einige Beispiele:

	binär	hex
alle Leitungen als Ausgänge	0000 0000	00
alle Leitungen als Eingänge	1111 1111	FF
Bit 7 (also Leitung PA7/PB7) als Ausgang, übrige als Eingang	0111 1111	7F

Falls mit Unterbrechungen gearbeitet wird, so ist mit einem weiteren Steuerwort die Unterbrechung durch die PIO zu erlauben (nicht verwechseln mit Unterbrechungserlaubnismechanismus der CPU!). Dieses Steuerwort gestattet auch das Verboten und Erlauben von Unterbrechungen durch die PIO zu späteren Zeitpunkten. Unterbrechungs-Steuerwort:

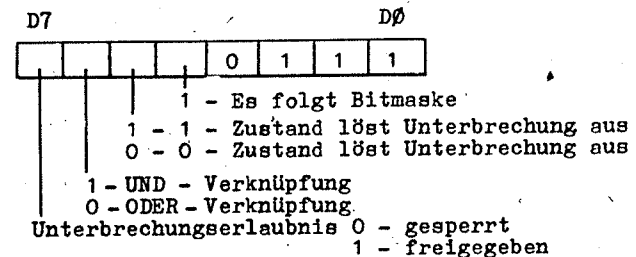


Die entsprechenden hexadezimalen Steuerworte sind:

- Unterbrechung sperren 03H
- Unterbrechung freigeben 83H

Wenn in der Betriebsart 3 (Bit-E/A) mit Unterbrechungen gearbeitet werden soll, so ist das eben beschriebene Steuerwort zwar ebenfalls zum Sperren bzw. Erlauben von Unterbrechungen anwendbar, vorher (im allgemeinen nach der Betriebsartenprogrammierung) müssen aber noch Steuerworte zur Festlegung der auslösenden Leitungen sowie deren Verknüpfung und Polarität ausgegeben werden.

Unterbrechungs-Steuerwort für Bit-E/A:



Als nächstes Steuerwort folgt eine Bitmaske zur Festlegung der in die Verknüpfung einbezogenen Leitungen mit der Zuordnung

0-Leitung einbezogen  
1-Leitung nicht einbezogen

Als Beispiel sei gefordert, daß vom Kanal A eine Unterbrechung ausgelöst wird, wenn an den Leitungen PA7 oder PA6

1-Potential auftritt:

	binär	hex
Unterbrechungs-Steuerwort für Bit-E/A:	1011 0111	B7
darauffolgende Bitmaske:	0011 1111	3F

### 8.6.3. Beispiel

Für Experimente mit dem Schaltkreis PIO benötigt man eigentlich eine Peripherie, zum Beispiel einige Schalter und Pegelanzeigen an den peripherieseitigen Anschlüssen des Schaltkreises. Dies ist im Grundgerät POLY-COMPUTER noch nicht vorhanden, mit Hilfe eines kleinen "Tricks" ist aber das nun folgende Programmbeispiel ausführbar, das sowohl die Programmierung des Schaltkreises als auch die Reaktion der CPU auf eine Unterbrechungsanforderung demonstriert. Wir programmieren einen PIO-Kanal auf Betriebsart 3 - Bit-Ein-/Ausgabe und definieren alle Leitungen als Ausgänge. Dies ist nicht identisch mit der Betriebsart 0-Byte-Ausgabe, denn die Quittungsleitungen RDY/STB werden nicht benutzt. Nach dieser Programmierung können wir auf den Kanal Datenbytes ausgeben, diese stehen dann bis zur Ausgabe des nächsten Bytes an den Kanalanschlüssen an und könnten von der Peripherie verarbeitet werden. Wir haben zwar keine Peripherie angeschlossen, wir können aber einen Eingabebefehl zur PIO ausführen und erhalten das ausgegebene Byte wieder zurück (sozusagen als Bestätigung, daß dieser Wert gerade ausgegeben wurde). Durch ein weiteres Steuerwort können wir die Unterbrechungserzeugung durch die PIO gestatten. Dabei sind nicht nur die als Eingänge definierten Leitungen, sondern beliebige Leitungen als Unterbrechungsursache definierbar. Wir fordern beispielsweise, daß eine Unterbrechung ausgelöst wird, wenn das höchstwertigste Bit (D7) auf 1-Pegel liegt. Dies würde bei Eingabe bedeuten, daß 1-Pegel an dieser Leitung als Unterbrechungsursache wirkt. Da wir nicht von außen eingreifen wollen, haben wir die Leitung als Ausgang definiert und können nun von der CPU aus ein Datenbyte mit D7=1 auf den PIO-Kanal ausgeben, um eine Unterbrechung herbeizuführen!

Schreiben Sie das nun folgende Programm in den Speicher ein und verfolgen Sie dessen Ausführung:

Adresse Befehlscode ; Betätigen Sie die RESET-Taste, um definierte (hex) (hex) ; Ausgangsbedingungen zu schaffen und lassen Sie ; die folgenden Befehle im Befehlsschrittbetrieb ; ausführen!

4000	3E 80	LD A,80H	; Unterbrechungsvektor
4002	D3 85	OUT 85H	; auf Steuerwortadresse ausgeben
4004	3E CF	LD A,0CFH	; Betriebsart 3
4006	D3 85	OUT 85H	
4008	3E 00	LD A,0	; alle Leitungen als Ausgänge
400A	D3 85	OUT 85H	
			; Die PIO ist jetzt programmiert es erfolgt eine Ausgabe und ein Kontrolleinlesen
400C	3E 55	LD A,55H	; Testdaten
400E	D3 84	OUT 84H	; auf Datenadresse ausgeben
4010	AF	XOR A	; A-Register löschen
4011	DB 84	IN 84H	; Einlesen von Datenadresse
			; an dieser Stelle können Sie sich das A-Register ansehen und feststellen, daß der Wert 55H wieder eingelesen wurde
			; Ausgabe des Unterbrechungssteuerwortes für Bit-E/A
4013	3E F7	LD A,0F7H	; 1-Zustand löst aus
4015	D3 85	OUT 85H	
4017	3E 7F	LD A,7FH	; nur Bit7=0 (löst aus)
4019	D3 85	OUT 85H	
			; Wir geben jetzt ein Datenbyte mit D7=1 aus
401B	3E FF	LD A,0FFH	
401D	D3 84	OUT 84H	
			; Erwartungsgemäß erzeugt die PIO eine Unterbrechung, erkennbar am Aufleuchten der Anzeige INT des Busanalysators
			; Arbeiten Sie jetzt im Einzelzyklusbetrieb weiter, (MCYCL betätigen), damit Sie die Annahme der Unterbrechung genau verfolgen können.
401F	00	NOP	
4020	00	NOP	
4021	00	NOP	; Leerbefehle
4022	00	NOP	
4023	00	NOP	
4024	00	NOP	



;Die Unterbrechung ist noch nicht angenommen worden,  
;weil das Unterbrechungserlaubnis-Flip-Flop der  
;CPU noch rückgesetzt ist!

4025 FB EI ;Unterbrechungen erlauben  
4026 18 FE M1: JR M1;Sprung auf sich selbst  
;Den letzten Befehl führt die CPU solange aus, bis  
;sie die Unterbrechung akzeptiert.  
  
;Der Unterbrechungsannahmezyklus ist daran erkenn-  
;bar, daß M1 und IORQ gleichzeitig aktiv sind.

Sie werden feststellen, daß während dieses Zyklus von der PIO der ganz am Anfang ausgegebene Unterbrechungsvektor 80H=10000000B auf den Datenbus anliegt. Da kein Befehl zur Änderung der Unterbrechungs- betriebsart der CPU ausgeführt wurde, arbeitet diese in IMO, d.h. sie führt den anliegenden Datenbusinhalt wie einen Befehl aus dem Speicher aus. 80H ist gerade der Befehlscode ADD B, auf diese Weise ist also keine sinnvolle Reaktion auf eine Unterbrechung möglich. Wir müßten schon erreichen, daß eine Programmverzweigung ausgeführt wird, wozu sich die Kurzruffehle (RST) hier besonders eignen. Deren Befehlscode hat jedoch Bit 0=1, so daß er nicht als "Unter- brechungsvektor" in die PIO ladbar ist. Wir schlußfolgern, daß ein Betrieb der PIO (wie auch anderer programmierbarer Bausteine der Schaltkreisfamilie) in IMO nicht ohne weiteres möglich ist. Die Betriebsart IMO ist vorrangig zu benutzen, wenn eine spezielle Schaltung vorhanden ist, die RST-Befehle bei der Unterbrechungsannahme auf den Datenbus schaltet. Versuchen wir es jetzt mit der nächsten Unterbrechungsbetriebsart IM1, indem wir die Leerbefehle auf Adresse 401F durch den Befehl zum Übergang auf IM1 ersetzen:

Adresse	hex. Kode	
4023	ED 56	IM1

Führen Sie das Programm noch einmal aus, vergessen Sie nicht, vor- her die RESET-Taste zu betätigen!

Beim Unterbrechungsannahmezyklus legt die PIO wieder den Wert 80H auf den Datenbus. Dieser wird aber diesmal ignoriert (wir hätten also auch auf dessen Einschreiben in die PIO verzichten können).

Wenn Sie weitere Zyklen ausführen lassen, können Sie beobachten, wie erst der niederwertige und dann der höherwertige Teil, des alten Programmzählers auf dem Stapelspeicher (Stack) abge- speichert werden und anschließend ein Befehlslesezyklus (M1, MREQ, RD) von Adresse 3BH (00000000 00111000 B) erfolgt.

Diese Unterbrechungsbetriebsart führt also in jedem Fall zu einer festen Adresse, im Fall des POLY-COMPUTER steht dort gleich ein Sprungbefehl zur Adresse 4000, so daß man dort ein Unterbrechung- behandlungsprogramm eingeben könnte. Die Betriebsart IM1 ist also in Systemen sinnvoll, in denen nur ein Unterbrechungsbehandlungs- programm (d.h. meist nur eine Unterbrechungsursache) vorkommt.

Wir kommen nun zur leistungsfähigsten, aber auch kompliziertesten Unterbrechungsbetriebsart IM2.

Ersetzen Sie den Befehl IM1 durch IM2:

Adresse	hex. Kode	
4023	ED 5E	IM2

Führen Sie das Programm aus! RESET nicht vergessen!

Nach dem Unterbrechungsannahmezyklus erfolgt wieder das Ab- speichern des alten Programmzählers. Anschließend erfolgt ein Lesen von Adresse 00000000 10000000 B = 80H.

Diese Adresse setzt sich zusammen aus dem I-Register des Prozes- sors (nach dem Rücksetzen 0) und dem aus der PIO gelesenen Unterbrechungsvektor.

Die Adresse liegt im ROM-Bereich, wir merken uns die gelesenen Daten (00010001B=11H, 00111011B=3BH).

Der nächste Zyklus stellt einen Befehlsholezyklus von der Adresse 0011110100010001B= 3B11H dar; wir erkennen, daß diese sich aus den gerade gelesenen Bytes zusammensetzt.

Damit wir eine beliebige Startadresse des Unterbrechungsbehand- lungsprogramms festlegen können, muß erreicht werden, daß die CPU diese Adresse aus dem RAM-Bereich des POLY-COMPUTERS liest. Dies wird ermöglicht durch Laden des I-Registers mit dem höher- wertigen Teil der Tabellenadresse. Wir fordern beispielsweise, daß das Unterbrechungsbehandlungsprogramm auf Adresse 4100H be- ginnt, die Tabelle (die hier nur eine einzige Startadresse ent- hält) befindet sich ab Adresse 4080H.

Um diese Forderungen zu erfüllen, müssen wir folgendes ausführen:

- Abspeichern der Startadresse in der Tabelle in der Reihenfolge niederwert./höherwert. Teil:

Adresse	Daten
4080	00
4081	41

- Laden des I-Registers mit dem höherwertigen Teil der Tabellenadresse; dazu fügen wir anstelle der Leerbefehle ab Adresse 401FH ein:

Adresse	Daten
401F	3E 40 LD A,40
4021	ED 47 LD I,A

- Das Unterbrechungsvektorregister der PIO muß mit dem niederwertigen Teil der Tabellenadresse, d.h. mit 80H geladen werden, was bereits im Programm geschieht.

Führen Sie nun das Programm wieder aus, bis die CPU den Beginn des Unterbrechungsbehandlungsprogramms (4100H) erreicht hat!

Falls etwas nicht so arbeitet, wie beschrieben, haben Sie wahrscheinlich eine Speicherbelegung nicht richtig eingegeben oder das Rücksetzen vor der Programmabarbeitung vergessen.

### 8.7. Der Schaltkreis U857 (CTC)

#### 8.7.1. Überblick

In Mikrorechnersystemen mit Steuerungsaufgaben besteht häufig die Notwendigkeit, definierte Zeitintervalle bzw. Frequenzen zu erzeugen oder externe Ereignisse zu zählen. Die CPU selbst kann diese Aufgabe übernehmen, da ja ihr Takt quarzstabilisiert ist und die Taktzyklusanzahl der einzelnen Befehle genau bekannt ist. Um eine bestimmte Zeitverzögerung zu erreichen, kann man nun eine Befehlsfolge (meist eine Programmschleife, in der ein Register bis auf einen bestimmten Wert gezählt wird) programmieren, deren Abarbeitung gerade die benötigte Zeitverzögerung lang andauert. Dieses Verfahren ist in vielen Fällen anwendbar; trotzdem gibt es Aufgaben, die damit nicht lösbar sind. Beispielsweise dann, wenn die Forderung nach mehreren oder kontinuierlichen Zeitgebern besteht, wenn der Rechner neben der Zeitmessung noch Verarbeitungen ausführen soll oder wenn sich ein externes auszuwertendes Signal so schnell ändert, daß eine programmtechnische Auswertung nicht mehr möglich ist. Für alle diese Fälle ist der Einsatz eines Zähler/Zeitgeberschaltkreises sinnvoll. In der von uns benutzten Schaltkreisfamilie ist dies der Typ U857, auch als CTC (Counter/Timer Circuit) bezeichnet. Bild 8.6. zeigt die Grobstruktur dieses Schaltkreises. Er enthält vier Zählkanäle, deren Struktur im Bild 8.7. dargestellt ist. Der wesentlichste Bestandteil eines Zählkanals ist der 8-Bit-Rückwärtszähler. Dieser Zähler wird vom Zeitkonstantenregister nach dem Rücksetzen sowie nach jedem Erreichen des Zählerstands 0 geladen. Wenn der Rückwärtszähler den Zählerstand 0 erreicht, gibt er am Ausgang ZC/TO (nur bei den Kanälen 0,1,2 vorhanden) einen  $\neg$ -Impuls ab; wenn erlaubt, wird eine Unterbrechung angemeldet.

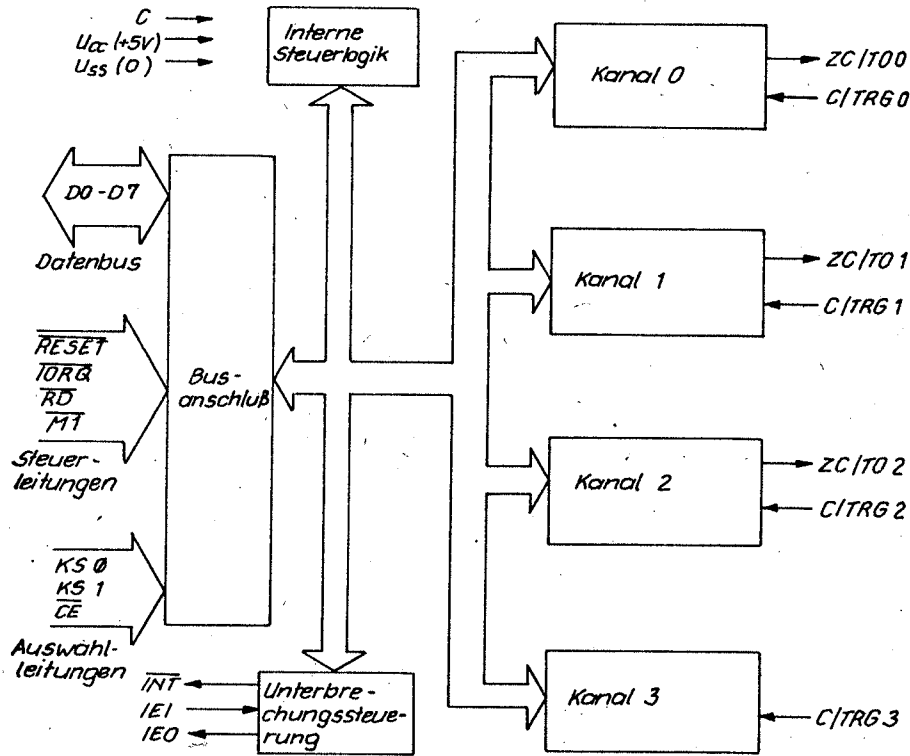


Bild 8.6. Grobstruktur U857D (CTC)

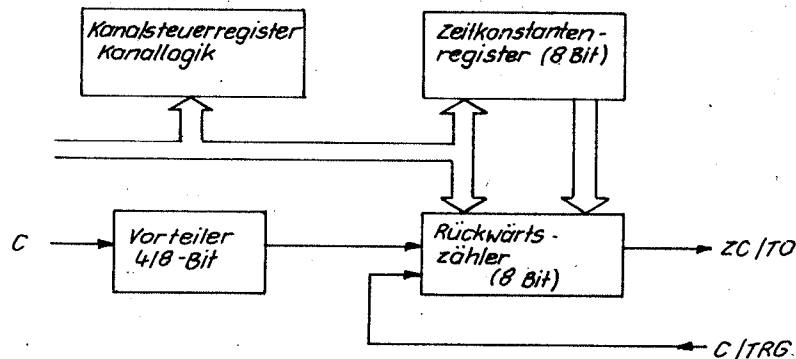


Bild 8.7. Struktur eines CTC-Kanals

Der Takt des Rückwärtszählers kann in Abhängigkeit von der Betriebsart aus verschiedenen Quellen stammen:

**Betriebsart Zähler:** Der Zählerstand wird bei jedem Impuls an dem externen Eingang C/TRG um 1 verringert.

Die zu zählende Flanke ist durch Programmierung festlegbar.

**Betriebsart Zeitgeber:** Der Rückwärtszähler wird über einen Verteiler wahlweise mit  $\frac{1}{16}$  oder  $\frac{1}{256}$  des Systemtaktes getaktet. Der Eingang C/TRGn (n=0,1,2,3) kann als Freigabeeingang für den Beginn der Zeitmessung dienen. Nach der erfolgten Freigabe hat er jedoch keinen Einfluß mehr auf den CTC.

Zum Rechner hin besitzt der CTC Anschlüsse für den Datenbus, einige Steuersignale sowie die Signale für Baustein- und Kanalauswahl. Die beiden Kanalauswahlleitungen KSO, KS1 werden meist mit den Adreßleitungen A0, A1 verbunden, so daß der Schaltkreis vier (aufeinanderfolgende) E/A-Adressen für die Kanäle 0-3 belegt. Mittels Schreiboperationen zur Kanaladresse werden das Betriebsartensteuerwort und die Zeitkonstante programmiert, bei Leseoperationen von der Kanaladresse wird der augenblickliche Stand des Rückwärtszählers gelesen.

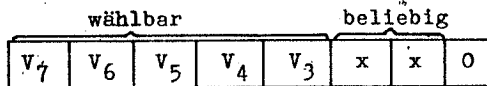
Beim Schreib- und Leszugriff zum CTC dürfen keine zusätzlichen Wartezyklen eingeführt werden, diese Zugriffe sollten also insbesondere nicht im Zyklusschrittbetrieb ausgeführt werden.

8.7.2. Programmierung des CTC

Wenn mit Unterbrechungen (IM2) gearbeitet werden soll, so ist der Unterbrechungsvektor in den CTC zu laden. Im Gegensatz zum Schaltkreis PIO hat hier nicht jeder Kanal einen beliebig wählbaren Unterbrechungsvektor. Der Unterbrechungsvektor ist nur in Kanal 0 einzugeben, er wird von Betriebsartensteuerworten unterschieden durch Bit 0=0!

Bit 1 und Bit 2 können beliebig sein, hier wird bei einer Unterbrechung vom CTC die binäre Kanalnummer eingesetzt.

In Kanal 0 einzugebender Vektor:



Erzeugter Vektor bei Unterbrechung durch

Kanal 0	V <sub>7</sub>	V <sub>6</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>3</sub>	0	0	0
Kanal 1	V <sub>7</sub>	V <sub>6</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>3</sub>	0	1	0
Kanal 2	V <sub>7</sub>	V <sub>6</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>3</sub>	1	0	0
Kanal 3	V <sub>7</sub>	V <sub>6</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>3</sub>	1	1	0

Durch ein Kanalsteuerwort wird die genaue Betriebsweise des Kanals festgelegt. Die einzelnen Bits darin haben folgende Bedeutung

- D0 - 1-Kennzeichnung des Steuerwortes als Kanalsteuerwort
- D1 - 1-Kanal rücksetzen, Fortsetzung der Arbeit erst nach dem Laden einer Zeitkonstante bzw. eines neuen Kanalsteuerwortes
  - 0-Kanal zählt weiter, eine eventuell folgende Zeitkonstante wird erst nach Erreichen des Zählerstandes 0 geladen
- D2 - 1-Das nächste Steuerwort ist als Zeitkonstante zu interpretieren
  - 0-Es folgt keine Zeitkonstante
- D3 - nur für Betriebsart Zeitgeber(Freigabe)
  - 0-Zeitgeber läuft sofort nach Laden der Zeitkonstante an
  - 1-Zeitgeber läuft erst nach externer Freigabe (C/TRG) an

- D4 - wirksame Flanke an C/TRG
  - 0 - fallende Flanke
  - 1 - steigende Flanke
- D5 - nur für Betriebsart Zeitgeber(Vorteilerfaktor)
  - 0 - Vorteilerfaktor 16
  - 1 - Vorteilerfaktor 256
- D6 - Betriebsart
  - 0 - Zeitgeber
  - 1 - Zähler
- D7 - Unterbrechungsfreigabe
  - 0 - gesperrt
  - 1 - freigegeben

Bei der nun möglicherweise folgenden Zeitkonstante ist zu beachten, daß der Wert 0 einer Zählung bis 256 entspricht. Einige Beispiele sollen die Programmierung demonstrieren: Betriebsart Zeitgeber ohne externe Freigabe, Vorteiler 256, Unterbrechung eingegeben:

Betriebsartensteuerwort 10100111B=A7H  
Zeitkonstante z.B. 80H

Betriebsart Zähler für steigende Flanken an C/TRG, keine Unterbrechungen

Betriebsartensteuerwort 01010111 = 57H  
Zeitkonstante (Zähleranfangswert) z.B. 0H

Der Kanal arbeitet ständig, wenn er einmal gestartet (programmiert) wurde, d.h. nach jedem Erreichen des Zählerstandes 0 wird automatisch wieder die Zeitkonstante geladen, der Zählvorgang des Rückwärtszählers beginnt erneut. Um die Arbeit des Kanals zu beenden, kann man ein Steuerwort mit D1=1 (Kanal rücksetzen) ohne nachfolgende Zeitkonstante ausgeben, z.B.

01000011B= 43H

Der Kanal arbeitet erst wieder nach Ausgabe eines neuen Steuerwortes und einer Zeitkonstante.

8.7.3. Beispiel

Eine häufige Anwendung für den CTC ist die Bereitstellung eines Taktes für periphere Einrichtungen. Wir stellen uns in diesem Beispiel die Aufgabe, an einem Ausgang alle 2,5 ms einen Impuls zu erzeugen. Wir nehmen weiterhin an, daß eine Erzeugung durch Programmlaufzeiten (wie in Abschnitt 8.7.1. angedeutet) nicht wünschenswert ist. Die Aufgabe ist also durch Einsatz eines Zeitgeber-Kanals zu lösen, der die geforderten Impulse an seinem Ausgang ZC/TO abgibt.

Zunächst ist die erforderliche Zeitkonstante zu ermitteln. Bei Zeitgeberbetrieb gehen wir vom Prozessortakt aus; dieser hat beim POLY-COMPUTER eine Taktfrequenz  $f_c = 921,6$  KHz. Damit wird das erforderliche Teilungsverhältnis

$$K = f_c \cdot t = 921,6 \text{ KHz} \cdot 2,5 \text{ ms} = 2304$$

Es setzt sich zusammen aus Verteilerverhältnis und zu programmierender Zeitkonstante:  $K = K_v \cdot K_x$ .

Mit einem gewählten Verteilerfaktor  $K_v = 256$  wird  $K_x = \frac{K}{K_v} = \frac{2304}{256} = 9$

Das Beispiel wurde so gewählt, daß sich ein "glatter" Wert für den Teilerfaktor ergibt. Dies wird nicht in jedem Fall so sein, die Taktfrequenz des POLY-COMPUTERS ist jedoch so gewählt, daß häufig benutzte "glatte" Werte und vor allem die den standardisierten Datenübertragungsraten entsprechenden Frequenzen ganzzahlige Teilerfaktoren ergeben.

Um das Zählen verfolgen zu können, lesen wir den Zählerstand in dichtestmöglicher Folge in den Speicher ein. Wir benutzen Kanal 1 des im POLY-COMPUTER enthaltenen CTC, dieser hat die Adresse 89H:

Adresse	Daten		
4000	3E 27	LD A,27H	;Zeitgeber Verteilerfaktor 256, ;keine Unterbrechungen
4002	D3 89	OUT 89H	
4004	3E 09	LD A,9	
4006	D3 89	OUT 89H	
			;der Zähler läuft jetzt
4008	06 FF	LD B,0FFH	;255 Werte einlesen
400A	21 00 41	LD HL,4100H	;Adresse
400D	DB 89	M1:IN 89H	;einlesen von 4100 - 41FF
400F	77	LD (HL),A	

```
4010 23      JNC HL
4011 10 FA    DJNZ M1
4013 76      HALT
```

Starten Sie das Programm mit dem Kommando GO 4000; es ist in sehr kurzer Zeit durchlaufen (HALT-Anzeige). Nach Betätigen der **MON** Taste sind die Zählerstände in Adressbereich 4100 - 41FF zu verfolgen.

8.8. Unterbrechungssteuerung mit Peripherieschaltkreisen  
8.8.1. Die Prioritätslogik der E/A-Schaltkreise

Bisher wurde davon ausgegangen, daß immer nur ein Peripherieschaltkreis eine Unterbrechung anmeldet. Es kann aber durchaus der Fall eintreten, daß mehrere Schaltkreise (oder mehrere Kanäle innerhalb eines Schaltkreises) gleichzeitig eine Unterbrechung anmelden. Es muß also eine Koordinierungslogik vorhanden sein, die verhindert, daß während des Unterbrechungsannahmezyklus mehrere Bausteine ihren Vektor auf den Datenbus legen. Ein besonderes Kennzeichen der Schaltkreisfamilie U855, U856, U857 ist es, daß diese Logik in Form einer Prioritätskette (Daisy Chain) bereits in den Bausteinen enthalten ist. Beim Entwurf des Mikrorechners ist nun eine Reihenfolge der Dringlichkeiten (Prioritäten) festzulegen. Jeder Peripherieschaltkreis besitzt die Anschlüsse IEI-Unterbrechungsfreigabeeingang (Interrupt Enable In)- und IEO-Unterbrechungsfreigabeausgang (Interrupt Enable Out). Der Eingang IEI des am höchsten priorisierten Schaltkreises wird mit 1-Pegel (+5V) verbunden, das erlaubt diesem Schaltkreis, jederzeit eine Unterbrechung anzumelden. Der Ausgang IEO liegt im Ruhezustand auf 1-Pegel. Wenn im betrachteten Schaltkreis eine Unterbrechung angemeldet wurde oder gerade behandelt wird, dann signalisiert der 0-Pegel am Ausgang IEO den weiteren (niedriger priorisierten) E/A-Einheiten, daß sie keine Unterbrechung anmelden dürfen. Die folgenden E/A-Einheiten schalten diesen 0-Pegel durch. Bild 8.8 zeigt die entstehende Kettenstruktur; (diese betrifft nicht nur die Zusammenschaltung mehrerer Schaltkreise, sondern ist auch innerhalb der Schaltkreise zwischen den Kanälen vorhanden). Beim Schaltkreis U855 (PIO) ist dabei Kanal A, beim U857 (CTC) Kanal 0 am höchsten priorisiert.

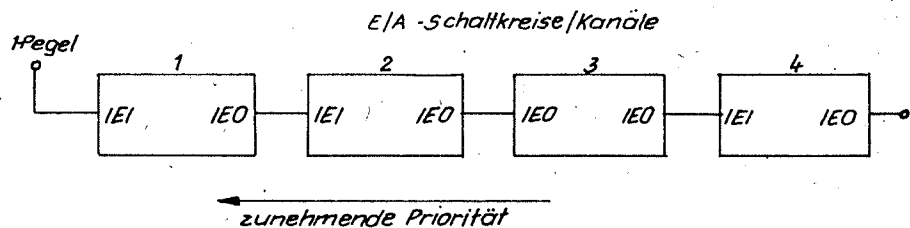


Bild 8.8. Prioritätskette der E/A-Einheiten

Damit eine periphere Schaltung das Ende ihres Behandlungsprogramms erkennen kann, wurde der spezielle Befehl RETI (Rücksprung von Unterbrechungsbehandlung, Abschnitt 8.4.3.) eingeführt. Dieser Befehl wirkt für die CPU wie RET, die periphere Schaltung dagegen die gerade behandelt wurde (erkennbar am IEI=1, IEO=0). erkennt den Befehlskode RETI und setzt daraufhin ihr Signal IEO auf 1, wodurch Unterbrechungen durch niedriger priorisierte Bausteine wieder zugelassen werden.

Der eben beschriebene Mechanismus gestattet eine verschachtelte Unterbrechungsbearbeitung, die an einem Beispiel erläutert sei (Bezeichnungen siehe Bild 8.8.):

Baustein 3 meldet eine Unterbrechung an (legt  $\overline{\text{INT}}$  auf 0-Pegel). Die CPU nimmt die Unterbrechung an (daraufhin legt der Baustein  $\overline{\text{INT}}$  wieder auf 1) und erlaubt durch den Befehl EI weitere Unterbrechungen. Solange Baustein 3 in Behandlung ist, kann Baustein 4 keine Unterbrechung anmelden (sein IEI ist 0). Der höher priorisierte Baustein 1 dagegen kann eine Unterbrechung anmelden, die CPU wird diese annehmen, das Behandlungsprogramm von Baustein 3 verlassen und den Baustein 1 behandeln. Nach Abschluß des Unterbrechungsprogramms von Baustein 1 (RETI) wird dasjenige von Baustein 3 fortgesetzt, bis auch dieses beendet ist (RETI) und die Prioritätskette wieder in ihren Grundzustand zurückgeht (alle IEI, IEO=1). In manchen Fällen ist die Verschachtelung unerwünscht. In dem man die Unterbrechbarkeit der CPU erst unmittelbar vor den RETI-Befehl mit EI freigibt, erreicht man, daß Unterbrechungsbehandlungen nicht ihrerseits wieder unterbrochen werden können.

### 8.8.2. Programmbeispiel-Unterbrechungsgesteuerte Digitaluhr

Zur Anwendung der Kenntnisse über die Programmierung der Peripherieschaltkreise und die Unterbrechungsverarbeitung folgt hier ein Programmierbeispiel, das auf dem POLY-COMPUTER das Verhalten einer Digitaluhr nachbildet.

Wie wir bereits in Abschnitt 8.7.3. sehen konnten, sind mit einem Zeitgeberkanal nicht beliebig lange Intervalle zu erzeugen, so daß eine direkte Erzeugung des notwendigen 1s-Taktes nicht erfolgen kann. Wir gehen davon aus, daß der Zeitgeberkanal alle 10ms eine Unterbrechung erzeugen soll. Wenn wir dann 100 Unterbrechungen abzählen, erhalten wir den gewünschten 1s-Takt. Von dort ausgehend können wir nach 60s die Minuten weiterschalten nach 60 Minuten die Stunden und ein Rückstellen der Stunden beim Stand 24.00 Uhr vornehmen.

Weiterhin müssen wir beachten, daß ein Unterbrechungsbehandlungsprogramm alle von ihm benutzten Register retten muß, um die unterbrochenen Hintergrundprogramme nicht zu stören.

Den entstehenden Programmablaufplan zeigt Bild 8.9..

Es ist nun eine Entscheidung zu treffen über die rechnerinterne Darstellung der Uhrzeit:

Alle Zählerwerte für 1/100 s, s, min, h werden in je einem Byte dargestellt, die 1/100 s als binärer Wert; die Sekunden, Minuten, Stunden dagegen als BCD-Wert (binär codierter Dezimalwert; siehe Kapitel 9). Dieser bringt den Vorteil einer sehr einfachen Anzeigumsetzung, außerdem kann mit der MEM Funktion des Monitors die Uhrzeit ohne Umsetzungen auf einen bestimmten Wert voreingestellt werden. Die BCD-Zahlendarstellung hat zur Konsequenz, daß zum Weiterschalten nicht mehr beispielsweise der Befehl INCA sondern die Befehlsfolge ADD 1; DAA zu benutzen ist (Korrektur beim Auftreten von Pseudotetraden; anstelle 09+1=10 würde sonst 09+1=0AH entstehen; Einzelheiten siehe Kapitel 9).

Das Programm stellt in dieser Form die Uhrzeit im Speicher bereit, wir möchten sie jedoch auf der Anzeige sehen. Aus diesem Grund benötigen wir noch ein Hintergrundprogramm, das folgende Aufgaben erfüllt:

1. Programmierung und Start des CTC (wir benutzen Kanal 1/Adresse 89H)
2. ständige Aktualisierung der 7-Segment-Anzeige mit der im Speicher vom Unterbrechungsbehandlungsprogramm bereitgestellten Uhrzeit.

Wir arbeiten mit der Unterbrechungsbetriebsart 2 (IM2).

Zur Umwandlung in die 7-Segment-Darstellung und deren Ausgabe auf die Anzeige benutzen wir einige Hilfen durch das Monitorprogramm. Wir greifen zurück auf ein in ROM des POLY-COMPUTER bereits vorhandenes Programm KONSOL (Kapitel 6), welches die Ansteuerung der 8 7-Segment-Anzeigen mit einer in einem RAM-Bereich ANZBER vorgegebenen Belegung vornimmt (jedes Byte dieses Bereiches entspricht einer Anzeigestelle, jedes Bit einem Segment/Punkt). Für die Umwandlung der binären bzw. BCD-Darstellung benutzen wir eine ebenfalls im ROM vorhandene Tabelle, die die 7-Segment-Kodes der Binärziffern 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E,F in dieser Reihenfolge enthält und auf der Adresse ANZDEC beginnt. (Nähere Einzelheiten zu diesen Programmen/Daten sind, falls erforderlich, dem Bedienhandbuch zu entnehmen). Den Programmablaufplan des Hintergrundprogramms zeigt Bild 8.10. Die Liste des gesamten Programms enthält Bild 8.11. Dazu sind einige Erläuterungen notwendig. Die Liste enthält neben den Mnemoniks der Maschinenbefehle noch einige weitere Operationen, die keine Maschinenbefehle darstellen sondern organisatorische Anweisungen zur Speicherzuweisung (siehe auch Kapitel 10):

ORG x

Der folgende Programmkode beginnt auf Adresse x

(MARKE) DB x

Unter der Adresse MARKE wird ein Byte reserviert und mit dem Wert x belegt (Define Byte)

(MARKE) DA x

Ab der Adresse MARKE werden zwei Bytes reserviert und mit dem Wert x in der für Adressen üblichen Reihenfolge niederwert./höherwert. Byte belegt. (Define Adress)

(MARKE) EQU x

Der Wert x ist an allen Stellen im Programm für die symbolische Bezeichnung MARKE zu verwenden. (Equates)

Sehen Sie sich das Programm genau an, bis Sie es wirklich verstehen, Nach erfolgter Eingabe wird es auf der Adresse 4040H gestartet.

Die Uhr läuft bei der im RAM stehenden Zeit 00.00.00. an, zum Stellen können Sie das Programm mit **MON** unterbrechen, mit der **MEM**-Funktion die gewünschte Uhrzeit in den RAM einschreiben (Speicherplätze SEK, MIN, STUN) und mit **GO** (ohne Angabe einer Adresse) wieder starten. Beachten Sie, daß die "Uhr" steht, solange das Monitorprogramm läuft.

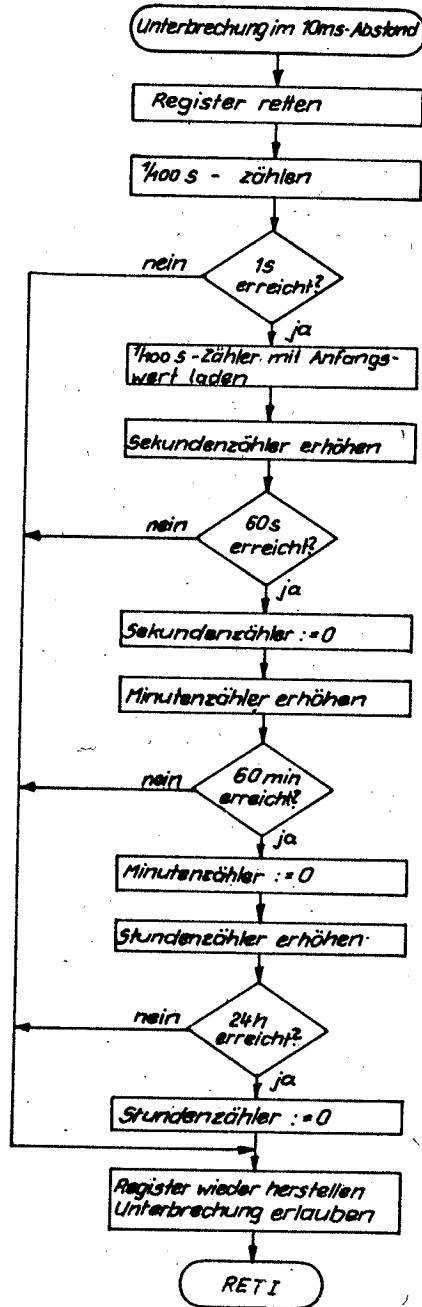


Bild 8.9. Programmablaufplan Unterbrechungsbehandlung für Digitaluhr

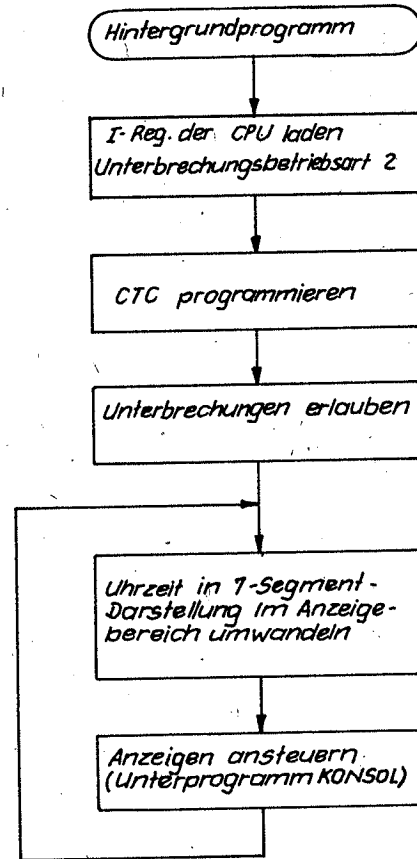


Bild 8.10 Programmablaufplan Hintergrundprogramm für Digitaluhr



ADR	OBJ-KODE	NR	POLY.UHR QUELLANWEISUNG	SEITE 1 POLY-880 ASM 1.0
		1	PROGRAMM ZUR REALISIERUNG EINER	
		2	DIGITALUHR MIT CTC-BENUTZUNG	
		3	STELLMOEGELICHKEIT MIT MONITOR	
		4		
		5	UNTERBRECHUNGSBEHANDLUNGSPROGRAMM	
4000		6	ORG 4000H	
4000	F5	7	UBR PUSH AF	
4001	E5	8	PUSH HL ; BENUTZTE REG. RETTEN	
4002	21C040	9	LD HL, HSEK	
4005	35	10	DEC (HL) ; 1/1005 BINARER ZAEHLEN	
4006	2023	11	JRNZ UBR1	
4008	3664	12	LD (HL), 100 ; UNTERSETZ.-FAKTOR	
400A	23	13	INC HL ; HL ZEIGT AUF SEK.	
400B	7E	14	LD A, (HL)	
400C	C601	15	ADD 1 ; SEKUNDEN WEITERSCHALTEN	
400E	27	16	DAA	
400F	77	17	LD (HL), A	
4010	D660	18	SUB 60H ; 60S ERREICHT ?	
4012	2017	19	JRNZ UBR1	
4014	77	20	LD (HL), A	
4015	23	21	INC HL	
4016	7E	22	LD A, (HL)	
4017	C601	23	ADD 1 ; MINUTEN WEITERSCHALTEN	
4019	27	24	DAA	
401A	77	25	LD (HL), A	
401B	D660	26	SUB 60H ; 60MIN ERREICHT ?	
401D	200C	27	JRNZ UBR1	
401F	77	28	LD (HL), A	
4020	23	29	INC HL	
4021	7E	30	LD A, (HL)	
4022	C601	31	ADD 1 ; STUNDEN WEITERSCHALTEN	
4024	27	32	DAA	
4025	77	33	LD (HL), A	
4026	D624	34	SUB 24H ; 24 STUNDEN ERREICHT ?	
4028	2001	35	JRNZ UBR1	
402A	77	36	LD (HL), A	
402B	E1	37	UBR1 POP HL	
402C	F1	38	POP AF ; REG. WIEDERHERSTELLEN	
402D	FB	39	EI	
402E	ED4D	40	RETI	
		41		
		42	HINTERGRUNDPROGRAMM	
		43	CTC-PROGRAMMIERUNG	
		44	ANZEIGEKONVERTIERUNG	
4040		45	ORG 4040H	
4040	3E41	46	UHRPRO LD A, 41H	
4042	ED47	47	LD I, A ; I-REG. LADEN	
4044	ED5E	48	IM2	
4046	3E00	49	LD A, 0 ; UBR.-VEKTOR FUER CTC	
4048	D388	50	OUT 38H ; IN KANAL 0 LADEN	
404A	3EA7	51	LD A, 0A7H ; KANALSTEUERWORT	
404C	D389	52	OUT 89H	
404E	3E24	53	LD A, 24H ; ZEITKONST. 36	
4050	D389	54	OUT 89H	
4052	FB	55	EI	
		56	ANZEIGESCHLEIFE	
		57	STAEENDIGE AKTUALISIERUNG DER ANZEIGE	
4053	21C340	58	ANZE LD HL, STUN	

ADR	OBJ-KODE	NR	POLY.UHR QUELLANWEISUNG	SEITE 2 POLY-880 ASM 1.0
4056	AF	59	XOR A	
4057	012041	60	LD AC, ANZBER	
405A	02	61	LD (BC), A	
405B	03	62	INC BC	
405C	02	63	LD (BC), A ; ERSTE 2 ANZEIGEN LOESCHEN	
405D	03	64	INC BC	
405E	7E	65	LD A, (HL) ; STUNDEN	
405F	CD8040	66	CALL ANZKON	
4062	2B	67	DEC HL	
4063	7E	68	LD A, (HL) ; MINUTEN	
4064	CD8040	69	CALL ANZKON	
4067	2B	70	DEC HL	
4068	7E	71	LD A, (HL) ; SEKUNDEN	
4069	CD8040	72	CALL ANZKON	
406C	112041	73	LD DE, ANZBER	
406F	CD4E01	74	CALL KONSOL ; ANZEIGE ANSTEUERN	
4072	18DF	75	JP ANZE	
		76		
		77	UNTERPROGRAMM ZUM KONVERTIEREN	
		78	EINES BYTES IN DIE 7-SEG.-ANZEIGE	
		79	EINGABE : ANZUZEIGENDES BYTE IN A	
		80	AUSGABE : ANZGESTELLEN (BC), (BC+1)	
		81	BC := BC + 2	
		82	VERAENDERTE REGISTER: AF, BC	
		83	ORG 4080H	
4080	E5	84	ANZKON PUSH HL	
4081	D5	85	PUSH DE	
4082	111003	86	LD DE, ANZDEC ; 7-SEG.-TABELLE	
4085	F5	87	PUSH AF	
4086	0F	88	RRCA	
4087	0F	89	RRCA	
4088	0F	90	RRCA	
4089	0F	91	RRCA ; HUEHERWERT. HALBBYTE RECHTSBUENDIG	
408A	E60F	92	AND 0FH	
408C	3600	93	LD H, 0	
408E	0F	94	LD I, A ; HL-VERSATZ FUER TABELLE	
408F	19	95	ADD HL, DE	
4090	7E	96	LD A, (HL) ; 7-SEG.-KODE	
4091	06F7	97	AND 0F7H ; PUNKT LOESCHEN	
4093	02	98	LD (BC), A ; IN ANZEIGEBEREICH	
4094	03	99	INC BC	
4095	F1	100	POP AF	
4096	E60F	101	AND 0FH ; NIEDERWERT. HALBBYTE	
4098	3600	102	LD H, 0	
409A	0F	103	LD I, A	
409B	19	104	ADD HL, DE	
409C	7E	105	LD A, (HL)	
409D	02	106	LD (BC), A	
409E	02	107	INC BC	
409F	01	108	POP DE	
40A0	E1	109	POP HL	
40A1	19	110	RET	
		111		
		112	SPEILHERSTELLEN	
40C0		113	ORG 40C0H	
40C0	64	114	HSEK DB 0	
40C1	00	115	SEF DB 0	
40C2	00	116	MIN DB 0	

Bild 8.11. Programmliste POLY.UHR

ADR	OBJ-KODE	NR	POLY.UHR QUELLANWEISUNG	SEITE 3 POLY-880 ASM 1.8
40C3	00	117	STUN DB 0	
		118		
		119	TABELLE DER UNTERBRECHUNGSPROGRAMME	
4100		120	ORG 4100	
4100	0000	121	UBRTAB DA 0	
4102	0040	122	DA 4000H ; BENUTZTER KANAL 1	
4104	0000	123	DA 0	
4106	0000	124	DA 0	
		125		
		126	BEREICH FUER ANZEIGEBELEGUNG	
		127	ANZBER EQU 4120H ; ANZEIGEBEREICH IM RAM	
		128		
		129	BENUTZTE PROGRAMME UND BEREICHE	
		130	DES MONITORPROGRAMMS	
		131	KONSOL EQU 14EH ; ANZEIGEANSTEUERPROG.	
		132	ANZDEC EQU 310H ; 7-SEG.-KODETABELLE	
		133		

### 8.9. Monitorunterstützung für Ein-/Ausgaben im POLY-COMPUTER

#### 8.9.1. Belegte E/A-Kanäle

Genauso wie man die Aufteilung des Speicherbereiches eines Rechners bei der Maschinenprogrammierung kennen muß, werden auch Informationen über die Lage der E/A-Adressen benötigt. Der E/A-Adressbereich im POLY-COMPUTER ist wie folgt eingeteilt:

00-7FH - Für Anwendererweiterungen am Systembus freigehalten

80H-8FH- System-PIO, sollte nur auf dem Weg über Unterprogramme des Monitors benutzt werden.

Ausgaben auf diese Adressen stellen die ordnungsgemäße Funktion des Monitors in Frage!

84H-87H- Anwender-PIO; Herausgeführt am Peripheriesteckverbinder (siehe Bedienhandbuch).

84H - Kanal A-Daten

85H - Kanal-A-Steuerworte

86H - Kanal B-Daten

87H - Kanal B-Steuerworte

88H-8BH- Zähler-/Zeitgeber-Schaltkreis CTC

88H - Kanal 0 - bereits vom Monitor belegt

89H - Kanal 1

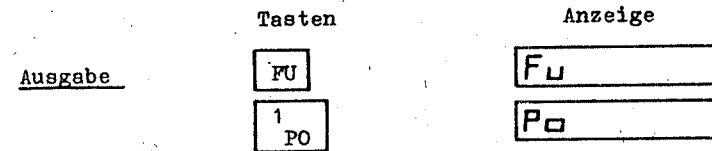
8AH - Kanal 2 } zur Verfügung des Anwenders

8BH - Kanal 3 }

8CH-0FFH - nicht benutzen

### 8.9.2. Zugriff zu E/A-Kanälen

Ebenso wie wir mit Hilfe des Monitorprogramms Speicherinhalte anzeigen und ändern können, besteht mitunter das Bedürfnis, von bestimmten Kanaladressen zu lesen oder dorthin Daten zu schreiben, ohne daß erst ein E/A-Befehl in den Speicher geschrieben und abgearbeitet werden muß. Diese Möglichkeit wird durch die Funktionen Ausgabe - PO (Port Out)- und Eingabe - PI (Port In)-realisiert. Aufgerufen werden diese Funktionen wie folgt:



Kanaladresse hexadezimal eingeben, z.B. 89H

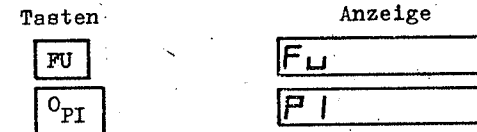


auszugebendes Datenbyte eingeben, z.B. A7H

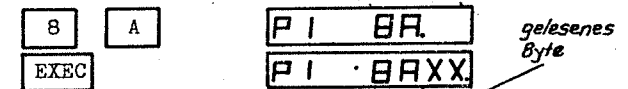


Mit dem Verlöschen des Datenbytes wird die erfolgte Ausgabe signalisiert, es können jetzt weitere Datenbytes auf die angezeigte Kanaladresse ausgegeben werden oder es kann zu einer anderen Funktion übergegangen werden.

#### Eingabe



Kanaladresse hexadezimal eingeben, z.B. 8AH



Bei jeder weiteren Betätigung der Taste  wird der Kanal erneut gelesen.

*gelesenes Byte*

### 8.9.3. Der Magnetbandanschluß des POLY-COMPUTERS

Mit zunehmendem Umfang Ihrer Programme werden Sie es sicher als ungünstig empfinden, nach jedem Einschalten des Gerätes die Programme neu eingeben zu müssen. Dies ist aber auch gar nicht notwendig, wenn Sie ein Magnetbandgerät an den POLY-COMPUTER anschließen. Das kann ein beliebiges Konsumgütergerät sein, vorzugsweise ein einfacher Kassettenrecorder (z.B. "Mira"). Dieses Gerät wird über ein meist bereits dazugehöriges Diodenkabel an den POLY-COMPUTER wie z.B. an einem Rundfunkempfänger angeschlossen. Die Diodenbuchse befindet sich auf der linken Seite des Rechners. Für das richtige Wiederauffinden auf dem Band müssen Sie selbst sorgen (z.B. Bandzählwerk notieren, Kommentar aufsprechen...). Die Ausgabe und das Einlesen von Speicherbereichen wird vom Monitor unterstützt, folgender Bedienablauf ist zweckmäßig:

#### Magnetbandschreiben

Taste	Anzeige
<input type="button" value="FU"/>	<input type="text" value="Fu"/>
<input type="button" value="5MO"/>	<input type="text" value="NO"/>
(MO-magnetic tape output)	
Anfangsadresse des auszugebenden Speicherbereiches angeben:	
z.B. 4000H	
<input type="text" value="4"/> <input type="text" value="0"/> <input type="text" value="0"/> <input type="text" value="0"/>	<input type="text" value="NO4000."/>
<input type="button" value="EXEC"/>	<input type="text" value="EA"/>
Endadresse des auszugebenden Speicherbereiches angeben:	
z.B. 4100H	
<input type="text" value="4"/> <input type="text" value="1"/> <input type="text" value="0"/> <input type="text" value="0"/>	<input type="text" value="EA4100."/>
<input type="button" value="EXEC"/>	<input type="text" value="READY ?"/>
Es wird also angefragt, ob das Magnetbandgerät bereit ist (ready ?)	
Jetzt ist Zeit, das Magnetbandgerät auf Aufnahme zu schalten, bei der nächsten Tastenbetätigung muß das Band laufen, denn dann beginnt die Aufzeichnung	
<input type="button" value="EXEC"/>	<input type="text" value=""/>

Jetzt erfolgt die Aufzeichnung, ihre Beendigung wird angezeigt durch

Die Ausgabe des gesamten RAM-Bereiches (4000 - 43FF) dauert etwa 10s.

Der gleichmäßige Ton am Anfang (Kennton) gehört noch nicht zu den Daten.

#### Magnetbandlesen

Taste	Anzeige
<input type="button" value="FU"/>	<input type="text" value="Fu"/>
<input type="button" value="4MI"/>	<input type="text" value="MI"/>
(MI - magnetic tape input)	
Anfangsadresse beim Einlesen z.B. 4000H	
<input type="text" value="4"/> <input type="text" value="0"/> <input type="text" value="0"/> <input type="text" value="0"/>	<input type="text" value="MI4000."/>
<input type="button" value="EXEC"/>	<input type="text" value="EA"/>
Endadresse beim Einlesen z.B. 4100H	
<input type="text" value="4"/> <input type="text" value="1"/> <input type="text" value="0"/> <input type="text" value="0"/>	<input type="text" value="EA4100."/>
<input type="button" value="EXEC"/>	<input type="text" value="READY ?"/>

Es wird wieder angefragt, ob das Magnetbandgerät bereit ist (ready ?)

Jetzt sollte das Magnetbandgerät kurz vor der Aufzeichnung oder auf dem Kennton stehen.

Jetzt ist das Magnetbandgerät auf Wiedergabe zu stellen, durch die genannte Bedienreihenfolge wird gesichert, daß der Rechner beim Einlaufen der ersten Daten auch wirklich bereits liest.

Der ordnungsgemäße Abschluß des Lesens wird angezeigt durch

Wenn Lesefehler eingetreten sind, erscheint die Anzeige

**E-XXXX**

Anfangsadresse eines fehlerhaften Blocks

Von der Anfangsadresse an können in den nächsten 32 Byte fehlerhafte Daten sein. Mit Betätigen von **EXEC** werden die Anfangsadressen weiterer fehlerhafter Blöcke angezeigt. Lesefehler treten praktisch nur infolge von Bedienungsfehlern (zu niedriger Aufzeichnungs-/Wiedergabepegel) oder sehr schlechtem Bandmaterial (meist hörbare Aussetzer/Höhenverluste) auf. Wenn keine Fertigmeldung erfolgt, dann wurde versucht, einen größeren Speicherbereich zu lesen als aufgezeichnet worden war.

#### 8.9.4. Der Fernschreiberanschluß des POLY-COMPUTERS

Neben dem Magnetbandanschluß ist ein Anschluß für eine Fernschreibmaschine (übliche Postfernschreiber, 50 Baud, 5-Bit-Kode) Die Anschlußschaltung einschließlich Speisung (40 mA-Stromschleife) ist bereits im POLY-COMPUTER enthalten. Die Programme zur Bedienung dieses Anschlusses sowie technische Einzelheiten sind im Bedienhandbuch zu finden. Der Fernschreiber kann für alphanumerische Eingaben oder Ausgaben genutzt werden, Hauptanwendung ist sicherlich die Herstellung von Listen bzw. Protokollen.

#### 8.10. Zusammenfassung

Die Ein- und Ausgabetechnik spielt bei der Mikrorechneranwendung eine Schlüsselrolle und nimmt deshalb im Arbeitsbuch breiten Raum ein. Die dargelegten Fakten und Zusammenhänge bilden lediglich eine erste notwendige Etappe auf dem Weg zur Beherrschung dieser Problematik. Vor allem die Ausführungen zur Unterbrechungssteuerung sollten aufmerksam studiert und die Beispiele nachvollzogen werden. Leider müssen sich diese Beispiele im Arbeitsbuch Teil I auf Operationen ohne Zusatzhardware (Prozeßnachbildungen o.ä.) beschränken, um für den gesamten Nutzerkreis des POLY-COMPUTER-Grundgerätes ausführbar zu sein. Weiterführende, hardwarebezogene Experimente (z.B. mit dem EXPERIMENTIER-MODUL) sind dem Arbeitsbuch Teil II zu entnehmen.

Fragen und Aufgaben:

1. Welche Steuersignale der CPU sind während E/A-Befehlenaktiv?
2. Wieviele E/A-Geräte kann die CPU U880 adressieren?  
Auf welchen Bits des Adreßbusses ist die E/A-Adresse enthalten?
3. Was ist ein Eingabeter und wozu wird es benötigt?
4. Nennen und beschreiben Sie mindestens drei Ein-/Ausgabeverfahren, die bei Mikrorechnern angewendet werden!
5. Welche vier Unterbrechungsbetriebsarten besitzt die CPU U880 und wie arbeiten sie?
6. Was zeigt das Interrupt-Flip-Flop IFF<sub>1</sub> an und welchen Zustand nimmt es nach einer Unterbrechungsunnahme an?
7. Welche Aufgabe hat das I-Register der CPU?
8. Welche Betriebsart der PIO eignet sich besonders zur Eingabe und Abtastung von Signalen eines Lochbandlesers?
9. Welche maximale Zeitdauer (und welche minimale) ist bei Anwendung eines CTC-Kanals als Zeitgeber bei der gegebenen Taktfrequenz im POLY-COMPUTER zwischen zwei ZC/TO-Impulsen erreichbar?
10. Beziehen Sie in Ihr Laufschrift-Programm aus Kapitel 6 den CTC als Zeitgeber ein, indem Sie ein Programm verfassen und testen, das den CTC aller 0,125 s zu einer Unterbrechungsanforderung veranlaßt, in deren Behandlung die Zeichen des Displays um eine Stelle nach links verschoben werden!
11. Erläutern Sie die Arbeitsweise der Prioritätskette!
12. Ergänzen Sie im Beispiel POLY-UHR die Anzeige der 1/10 und 1/100 Sekunden!
13. Verändern Sie das Programm POLY-UHR so, daß eine Stoppuhrfunktion möglich ist, die über eine beliebige Taste (Monitorprogramm KONSOL!) gestartet bzw. gestoppt werden kann!

9. Binäre und dezimale Arithmetik

Im Gegensatz zu Anlagen der elektronischen Datenverarbeitung ist der Mikrorechner nicht für die Lösung umfangreicher mathematischer Probleme entwickelt worden und ist demzufolge dafür in seiner gegenwärtigen Form auch nicht sonderlich gut geeignet. Sein Einsatzgebiet ist vor allem die Prozeßsteuerung. Trotzdem kommt der Anwender oft nicht umhin, ausgewählte mathematische Operationen (z.B. Mittelwertbildung einer Reihe von Meßwerten) durchzuführen, so daß dieses Kapitel seine Daseinsberechtigung im Arbeitsbuch erhält.

Das gesamte Gebiet der numerischen Mathematik, d.h. der Methoden zur Lösung mathematischer Problemstellungen mittels Digitalrechner, ist allerdings derart umfangreich, daß wirklich nur einige Ansätze und wichtigste Grundlagen an dieser Stelle vermittelt werden können und der interessierte Leser auf die einschlägige Literatur verwiesen werden muß.

Bisher verwendeten wir in unseren Beispielen binäre Daten verschiedenen Umfangs, die wir entweder als Binärzahl (z.B. beim Zähler von Zyklusdurchläufen) oder einfach als binäre Schalter (AUS - EIN) interpretiert haben. Mathematische Aufgaben stellen aber häufig Forderungen, die mit der einfachen Interpretation der Daten als Binärwert nicht mehr lösbar sind.

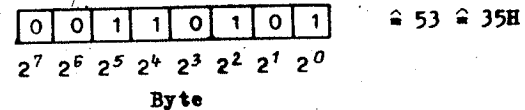
Folgende wichtigste Randbedingungen werden gestellt:

- bestimmte Rechengenauigkeit (z.B.  $10^{-5}$ )
- Zahlenbereich (z.B.  $-10^{99} < x \leq 10^{99}$  z.B. für wissenschaftliche Taschenrechner üblich)
- eindeutige Darstellbarkeit positiver und negativer Zahlen
- bestimmte maximale Rechendauer.

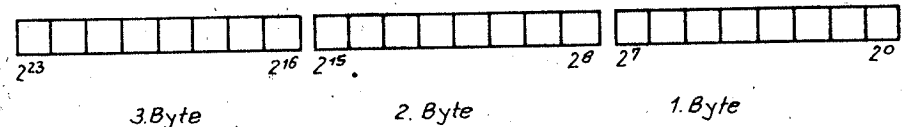
Die für den jeweiligen Anwendungsfall sinnvolle Zahlendarstellung ist dem Anwender selbst überlassen. Einleuchtend dürfte sein, daß mit zunehmender Rechengenauigkeit die Rechenzeit ebenfalls zunimmt, und zwar meist nicht linear sondern exponentiell.

9.1. Darstellung und Operationen natürlicher Zahlen

Natürliche Zahlen (0, 1, 2...; d.h. ganze positive Zahlen) werden als Binärzahlen rechnerintern dargestellt, d.h. die Bits entsprechen in ihrem Wert von rechts nach links aufsteigenden Zweierpotenzen. In einem Byte (8Bits) ist demnach als größte Zahl  $2^8 - 1 = 255$  und als kleinste Zahl die Null darstellbar.



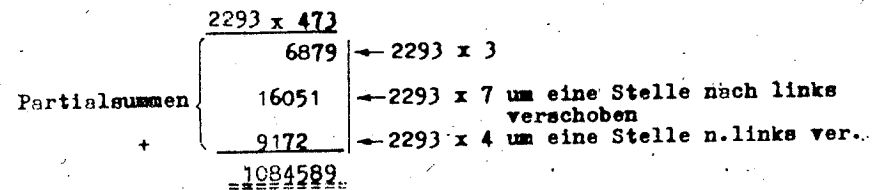
Soll der Zahlenbereich erweitert werden (größere Zahlen sollen verarbeitet werden), so sind sinnvollerweise mehrere Bytes zu der gewünschten Länge aneinandersureihen.



Mit drei Bytes könnte als größte Zahl  $2^{24} - 1 = 16777215$  verarbeitet werden. Allerdings läßt sich dieses Verfahren zur Zahlenbereichserweiterung nicht andlos fortsetzen. Z.B. wären zur Erweiterung auf den Zahlenbereich wissenschaftlicher Rechner ( $10^{99}$ ) bereits mehr als 40 Bytes (!) zur Darstellung einer Zahl erforderlich.

Beispiele zur Addition, Subtraktion und Multiplikation von Binärzahlen sind z.B. im Kapitel 4 enthalten. Neben der Möglichkeit zur Nachbildung der Multiplikation  $m \times n$  durch n-fache Addition von  $m$  kann selbstverständlich auch in einem Rechner das wesentlich kürzere Verfahren der Partialsummenbildung und stellungerechte Addition Verwendung finden, das uns bei Dezimalrechnungen geläufig ist.

Im Dezimalsystem würde folgendes ausgeführt:

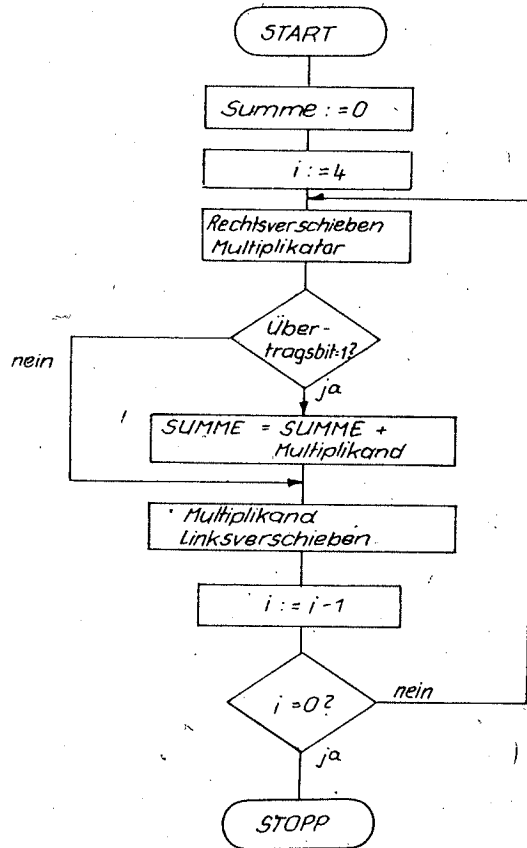


Auf analoge Weise kann die Multiplikation von Binärzahlen ausgeführt werden.

$$\begin{array}{r}
 1101 \times 1010 \quad (13 \times 10 = 130) \\
 \hline
 0000 \quad \leftarrow 1101 \times 0 \\
 1101 \quad \leftarrow 1101 \times 1 \text{ um eine Stelle nach links} \\
 0000 \quad \leftarrow 1101 \times 0 \text{ um zwei Stellen nach links} \\
 + 1101 \quad \leftarrow 1101 \times 1 \text{ um drei Stellen nach links} \\
 \hline
 10000010
 \end{array}$$

Die Multiplikationsoperation mit einer Binärstelle liefert entweder den Multiplikanden selbst oder den Wert Null.

Der Programmablaufplan für ein Multiplikationsprogramm nach diesem Verfahren könnte folgendermaßen aussehen (Binärzahlen 4-stellig):



Als Programm ergibt sich:

Adresse (hex)	Befehlskode (hex)	Mnemonic	Kommentar
4000	AF	MULT: XOR A	; A:=0, in A steht Summe ; B enthalte Multiplikand ; C enthalte Multiplikator
4001	16,04	LD D,4	; D:=4, D enthält Stellenzähler
4003	CB,09	M1: RRC C	; CY:= niederwertiges Multiplikatorbit
4005	E2,09,40	JNC M2	
4008	80	ADD B	; SUMME:=SUMME+Multiplikand
4009	CB,20	M2: SLA B	; Linksverschieben Multiplik.
400B	15	DEC D	; D:=D-1
400C	C2,03,40	JNZ M1	; D=0?
400F	C3,0F,40	M3: JNP M3	; Stoppschleife

Das Programm kann mit dem POLY-COMPUTER getestet werden. In B und C wird die 4 Bit-Faktoren eingegeben, A enthält das Produkt. Steht in B eine Zahl größer 15 (4 Bits), so könnte bei einer Partialsummenbildung ein Übertrag entstehen, der in diesem Programm nicht getestet wird. Somit würde ein falsches Ergebnis auftreten.

### 9.2. Ganze vorseichenbehaftete Zahlen

In Rechnern kann zwischen einer positiven und einer negativen Zahl z.B. durch den Zustand eines bestimmten zusätzlichen Bits unterschieden werden. Meist wird dazu das höchstwertigste Bit der Zahl verwendet. Bei Einbyteszahlen ist das also Bit 7, das für diesen Zweck reserviert wird.



S - Vorseichenbit (Signum) 1-negative Zahl  
0-positive Zahl

Der Zustand dieses 7. Bits wird nach arithmetischen Operationen in das S-Flag (Vorseichenflag) übernommen.

Für den Zahlenwert stehen jetzt nur noch sieben Bits zur Verfügung, so daß der Zahlenbereich z.B. zwischen -128 (-127) und +127 liegt.

Die Darstellung des Zahlenwertes kann auf sehr unterschiedliche Weise erfolgen. Wir wollen die drei wichtigsten Arten kurz betrachten.

(1) Vorzeichen-Betrag

S	Betrag	S=Vorzeichen: 0 - positiv 1 - negativ
---	--------	--

Der Betrag entspricht einer vorzeichenlosen Binärzahl mit sieben Bits-Länge.

Beispiel:

01111111	= +127
00000001	= +1
00000000	= 0 (+0)
10000000	= 0 (-0)
10000001	= -1
11111111	= -127

Unvorteilhaft ist die mehrdeutige Repräsentation der Null, was besonders bei Tests den Aufwand erhöht.

(2) Einerkomplement

Positive Zahlen werden wie bei der Vorzeichen-Betrags-Darstellung gebildet (Bit 7=0, d.h. größte darstellbare Zahl in einem Byte ist die 127). Negative Zahlen erhalten wir durch bitweise Negation der positiven Zahl gleichen Absolutwertes.

Beispiel:

+15:	00001111
-15:	11110000

Auch bei dieser Darstellung fungiert Bit 7 als Vorzeichenbit.

Weitere Beispiele:

01111111	= +127	größte Zahl (in einem Byte!)
00000001	= +1	
00000000	= (+) 0	
10000000	= (-) 0	
11111110	= -1	
10000000	= -127	kleinste Zahl (in einem Byte)

Auch hier tritt die unangenehme Doppeldeutigkeit bei der Null-darstellung auf.

(3) Zweierkomplement

Die positiven Zahlen entsprechen den Darstellungen (1) und (2). Negative Zahlen werden gebildet, indem zur Einerkomplementdarstellung eine binäre 1 addiert wird.

Beispiel:

+15:	00001111
Einerkomplement:	11110000
-15:	<u>11110001</u>

Bit 7 fungiert wieder als Vorzeichenbit

Beispiele:

01111111	= +127	größte Zahl (in einem Byte)
00000001	= +1	
00000000	= 0	
11111111	= -1	
10000000	= -128	kleinste Zahl (in einem Byte)

Diese Darstellung überwindet die Doppeldeutigkeit der Null-darstellung und wird u.a. auch deshalb überwiegend angewendet.

Einige Befehle der CPU U880 verwenden diese Darstellung für die relative Adressierung (siehe Kapitel 10).

Die Umwandlung einer negativen, im Zweierkomplement dargestellten Zahl in eine positive Zahl gleichem Betrages erfolgt auf die gleiche Weise wie ihre Bildung, d.h. durch bitweise Negation und Addition einer 1.

Beispiel:

-5:	11111011
Einerkomplement	00000100
+5:	<u>00000101</u>

Die Zweierkomplementdarstellung und -interpretation liefert als einzige stets richtige Ergebnisse bei der Anwendung der Befehle zur Binäraddition und -subtraktion, solange die Zahlen den zulässigen Wertebereich nicht überschreiten.

Beispiel:

3	00000011
+(-5)	+ 11111011
-2	<u>11111110</u>

Demit ist es sogar möglich, die Subtraktionsoperation durch Addition des Komplementes zu ersetzen.

Sollten beispielsweise in einem Rechner negative Zahlen eingegeben und verarbeitet werden, so geschieht dies meist durch Eingabe des Vorzeichens und des Betrages. Anschließend ist die Zahl im Rechner zu komplementieren. Dazu eignet sich bei Einsatz der CPU U880 der bereits vorgestellte Befehl NEG, der den Inhalt des A-Registers in die Zweierkomplementdarstellung überführt.

9.3. Zahlenbereichsüberschreitungen beim Rechnen mit ganzen Zahlen und Zweierkomplementdarstellung

Wir wollen uns in diesem Abschnitt auf die Verarbeitung von Zahlen beschränken, die in einem Byte darstellbar sind. Mehrbyte-darstellungen werden analog behandelt, wobei dann das höchstwertigste Bit des höchstwertigsten Bytes als Vorzeichen zu interpretieren ist.

Die Operation

```

+100   01100100
+ 30   + 00011110
-126   -10000010
    
```

Überschreitet den Zahlenbereich (+127 bis -128) und liefert demzufolge ein falsches Ergebnis, wenn wir mit vorzeichenbehafteten Zahlen arbeiten. Die CPU U880 besitzt drei Flags, die bei Beurteilung des Ergebnisses arithmetischer Operationen besondere Bedeutung besitzen (C, S, P/V).

Das C (Carry-Übertrag) - Flag wird bei Auftreten eines Übertrages vom Bit 7 in das (nicht vorhandene Bit 8) auf 1 gesetzt, ansonsten auf 0 rückgesetzt.

Das S (Sign-Vorzeichen) - Flag ist lediglich ein Abbild des Bit 7 des A-Registers nach arithmetischen und logischen Operationen. Für Operationen mit vorzeichenbehafteten Zahlen (Bit 7 ist Vorzeichenbit) existiert das mehrfach genutzte P/V (Parity/Overflow) -Flag.

Bei logischen Befehlen wird es als Paritätsflag verwendet (=1, wenn Anzahl der Einsen im Zielregister gerade, sonst = 0), bei arithmetischen Befehlen ist es als Indikator für Überläufe von oder nach Bit 7 (Vorzeichenbit) eingesetzt.

Im letzten Beispiel (100+30) wäre demnach folgender Flagzustand zu erwarten:

```

C = 0   kein Übertrag
S = 1   Vorzeichenbit ist 1
P/V = 1   es trat Überlauf von Bit 6
           nach Bit 7 auf.
    
```

Dieser Flagzustand (0,1,1) ist demnach ein Zeichen für eine Zahlenbereichsüberschreitung. Bei der Addition und Subtraktion von Zahlen in Zweierkomplementdarstellung ist der Zustand dieser drei Flags allgemein so zu interpretieren:

Operation	C	S	P/V	Ergebnis
positive Zahl $\pm$ pos.Zahl	0	0	0	positive Zahl
positive Zahl + pos.Zahl	0	1	1	Überlauf
positive Zahl - pos.Zahl	1	1	0	negative Zahl
negative Zahl $\pm$ neg.Zahl	1	1	0	negative Zahl
negative Zahl + neg.Zahl	1	0	1	Überlauf
negative Zahl - neg.Zahl	0	0	0	positive Zahl
positive Zahl $\pm$ neg.Zahl	1	0	0	positive Zahl
positive Zahl + neg.Zahl	0	1	0	negative Zahl
positive Zahl - neg.Zahl	1	1	1	Überlauf
negative Zahl $\pm$ pos.Zahl	0	1	0	negative Zahl
negative Zahl + pos.Zahl	1	0	0	positive Zahl
negative Zahl - pos.Zahl	0	1	1	Überlauf

Die vier Überlauf signalisierenden Flagzustände müssen zu einer Fehlerbehandlung entsprechend der verwendeten Operandenvorzeichen führen.

**Achtung!** Einige der arithmetischen Befehle (z.B. Zweibyteaddition ADD HL, ss ; ss-Registerpaar) beeinflussen die Flags in anderer bzw. keiner Weise, so daß deren Auswertung keinen Aufschluß über das Ergebnis der Operation bringt. Deshalb ist bei der Anwendung der Befehle stets genau deren Wirkung auf die Flags zu beachten (siehe Befehlstabelle des Systemhandbuchs)!)

9.4. Festkommazahlen (fixed point numbers)

Für Rechnungen mit gebrochenen Zahlen ist das Festkommazahlenformat anwendbar. Ähnlich wie bei der Darstellung von Dezimalbrüchen

z.B.  $31,571 = 30 \cdot 10^1 + 1 \cdot 10^0 + 5 \cdot 10^{-1} + 7 \cdot 10^{-2} + 1 \cdot 10^{-3}$

lassen sich reelle Zahlen als Binärbruch aufschreiben,

z.B.  $101,101 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$   
 $= 5,625$  (decimal)

Im Rechner kann sich der Programmierer die Bitanzahl und die Lage des Kommas vorgeben, z.B. 8 Bits und das Komma nach Bit 3.



gedachtes Komma



Für negative Zahlen kann auch hier z.B. die Zweierkomplementdarstellung Verwendung finden.

Zwei Festkomma (FK)-zahlen werden z.B. addiert, indem sie stellengerecht positioniert und addiert werden.

$$\begin{array}{r|l} \text{Beispiel: } 101,101+11,1 = 101,101 & 5,625 \\ +011,100 & +3,5 \\ \hline 1001,001 & 9,125 \end{array}$$

Aus Gründen der Einfachheit wird meist ein FK-Format gewählt, bei dem sich das Komma ganz links vor der höchsten Stelle befindet, so daß nur Zahlen zwischen 0 bzw. -1 und 0,99... (je nach Stellenzahl) auftreten können. Zu verarbeitende reale Zahlenwerte müssen demnach vor der Verarbeitung auf den zu erwartenden Größtwert normiert werden, d.h. alle Zahlen werden vor der Verarbeitung durch diesen Wert dividiert, so daß nur Zahlen in den genannten Bereich auftreten.

Bei 8 Bit-Darstellungen positiver FK-Zahlen



ergibt sich somit als größte darstellbare Zahl

$$1 \cdot 2^{-1} + 1 \cdot 2^{-2} + \dots + 1 \cdot 2^{-8} = 0,99609375$$

Der Abstand zwischen zwei benachbarten Zahlen beträgt

$$2^{-8} = 0,00390625.$$

Aus diesem Wert kann der zu erwartende Quantisierungsfehler (d.h. der Fehler, der durch das Fehlen der Zwischenwerte auftritt) abgeschätzt werden.

Übliche Festkommaprogramme arbeiten mit 24 bzw. 32 Bits Zahlenlänge.

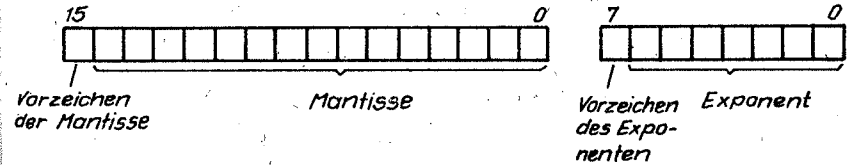
### 9.5. Gleitkommazahlen (floating point numbers)

Der bisher wenig befriedigende, mit vertretbarem Aufwand beherrschbare, sehr geringe Zahlenbereich wird mit Hilfe der Gleitkomma (GK)-darstellung wesentlich vergrößert. Sie besteht in der konsequenten Umsetzung der Potenzschreibweise in die rechnerinternen Möglichkeiten.

Eine in Potenzschreibweise dargestellte Zahl besteht aus zwei Komponenten, nämlich aus Mantisse und Exponent.

$$\text{Beispiel: } \underbrace{5,788}_{\text{Mantisse}} \cdot 10^{\underbrace{-6}_{\text{Exponent}}}$$

Die Größe der Mantisse bestimmt die Genauigkeit der Rechnungen, die Größe des Exponenten dagegen den Zahlenbereich. Die Basis des Exponenten ist fest vereinbart und muß nicht mit abgespeichert und verarbeitet werden. Für das Gleitkommaformat existiert eine Reihe von verwendeten konkreten Formaten. Ein gebräuchliches Format ist 15 Bit Mantisse und 7 Bit Exponent.



so daß pro Zahl drei Bytes benötigt werden. Unter Verwendung der Zweierkomplementdarstellung ergibt sich ein Zahlenbereich von

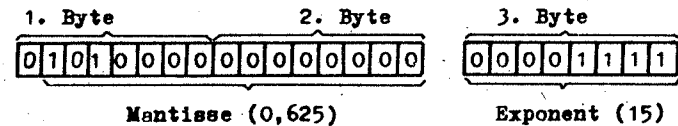
$$- 32768 \cdot \text{Basis}^{127} \text{ bis } 32767 \cdot \text{Basis}^{127}$$

Die betragsmäßig kleinste von Null verschiedene Zahl (Elementar-einheit) ist  $\text{Basis}^{-128}$ ,

wobei Basis meist 2 oder 10 ist und die Stellung des Kommas innerhalb der Mantisse festgelegt werden muß. Häufig wird ähnlich dem empfohlenem FK-Format verfahren und das Komma links vor der Mantisse angenommen.

$$\text{Beispiel: } 0,625 \cdot 2^{15}$$

Diese Zahl hätte im Rechner folgende gestalt:



Operationen im Gleitkomaformat sind entsprechend den Regeln der Potenzrechnung vorzunehmen.

Die GK-Darstellung ist universell und an jede Problemstellung anpaßbar. Allerdings sind die dazu erforderlichen Programme recht komplex. Neben den eigentlichen Operationsprogrammen (für Addition, Multiplikation usw.) umfassen sie eine Reihe von Konvertierungsprogrammen, Fehlerbehandlungen, Normalisierungsvorschriften u. a. m. Deshalb werden meist Standard-Mathematikprogramme vom Hersteller der Prozessoren oder Rechner geliefert, die vom Anwender genutzt werden sollten.

### 9.6. BCD-Darstellung von Zahlen

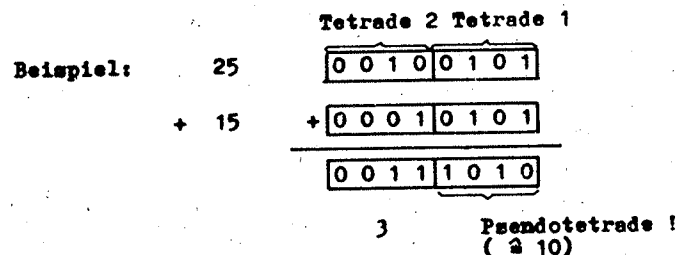
Häufig liegen die zu verarbeitenden Zahlenwerte als Dezimalzahlen vor. Eine Konvertierung in die rechnerinterne Binärdarstellung wäre zur Nutzung der Binärrarithmetik-Befehle der CPU stets erforderlich, ebenso eine Rückkonvertierung der Ergebnisse bei Ausgabe auf Drucker, Display o.ä.

Diese recht aufwendige Konvertierung kann wenn erforderlich eingespart werden, indem die BCD (binary coded decimals = binär kodierte Dezimalzahl)-Darstellung verwendet wird. Dabei wird jede Dezimalziffer in vier Bits als Binärzahl kodiert



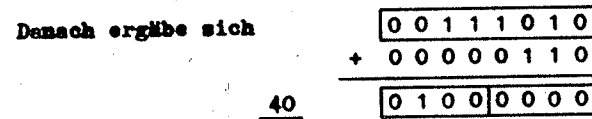
In einem Byte ist demnach eine zweistellige Dezimalzahl unterzubringen. Diese Darstellung wird gepackte Dezimalzahl genannt und am häufigsten angewendet. Die entsprechende "ungepackte" Dezimaldarstellung speichert nur eine Dezimalziffer pro Byte. Die vier eine Dezimalziffer darstellenden Bits werden als Tetrade bezeichnet. Neben sinnvollen Tetraden 0000 bis 1001 (0-9) existieren die Pseudotetraden 1010 bis 1111 (A-F), denen keine Dezimalziffer zugeordnet werden kann. Treten diese bei Rechnungen auf, müssen entsprechende Korrekturrechnungen vorgenommen werden

Die CPU U880 verfügt über keine speziellen Dezimalarithmetikbefehle, besitzt aber einen Korrekturbefehl, der die Anwendung der Befehle der Binärrarithmetik erlaubt und anschließend das Ergebnis korrigiert.



Das Ergebnis dieser binären Addition zweier gepackter Dezimalzahlen ist so noch nicht verarbeitbar und muß noch korrigiert werden.

Der Korrekturbefehl müßte u. a. beim Auftreten von Pseudotetraden eine zusätzliche Addition von 6 vornehmen, um den Bereich der Pseudotetraden zu überspringen (10 - 15).



das richtige Ergebnis. Der Korrekturbefehl ist der DAA-Befehl.

<b>Mnemonic:</b>	DAS	(decimal adjust accumulator - Dezimaljustierung des A-Registers)
<b>Befehlscode:</b> (hex)	27	
<b>Wirkung:</b>	Konvertiert den A-Registerinhalt in eine gepackte Dezimalzahl, <u>wenn vorher</u> eine Addition oder Subtraktion gepackter Dezimalzahlen vorgenommen wurde. In Auswertung der Flags N (gibt an, ob eine Addition oder Subtraktion vorausging), C und H (Übertragsflag von Bit 3 nach Bit 4 - d.h. zwischen den Dezimalsiffern) und des A-Registers werden folgende Operationen vorgenommen:	

nach	bei C vor DAA gleich	bei Tetrade 2 gleich	bei H vor DAA gleich	bei Tetrade 1 gleich	wird addiert (hex)	C nach DAA gleich
Addition (ADD, ADC, INC)	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-8	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	65	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
Subtraktion (SUB, SBC, DEC, NEG)	0	0-9	0	0-9	00	0
	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	A0	1
	1	6-F	1	6-F	9A	1

Unter der Voraussetzung, daß in den Registern A und B gepackte Dezimalzahlen enthalten sind, sieht eine vollständige Dezimaladdition dieser beiden Werte wie folgt aus:

ADD B  
DAS

Anschließend steht in A das Ergebnis als gepackte Dezimalzahl. Mit gepackten Dezimalzahlen sind natürlich all die genannten Darstellungsarten (FK-, GK-Darstellung) sinngemäß realisierbar. Zur Darstellung negativer Zahlen wird die Vorzeichen/Betragsdarstellung oder das Hunderterkomplement verwendet. Dabei wird z.B. -1 als 99, -2 als 98 usw. dargestellt. Zum Vorzeichenwechsel einer Dezimalzahl wird das niederwertigste Byte von 100 subtrahiert, alle übrigen von 99. In einem Byte ist somit ein Zahlenbereich von -19 bis +79 darstellbar

+79 0 1 1 1 1 0 0 1  
-19 1 0 0 0 0 0 0 1

Für sinnvolle Anwendungen ist stets mehr als ein Byte Zahlenlänge erforderlich.

### 9.7. Andere mathematische Operationen

Der Prozessor U880 besitzt nur für die elementaren mathematischen Operationen Addition und Subtraktion spezielle Befehle. Andere mathematische Operationen müssen auf diese unter Zuhilfenahme von Logikbefehlen zurückgeführt werden. Für die Multiplikation und Division wurden bereits Beispiele angegeben. Andere Standardfunktionen, die uns z.B. vom wissenschaftlichen Taschenrechner her geläufig sind (z.B. Winkelfunktionen sin, cos, Exponentialfunktionen  $e^x$ ,  $y^x$ ,  $\sqrt[x]{y}$ , Logarithmen usw.) werden entweder mit Hilfe von Reihenentwicklungen (relativ lange Rechenzeiten) oder über Speichertabellen mit Interpolation der Zwischenwerte (großer Speicherplatzbedarf) realisiert.

Z.B. kann die Funktion  $y = \sqrt{x}$  mit dem

$$\text{Iterationsverfahren } Y_{i+1} := \frac{1}{2} \left( Y_i + \frac{x}{Y_i} \right) \quad x_0 = x,$$

das recht schnell gegen  $\sqrt{x}$  konvergiert, gebildet werden.

Für  $e^x$  wird die Reihenentwicklung

$$e^x = \sum_{n=0}^k \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots + \frac{x^k}{k!}$$

angewendet.

9.8. Zusammenfassung

8 Bit-Mikrorechner sind als universelle, programmierbare Rechner für die Lösung auch anspruchsvoller mathematischer Probleme prinzipiell geeignet. Während für sehr viele Steuerungsaufgaben die Wortbreite und der Befehlssatz der CPU U880 v8111g ausreichen, werden mathematische Operationen in einem größeren Zahlenbereich, die auch noch möglichst schnell ausgeführt werden sollen, zu Problemfällen, die oft nur mit trickreicher Programmierweise bzw. erheblichem Verzicht auf Genauigkeit lösbar sind.

Ein ausführliches Beispiel zu diesem Kapitel findet der Leser im Kapitel 10.

Fragen und Aufgaben:

1. Realisieren Sie ein Programm zur Division von natürlichen Zahlen (8 Bits Länge), das analog dem Multiplikationsverfahren aus Abschnitt 9.1. arbeitet!
2. Auf welche der drei im Abschnitt 9.2. dargelegten Darstellungsformen für negative Zahlen trifft die Aussage 1 1 1 1 1 0 1 1  $\hat{=}$  -5 zu?  

1 1 0 1 0 0 0 1 ,
3. Welche Zustände nehmen die Flags C, S, P/V nach der Ausführung der Operation 70 + 35 an?
4. Welcher Dezimalzahl entspricht die Bitfolge  

1 1 0 1 0 0 0 1 ,

wenn es sich um eine Postkommaerstellung mit der angegebenen Kommastelle handelt?
5. Realisieren Sie ein Programm zur Addition von gepackten positiven Dezimalzahlen mit vier Stellen. Ein- bzw. Ausgabe ist über Tastatur bzw. Display des POLY-COMPUTERS vorzunehmen!

10. Ergänzungen und Beispiele

10.1. Die vollständige Befehlsliste

10.1.1. Überblick

Der Befehlssatz (Summe aller Befehle) läßt sich in acht Befehlsgruppen einteilen:

1) Lade- und Austauschbefehle

Sie besitzen die allgemeine Struktur

LD	ziel, quelle	bzw.
PUSH	operand	
POP	operand	

Auch die EX...-Befehle zählen dazu. Die Aufgabe dieser Befehle ist der Datentransport ohne Manipulation der Daten oder der Flags.

2) Blockbefehle für Transport und Vergleich

Im Unterschied zur Gruppe 1 wird mit einem Blockbefehl nicht nur ein Byte oder ein Wert, sondern ein Block von 1 bis 65536 Bytes transportiert bzw. Vorbereitungen dafür getroffen (siehe Abschnitt 10.1.3.). Ebenso kann ein Suchprozeß nach einem bestimmten Datenmuster durch einen Blockvergleichsbefehl (auf der Basis des Logikbefehls CP) vorgenommen werden (siehe ebenso Abschnitt 10.1.3.).

Allgemeine Struktur:	LDIR
	LDI
	LDDR
	LDD
	CPIR
	CPI
	CPDR
	CPIR

HL und DE enthalten Adressen für Ziel und Quelle der Operation, BC die Anzahl der Operationen.

3) Arithmetische und Logikbefehle

Arithmetische und Logikbefehle verknüpfen zwei Operanden, von denen sich einer im A-Register befinden muß (bis auf wenige Ausnahmen).

Als Einbyteoperationen sind möglich

- Addition (ADD, ADC)
- Subtraktion (SUB, SBC)
- Logisches UND (AND)
- Logisches ODER (OR)
- Exklusiv ODER (XOR)
- Vergleich (CP)
- Inkrementieren (INC)
- Dekrementieren (DEC)
- Dezimaljustierung (DAS) und Negation (NEG) und Komplementbildung (CPL).

Als Zwei byteoperationen sind

- Addition (ADD HL,...  
ADC HL,...  
ADD IX,...  
ADD IY,...)
- Subtraktion (SBC HL,...)
- Inkrementieren (INC)
- Dekrementieren (DEC)

möglich.

#### 4) Rotier- und Verschiebefehle

Es wird zwischen Rotation nur im A-Register

(RLCA, RLA, RRCA, RRA) und in allen Universalregistern und Speicherplätzen unterschieden (RLC, RL, RRC, RR). Verschiebungen sind auf dreierlei Art möglich (SLA, SRA, SRL). Für die BCD-Arithmetik existiert eine spezielle Rotieroperation (RLD).

#### 5) Bitbefehle

Beliebige Bits in Registern und Speicher können getestet (BIT), auf 1 gesetzt (SET) oder auf 0 rückgesetzt werden (RES)

#### 6) Programmverzweigebefehle

Mittels Sprungbefehlen kann bedingt (J bed. adresse) und unbedingt (JMP adresse) sowie absolut (JMP adresse) oder relativ zum augenblicklichen PC-Wert (JR verschiebung; DJNZ verschiebung) eine Programmverzweigung erfolgen (zur relativen Adressierung siehe Abschnitt 10.1.2.).

Bedingte Verzweigungen können jeweils in Abhängigkeit vom Zustand der Flags C, Z, S, und P erfolgen.

Unterprogrammaufrufe können ebenso bedingt (C bed. adresse) oder unbedingt (CALL adresse) erfolgen. Der Kurzuruf zu den absoluten Adressen 0, 8, 10H, 18H, 20H, 28H, 30H, 38H wird mittels RST-Befehlen vorgenommen. Die Rückkehr vom UP oder von der Unterbrechungsbehandlungsroutine zum aufrufenden bzw. unterbrochenen Programm wird mit RET, RETI bzw. RETN vorgenommen und kann ebenso wie der CALL-Befehl bedingt (in Abhängigkeit der Flagsustände) ausgeführt werden (R bed.).

#### 7) Eingabe- und Ausgabebefehle

Vom und zur Peripherie erfolgt der Datentransport mittels E/A-Befehlen, die entweder nur mit dem A-Register als Empfangs- bzw. Senderegister arbeiten (IN n; OUT n), oder aber sämtliche Universalregister akzeptieren (IN r; OUT r). Darüber hinaus können mit nur einem Befehl ebenfalls Blockübertragungen von 1 bis 256 Bytes veranlaßt werden (OUTI, OTIR, OUTD, OTDR - siehe dazu auch Abschnitt 10.1.3.).

#### 8) CPU-Steuerbefehle

Diese Befehlsgruppe realisiert verschiedene Aufgaben zur Unterbrechungsvorbereitung (IN 0...2; DI; EI) bzw. zur Steuerung des Programmablaufes (NOP; HALT).

Nach diesem Überblick sollte der Leser die Befehlsliste (Systemhandbuch) gründlich studieren und die Wirkungen der einzelnen Befehle bei Unklarheiten durch Ausführung mit dem POLY-COMPUTER verdeutlichen. Nur bei relativ tiefgründiger Kenntnis der verfügbaren Befehle ist auch die Gestaltung effektiver Programme möglich.

#### 10.1.2. Relative Adressierung

Vor allem bei Programmverzweigungen ist die relative Adressierung sinnvoll anwendbar. Im Gegensatz zur absoluten Adressierungsart (z.B. JMP 4000H bedeutet eine Verzweigung zur absoluten Adresse 4000H) enthält der Befehl keine absolute Zieladresse (4000H), sondern eine sogenannte Verschiebung. Diese Verschiebung (normalerweise ist das eine vorseichenbehaftete Zahl im Bereich -128 bis +127) wird zum Wert des Programmzählers PC hinzu addiert und die neu entstandene Adresse als Zieladresse für die Verzweigung verwendet. D.h. je nach Anordnung eines solchen Befehls im Speicher erfolgt die Verzweigung zu verschiedenen Adressen (siehe Bild 10.1.)

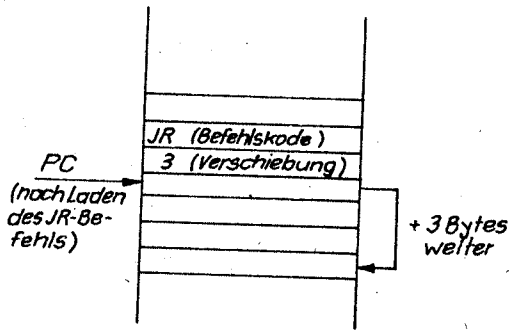


Bild 10.1. Wirkung des Relativsprungs JR +3

Mnemonic:	JR e	(jump relativ e-Verschiebung) 2-er Komplement
Befehlskode: (hex)	18 e	(-128 ≤ e ≤ +127)
Wirkung:	Versweigung zur Adresse PC+e, wobei PC auf den dem JR-Befehl folgenden Speicherplatz zeigt.	

Die Anwendung des JR-Befehls bringt gegenüber dem Absolutsprung JMP folgende Vorteile:

- Programm mit nur relativer Adressierung ist auf beliebigen Speicherplätzen ausführbar.
- Einsparung eines Befehlsbytes.

Als Nachteile sind zu nennen:

- für größere Sprungweiten (>128) nicht anwendbar;
- bei der manuellen Kodierung ist die Fehlerwahrscheinlichkeit höher (Abzählen der Sprungweite, Zweierkomplement bilden)
- Einschränkungen bei bedingten Sprüngen.

Aus diesen Erwägungen heraus ist die Anwendung für den "Handkodierer" wenig sinnvoll, während der Befehl für den mit rechen-technischen Hilfsmitteln übersetzenden Programmierer vorteilhaft-weiß-speicherplatzsparend-einsatzbar ist.

### 10.1.3. Blockbefehle

Der Prozessor U880 verfügt über eine Gruppe von Befehlen, die einige häufiger benötigte Programmroutinen durch einen Befehl ersetzen, die Blockbefehle. Sie können drei Aufgabengruppen bearbeiten:

- Transport eines Datenblockes innerhalb des Speichers
- Durchsuchen eines Datenblockes im Speicher nach einem festen Datenmuster
- Eingabe bzw. Ausgabe eines Datenblockes von bzw. zu der Peripherie.

Innerhalb jeder Gruppe sind zwei Befehlstypen zu unterscheiden:

- Befehle, die automatisch mehrfach wiederholt ausgeführt werden und
- Befehle, die lediglich die Parameter für eine wiederholte Ausführung vorbereiten, aber nur einmalig ausgeführt werden.

- Transport eines Datenblockes innerhalb des Speichers  
Der Transport der Daten eines adressmäßig zusammenhängenden Speicherbereiches auf einen anderen Bereich ist natürlich mit Hilfe der bereits bekannten Befehle bereits lösbar. Lediglich zur Programmvereinfachung (weniger Speicher, geringerer Zeitaufwand) können die Befehle LDIR, LDI, LDDR bzw. LDD eingesetzt werden.

Mnemonic:	LDIR	LDI	(Load, increment and repeat-Transportiers, erhöhe und wiederhole
Befehlskode: (hex)	ED B0	ED A0	
Wirkung:	Vor Ausführung der Befehle muß BC mit der Blocklänge ( $1 \leq (BC) \leq 65535$ ), 0065536! DE mit der Anfangsadresse des Zielbereiches und HL mit der Anfangsadresse des Quellbereiches gefüllt sein! LDIR bzw. LDI transportieren dann ein Datenbyte von von der Adresse (HL) zur Adresse (DE), erhöhen beide Registrierpaare um 1 und vermindern das Zählregister BC um 1. <u>Nur</u> der Befehl LDIR wiederholt diese Operation solange, bis BC den Wert 0 enthält.		

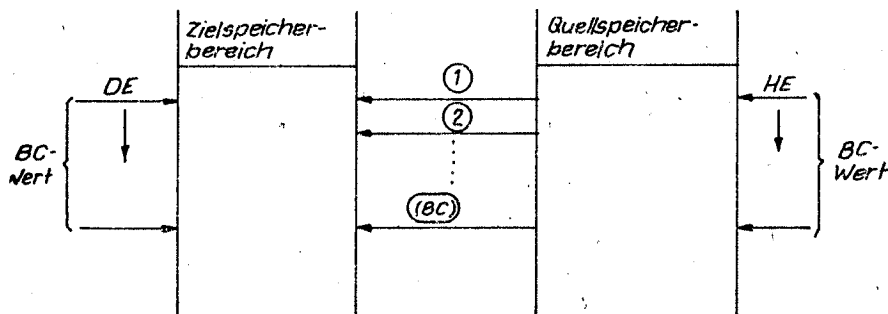
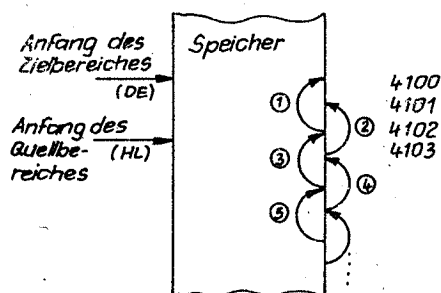


Bild 10.2. Wirkung des Befehles LDIR

LDIR bereitet lediglich die nächste Befehlsausführung vor. Die Befehle LDDR und LDD haben prinzipiell die gleiche Wirkung, nur daß hierbei der Transport mit der jeweils höchsten Adresse der Speicherbereiche beginnt und dann rückwärts bis zur Anfangsadresse fortgesetzt wird (LDDR → Load, decrement und repeat). Die Speicheradressen in HL und DE werden also nach jedem Byte-transport um 1 verringert.

**Beispiel:** Ein Programmbereich der Länge 100H ist um zwei Bytes im Speicher nach vorn zu verschieben.

Anfangsadresse: 4102H



Vor Programmbeginn füllen wir den interessierenden Speicherbereich mit der Monitorfunktion **FL** (Fill - Füllen) z.B. mit 00H (von 4102H bis 4201H). 4100H und 4101H füllen wir mit FFH. Monitorfunktionen wie **FL** stellen die dritte Belegung der Hexastatur dar (Ziffern, Registernamen, Funktionen) und sind über die Funktionstaste **FT** anwählbar.

**FU** **FU**

**FL** **FL**

Anschließend ist die Anfangsadresse des Speicherbereiches anzugeben:

**4** , **1** , **0** , **2** **FL4102**

**EXEC** **LE**

Jetzt ist die Länge des zu füllenden Bereiches einzugeben:

**1** , **0** , **0** **LE 100**

**EXEC** **DA**

und das einzuspeichernde Datenmuster anzugeben:

**0** , **0** **DA 00**

**EXEC** **F**

Das jetzt erscheinende F zeigt an, das die Funktion ordnungsgemäß ausgeführt wurde (F - Finish = Ende).

Anschließend ist mit der Funktion **MEM** der Inhalt der Speicherplätze 4100H und 4101H mit FFH zu laden.

Das folgende Programm realisiert den gewünschten Transport:

Adresse (hex)	Befehlscode (hex)	Mnemonic	Kommentar
4000	21,02,41	TRANS: LD HL,4102H	;HL:=erste Quelladresse
4003	11,00,41	LD DE,4100H	;DE:=erste Zieladresse
4006	01,00,01	LD BC,0100H	;BC:=Länge d. Daten- ; blockes in Bytes
4009	ED, BC	LDIR	;Transport
400B	C3,0B,40	T1: JMP T1	;Stoppschleife

Bei Ausführung dieses Programmes im Einzelszyklusbetrieb ist zu erkennen, daß ein Befehlszyklus von LDIR aus vier Zyklen besteht und bei jeder Wiederholung sogar der Befehlscode neu geladen wird. Bei Umspeicherung auf höhere Speicheradressen ist zur Vermeidung von Überspeicherungen der LDDR-Befehle sinnvoller! (Machen Sie sich diese Zusammenhänge anhand des Bildes 10.2. deutlich).

**Bemerkung:** Für Datentransporte im POLY-COMPUTER-Speicher steht die Monitorfunktion ME (move-transportieren) zur Verfügung, deren Handhabung dem Bedienungshandbuch zu entnehmen ist.

(b) Durchsuchen eines Datenblockes im Speicher nach einem festen Datenmuster

Diese Funktion ist mit einem der Befehle CPIR, CPI, CPDR, CPD zu realisieren. Die Zeichen I und D kennzeichnen wieder die Richtung der Durchsuchung (I-Inkrementieren der Adresse, d.h. suche von vorn; D-Dekrementieren der Adresse, d.h. suche vom Ende des Speicherbereiches her). Das R in Mnemonik deutet auf automatische Wiederholung der Operation (R-repeat) bis zur Erfüllung der Abbruchbedingung hin.

Mnemonic: CPIR CPI (compare, increment and repeat  
Vergleiche, inkrementiere und wiederhole)

Befehlscode: ED ED  
(hex) B1 A1

Wirkung: Vor Befehlsausführung müssen

HL die Anfangsadresse,

BC die Länge des zu durchsuchenden Bereiches und

A das gesuchte Datenmuster enthalten.

Der A-Inhalt wird mit dem Inhalt des Speicherplatzes mit der Adresse (HL) verglichen (A-(HL)=?). Anschließend wird die Adresse in HL um 1 erhöht und das Zählerregister BC um 1 verringert.

War A=(HL), so wird das Z-Flag gesetzt, ist BC=0 (d.h. Ende des Bereiches ohne Finden des Datenmusters), wird P/V-Flag=0 und sonst =1.

Nur der CPIR-Befehl wird solange weder A=(HL) noch BC=0 eingetreten ist, wiederholt.

In HL ist dann die Adresse des gesuchten Datenbytes enthalten.

Die CPDR- und CPD-Befehle verhalten sich analog, durchsuchen jedoch den Speicherbereich von der höchsten Adresse beginnend.

(c) Eingabe bzw. Ausgabe eines Datenblockes von bzw. zu der Peripherie.

Für die Eingabe existieren dazu die Befehle INI, INIR, IND und INDR, für die Ausgabe die Befehle OUTI, OTIR, OUTD und OTDR. Auf die Bedeutung der Mnemoniks trifft das in (a) und (b) bereits gesagte ebenfalls zu (R - steht für automatische Wiederholung, I bzw. D für die Ein- bzw. Ausgabe mit steigender bzw. fallender Adresse).

Für alle diese Befehle gilt:

- die Ein- bzw. Ausgabe erfolgt auf bzw. von der Speicheradresse die in HL enthalten ist;
- die Geräteadresse ist in C enthalten und bleibt bei der Operation konstant;
- im Register B ist die Blocklänge enthalten (1-255; 0=256).

Als Beispiel wird der Befehl INIR vorgestellt.

Mnemonic: INIR (Input, increment and repeat  
Eingabe, Inkrementieren und Wiederholen)

Befehlscode: ED ED  
(hex) B2

Wirkung: Vor Befehlsausführung müssen gefüllt werden:

C mit der Peripherie-Geräteadresse

HL mit der Anfangsadresse des Speichereingabebereiches

B mit der Anzahl der einzugebenden Bytes

INIR vollzieht dann eine Eingabe eines Datenbytes von dem Peripheriekanal mit der Adresse (C) auf den Speicherplatz der Adresse (HL). Anschließend wird HL um 1 erhöht und B um 1 vermindert. Diese Operationen werden solange wiederholt, bis B=0.

Eingaben mittels INIR werden meist in Zusammenarbeit mit der WAIT-Steuerung der CPU verwendet, so daß eine Synchronisation zwischen CPU und Peripherie erreicht wird.



10.2. Erzeugung von Maschinencode mittels Programmunterstützung

Die von uns praktizierte mnemonische und symbolische Programm-schreibweise wird auch als Quellcode eines Programmes be-zeichnet, die nach der Kodierung vorliegende Zahlenkolonne als Maschinencode. Die Umsetzung von Quellcode in Maschinencode ist vor allem bei größeren Programmen und bei Änderungen per Hand und Befehlstabelle sehr unhandlich, zeitraubend und fehler-anfällig. Aus diesem Grunde wurden Programme geschaffen, die diesen Übersetzungsprozeß automatisieren, sogenannte Sprachen-Übersetzer.

10.2.1. Assembler

Ein Assembler ist ein Übersetzungsprogramm, das aus dem mnemonischen Code den Maschinencode erzeugt.



Der erzeugte Code kann bereits auf absolute Speicherplätze bezogen sein (Maschinencode), oder aber noch für Adreßänderungen und Referenzen aus anderen Programmen offen sein (Objektkode). Den mnemonischen Code nennt man deshalb auch oft Assemblersprache. Neben Assemblersprachen, die sehr stark auf die individuellen Besonderheiten der jeweiligen CPU eingehen, existiert eine Vielzahl anderer Rechnersprachen, die stärker an bestimmte Problemstellungen angelehnt sind und deshalb auch als problemorientierte Sprachen bezeichnet werden. Ein wesentliches Ziel der Sprachentwicklung für Rechner ist es, eine möglichst nahe Verwandtschaft zu Ausdrücken der menschlichen Sprache zu erreichen, um den Lernprozeß bei der Aneignung von Rechnersprachen abzukürzen.

10.2.2. Die Assemblerliste

In den nachfolgenden Beispielen sowie für das Monitorprogramm wurde für die Sprachübersetzung ein Assemblerprogramm verwendet. Für das Verständnis dabei entstehender Programmlisten, die der Dokumentation dienen, soll ihr Aufbau erläutert werden.

Die Assemblerliste ist in vier Felder aufgeteilt.

Adreßfeld	Objektkodefeld	Zeilennummer	Quellkodefeld
-----------	----------------	--------------	---------------

Im Adreßfeld ist die Adresse des ersten Befehlscodebytes enthalten. Das Objektkodefeld beinhaltet den Befehlscode. Die Zeilennummer numeriert fortlaufend die Quellkodeszeilen. Im Quellkodefeld ist der nach Marken, Mnemonik, Variablen und Kommentar unterteilte Programmquelltext enthalten. Die Felder werden durch definierte Trennzeichen (z.B. Leerzeichen, Tabulator) abge sondert. Der Kommentar muß mit einem Semikolon beginnen.

Zur Verwaltung des Speicherplatzes und der diesbezüglichen Steuerung des Assemblerprogrammes darf der Quellcode sogenannte Pseudobefehle oder Pseudoanweisungen enthalten. Diese Anweisungen werden lediglich vom Assemblerprogramm verarbeitet, für sie existiert kein Befehlscode.

Die wichtigsten Pseudoanweisungen sind:

- |           |         |   |  |
|-----------|---------|---|--|
| ORG       | adresse | - | die nachfolgenden Befehle beginnen ab (adresse)<br>( - bedeutet augenblicklicher Wert des Speicherplatzzuweisungszählers)                        |
| DA        | adresse | - | definiere Adresse; eine 16 Bit-Adresse mit dem Wert (adresse) wird an dieser Stelle im Speicher erwartet; zwei Bytes Speicher werden reserviert. |
| DB        | byte    | - | definiere Byte; ein Byte mit dem Wert (byte) wird an dieser Stelle im Speicher erwartet; ein Byte Speicher wird reserviert                       |
| Marke EQU | wert    | - | Zuweisung eines Wertes; bei Auftauchen von Marke im programm wird bei der Übersetzung wert eingesetzt; es wird kein Speicher reserviert          |

Beispiel:            LD A,ZAHL            →    3E,05  
                         ZAHL EQU 5

### 10.3. Beispiele

Nachfolgend sind zwei umfangreiche Programmbeispiele aufgeführt die den vermittelten Lehrstoff integrieren und auch ein wenig Freude beim Umgang mit der Mikrorechner-technik erzeugen sollen. Für den Mikrorechner-Einsatz typischere Beispiele sind im Arbeitsbuch Teil II enthalten, die die externe Peripherie mit einbeziehen.

#### 10.3.1. Der POLY-COMPUTER als einfacher Taschenrechner

In diesem Abschnitt sollen Sie ein Programmierbeispiel kennenlernen, das auf dem POLY-COMPUTER das Verhalten eines einfachen Taschenrechners nachbildet. Es muß hier betont werden, daß die Funktion eines (nicht programmierbaren) Taschenrechners durch einen speziellen Schaltkreis mit wenig Aufwand realisiert werden kann. Eine sinnvolle Anwendung der Taschenrechnerfunktion ist aber beispielsweise gegeben, wenn ein Mikrorechner Meßdaten erfaßt und aufbereitet, mit denen anschließend noch von Hand arithmetische Operationen ausgeführt werden sollen. Der Bediener einer solchen Einrichtung braucht Meßdaten, mit denen er rechnen will, nicht mehr von einer Anzeige ablesen und in einem Taschenrechner eintippen; eine wesentliche Fehlerquelle wird vermieden.

Dieses Programm wurde als Beispiel aufgenommen, weil es viele Aspekte enthält, wie Arithmetik, Tastatur- und Anzeigebildung, Arbeit mit Tabellen, Unterprogrammtechnik mit Fehlermeldung an höhere Ebenen. Weiterhin gibt es viele Ansatzpunkte für Modifikationen. Zunächst müssen wir festlegen, welches Ein- und Ausgabeverhalten wir von unserem "Taschenrechner" fordern und, damit eng verbunden, sein globales Funktionsprinzip. Wir wählen ein Funktionsprinzip, das dem Verfahren mit Kellerspeicher bzw. der "umgekehrten polnischen Notation" ähnlich ist, weil dadurch eine besonders einfache und übersichtliche Programmstruktur möglich ist.

Es werden zwei Rechenregister X, Y benötigt, von denen jedes eine Zahl speichern kann (nicht zu verwechseln mit den Prozessorregistern IX, IY!).

Der Ablauf einer Rechnung ist nun wie folgt:

Die Eingabe einer Zahl erfolgt grundsätzlich in das Register X, angezeigt wird ebenfalls immer dieses Register.

Nachdem der erste Operand eingegeben wurde, drückt man die Taste **E** (Übernehmen der Zahl).

Die Betätigung dieser Taste bewirkt den Transport der eingegebenen Zahl vom Register X in Register Y. Anschließend wird der zweite Operand in das Register X eingegeben. Die daraufhin eingegebene Operationstaste bewirkt die Verknüpfung der Operanden, das Ergebnis muß im Register X abgespeichert werden, es wird angezeigt und kann weiter verarbeitet werden. Ausgehend von dieser Beschreibung ist eine Grobstruktur des Programms darstellbar (Bild 10.3.) Beachten Sie, daß in diese noch keine Einzelheiten des zu programmierenden Mikrorechners eingehen!

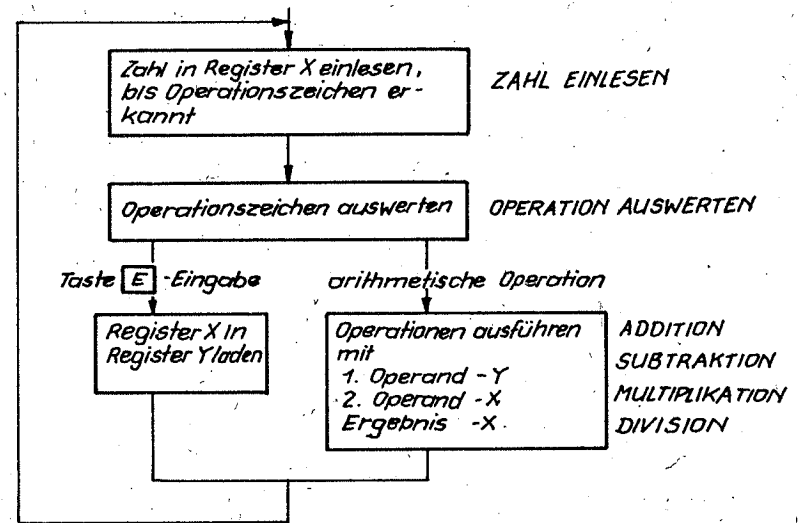


Bild 10.3. Programmstruktur

Der Programmablaufplan hat ausnahmsweise keinen Ausgang, der "Taschenrechner" läuft also solange, bis er durch das Betriebsprogramm (Taste **MON**) oder durch Rücksetzen unterbrochen wird. Als nächstes müssen wir den zu verarbeitenden Zahlenbereich festlegen. Im Interesse maximaler Einfachheit und Kürze beschränken wir uns darauf, positive ganze Zahlen zu verarbeiten die in zwei Bytes binär dargestellt werden, d.h.

$0 \leq n \leq 65535$ . Die Eingabe und Anzeige soll natürlich dezimal erfolgen, so daß Konvertierungen vom Dezimal- in das Binärsystem und umgekehrt notwendig werden.

Die Operationen Addition, Subtraktion, Multiplikation und Division sind mit Hilfe des im Prozessor U880 vorhandenen (binären) Wortarithmetik einfach zu realisieren.

Für die Zahlenregister X und Y können bei den getroffenen Festlegungen Registerpaare des Prozessors verwendet werden:

HL - X  
DE - Y

Nun kann die Verfeinerung der Moduln angegangen werden:

Modul: ZAHL EINLESEN

Bei der Gestaltung dieses Moduls ist zu beachten, daß jede eingegebene Ziffer sofort auch mit anzuzeigen ist. Außerdem muß bei der Eingabe der 1. Ziffer der alte X-Inhalt angezeigt werden, damit das Ergebnis der eventuell vorangegangenen Operationen abgelesen werden kann. Die Inhalte der Rechenregister X, Y dürfen nicht zerstört werden, ausgenommen die Eingabe eines neuen Wertes in X.

Begonnen wird mit der Anzeige des alten X-Inhalts. Zur Konvertierung wird ein Divisionsalgorithmus benutzt (siehe Abschnitt 1.2. - Zahlensysteme). Damit X nicht zerstört wird, benutzt die Konvertierung die Hilfsvariable XANZ. Der Algorithmus liefert zuerst die niederwertigste Dezimalstelle. Für die Umwandlung in eine 7-Segment-Darstellung und das positionieren in der Anzeige verwenden wir zweckmäßigerweise ein Unterprogramm des MONITORS mit dem Namen ZIPANZ.

Wenn Sie solche bereits vorhandenen Programmteile benutzen, müssen Sie sich vorher Klarheit verschaffen über deren Funktion und "Anschlußbedingungen" (d.h. welche Daten wo übergeben werden). Das Unterprogramm ZIPANZ ermittelt den 7-Segment-Kode einer hexadezimalen Ziffer und schiebt ihn von rechts in einen Anzeigebereich ein. Da die Dezimalziffer eine Untermenge der Hexadezimalziffern sind, können auch Dezimalziffern in die Anzeige geschrieben werden. Die bereits im gewählten Anzeigebereich stehenden Ziffern werden nach links verschoben. An das Unterprogramm ZIPANZ sind zu übergeben (siehe auch Bedienhandbuch):

Übergabe	in A, Bit 0-3	:binäre Kodierung der einzuschreibenden Ziffer
	in B	:Länge des Anzeigebereiches -1
	in HL	:Beginn des Anzeigebereiches

Wir wählen B=7 und HL-Anzeigepufferanfang; damit wird die gesamte 8-stellige Anzeige als Einheit benutzt.

Da unser Konvertierungsalgorithmus die anzuzeigenden Ziffern in der umgekehrten Reihenfolge liefert wie sie das Anzeigeprogramm benötigt, speichern wir die konvertierten Ziffern auf dem Stapelspeicher ab und holen sie wieder von dort. Dabei erhalten wir wie gewünscht die höchstwertigste Ziffer zuerst.

In unserem Programm folgt dann die Tastaturabfrage, für die wir ein Programm des MONITORS mit dem Namen KONSOL verwenden. Dieses Programm erst nimmt die gemultiplizierte Ansteuerung der Anzeige mit der im Anzeigepuffer stehenden Belegung vor. Weiterhin fragt es die Tasten ab und übergibt bei Drücken einer Taste eine dieser zugeordnete Kodierung. Für unsere Anwendung interessieren uns folgende Übergabebedingungen: (näheres im Bedienhandbuch)

Übergabe:	in DE	:Anzeigepufferanfang
Rückgabe:	Z-Flag gesetzt,	wenn keine Taste gedrückt
	in C	:binäre Kodierung bei betätigter Taste der Hexadezimaltastatur (OO-OPH)

Da wir für eine dezimale Eingabe nur die Ziffern 0...9 benötigen können die Tasten **A** - **P** gleich als Funktionstasten benutzt werden.

Wenn gleich als erste Taste eine Operationstaste gedrückt wurde, dann erfolgt keine Änderung von X, um Kettenrechnungen zu ermöglichen. Wenn eine Zifferntaste gedrückt wurde, dann bildet diese den Anfangswert des neuen X-Register-Inhalts, Die Anzeige wird gelöscht und anschließend die eingegebene Ziffer als höchstwertigste Dezimalstelle ganz rechts angezeigt. Durch einen Multiplikationsalgorithmus werden die weiteren Dezimalstellen konvertiert. Wenn eine Operationstaste erkannt wird, dann ist der neue X-Register-Inhalt gültig. Bild 10.4. zeigt den Programmablaufplan dieses Moduls. Er ist genügend detailliert für eine Umsetzung in Rechnerbefehle. Die Programme dieses Beispiels sind am Schluß des Abschnitts aufgeführt.

Division wurde auf eine Subtraktion zurückgeführt, d.h. vom X-Wert wird so oft 10 subtrahiert, bis ein Überlauf eintritt. Wenn man zu dem so erhaltenen Wert wieder 10 addiert, erhält man den Divisionsrest. Die Anzahl der Differenzbildungen entspricht dem Quotienten.

Die für die zweite Konvertierung notwendige Multiplikation führen wir auf eine wiederholte Addition zurück. Dabei kann es vorkommen, daß wir eine zu große Zahl eingeben (die nicht in 16 Bit darstellbar ist). In einem solchen Fall müssen wir die Eingabe abbrechen und eine Fehlermeldung auslösen. Da Fehler auch an anderen Stellen auftreten können, wird ein gemeinsam benutzter Fehlermeldemechanismus zweckmäßig sein. Es wäre jetzt aber un- zweckmäßig, bei Feststellung eines Fehlers einfach zur Fehlerbe- handlung zu springen. Es ist vielmehr vor allem im Sinne einer guten Strukturierung sinnvoll, nur an das Programm, welches das Unterprogramm ZAHL aufruft, ein Kennzeichen über den aufgetretenen Fehler zu übergeben und die eigentliche Reaktion dann in der höchsten Programmebene vorzunehmen. Wir benutzen als Fehlerkenn- zeichen das gesetzte Übertrags-Flag (CY=1).

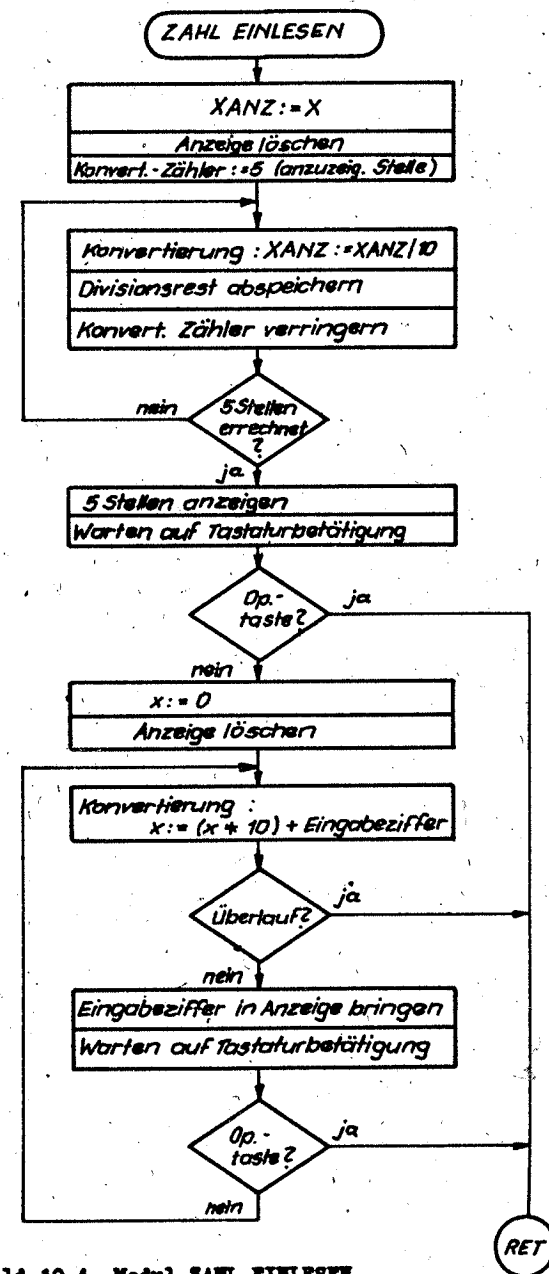


Bild 10.4. Modul ZAHL EINLESEN

Modul: OPERATION AUSWERTEN

Dieser Modul nimmt eine zentrale Stellung in dem Programm ein, deshalb bildet er die höchste Programmebene, von der aus die übrigen Moduln als Unterprogramme aufgerufen werden. Das eigentliche Problem besteht darin, daß in Abhängigkeit des eingegebenen Operationszeichens (Tasten A-F, Kodierung deshalb OA-OF) verschiedene Funktionsunterprogramme aufzurufen sind. Dies wird realisiert durch eine abgespeicherte Liste, die die Anfangsadresse der Unterprogramme in geordneter Reihenfolge enthält. Dabei steht wie üblich das niederwertige Byte der Anfangsadressen auf der niedrigeren Speicheradresse.

Basisadresse Funktion  
(Hexadezimaltaste)

A	Anfangsadresse	ADDITION
B	Anfangsadresse	SUBTRAKTION
C	Anfangsadresse	MULTIPLIKATION
D	Anfangsadresse	DIVISION
E	Anfangsadresse	EINGABE
F	Anfangsadresse	RESERVE

Aus dem Programmablaufplan (Bild 10.5.) ist zu ersehen, daß nach dem Einlesen einer Zahl und dem darauffolgenden Operationszeichens (erster Modul) die Tabellenposition errechnet wird. Durch Subtraktion von 10 wird der Wertebereich der Operationstastenkodierungen auf 0-5 verschoben. Die anschließende Verdopplung ist notwendig, da jeder Kodierung 2 Byte, d.h. eine 16-Bit-Adresse entsprechen soll. Durch Addition des entstandenen Wertes zur Basisadresse der Tabelle entsteht ein Zeiger auf das niederwertige Byte der ausgewählten Unterprogramm-Startadresse. Sowohl bei der Eingabe als auch beim Rechnen können Situationen auftreten, die eine korrekte Verarbeitung verhindern (z.B. Division durch 0). Die aufgerufenen Programme liefern bei ihrer Rückkehr ein Fehlerkennzeichen (CY-Flag), bei dessen Auftreten eine auffällige Anzeige erfolgen soll.

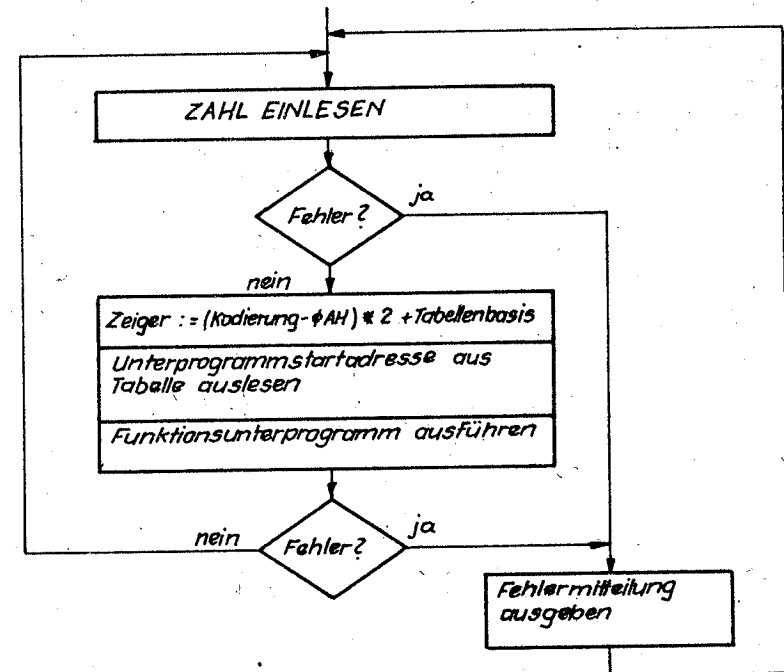


Bild 10.5. Modul OPERATION AUSWERTEN

Diese wird durch Benutzung eines weiteren MONITOR-Unterprogramms, FUNKAN, realisiert, mit dessen Hilfe zwei beliebige Zeichen in die ersten Stellen der Anzeige gebracht werden (hier "FE" für Fehler), der Rest der Anzeige wird gelöscht.

Übergaben: in DE - 7-Segment-Kodierung für "FE"  
in HL - Anzeigepufferanfang

Anschließend wird wieder das schon besprochene Tastaturabfrageprogramm KONSOL so lange aufgerufen, bis eine Taste gedrückt wurde. Auf diese Weise bleibt die Fehleranzeige bestehen, bis der Bediener durch Betätigen einer Taste (in unserem Falle einer beliebigen) quittiert, daß er die Fehlermeldung bemerkt hat.

Modul: ARITHMETIK

Erst in diesem Modul erfolgt die eigentliche Rechnung.

Er besteht aus Unterprogrammen für jede Rechenart sowie für die Eingabe und Reservfunktion.

Die Addition kann bei den getroffenen Festlegungen für die Darstellung von X und Y sehr einfach durch den Maschinenbefehl ADD HL, DE erfolgen, das Ergebnis steht wie gewünscht im Register HL (X). An dieser Stelle müssen wir aber noch etwas unbedingt beachten: Der Benutzer des "Taschenrechners" kann beliebige Werte eingeben. Wenn diese den Darstellungsbereich überschreiten, wird bereits vom Eingangmodul wie beschrieben eine Fehlermeldung veranlaßt. Genauso wichtig ist es aber auch, Bereichsüberschreitungen oder unzulässige Operanden bei den Verarbeitungsoperationen festzustellen und anzuzeigen. Damit die Auswertung und Anzeige der genannten Fehler nicht in jedem Verarbeitungsprogramm erfolgen muß, übergeben die Arithmetikunterprogramme nur eine Mitteilung an die aufrufende Ebene über das Auftreten eines Fehlers. In unserem Falle verwenden wir das gesetzte Übertrags-Flag (CY), um einen aufgetretenen Fehler zu melden. Diese Festlegung ist zweckmäßig, da beispielsweise bei der Addition mit dem Befehl ADD HL, DE dieses Flagbit gesetzt wird, wenn die Operation einen Übertrag in das (nicht mehr vorhandene) 17. Bit verursacht. Sie sollten aber immer beachten, daß die Art und Weise der Fehlerdiagnostizierung u.a. von der Zahlendarstellung abhängig ist und in anderen Anwendungen eventuell modifiziert werden muß. Für die Subtraktion steht der Befehl SBC HL, DE zur Verfügung. Vor dessen Ausführung ist zu sichern, daß sich die Operanden in den richtigen Registern befinden und das Übertrags-Flag 0 ist.

Falls bei der Operation eine in unserem Beispiel nicht darstellbare negative Zahl entstehen würde, ist wieder das CY-Flag gesetzt, was der höheren Ebene den eingetretenen Fehler signalisiert. Die Multiplikation ist im Prozessor U880 nicht als Maschinenbefehl vorhanden, wir realisieren sie deshalb durch fortlaufende Addition:

$$a \cdot b = a + a + a + \dots + a$$
$$b - \text{mal}$$

Die Zählung des einen Faktors erfolgt durch Verringern (DEC) und Test auf 0; der Test auf 0 muß dabei vorangehen, damit auch der Faktor 0 richtig verarbeitet wird.

Bei der Addition kann die Überprüfung auf Bereichsüberschreitung wieder aufgrund des gesetzten Übertrags-Flags erfolgen. Dieses Unterprogramm hat zwei Austrittspunkte (im Normal- und im Fehlerfall). Es ist hier besonders darauf zu achten, daß bei ordnungsgemäßer Ausführung der Operation das Fehlerkennzeichen nicht gesetzt ist. Dies ist erfüllt, da die logische Operation OR das CY-Flag zurücksetzt.

Die Division führen wir auf eine wiederholte Subtraktion des 2. Operanden zurück, dabei wird gezählt, wie oft dieser vom 1. Operanden subtrahiert werden kann, ohne daß Übertrag eintritt. Die Anzahl der Operanden ist dann der Quotient. Es wird also nur der ganzzahlige Teil eines Quotienten berechnet, ein eventueller Rest wird ignoriert. Da sich das Auftreten eines Übertrags nachträglich bei der Subtraktion einfacher feststellen läßt als dessen voraussichtliches Auftreten vor der Operation, wird hinterher auf Überlauf geprüft. Der auf diese Weise um 1 erhöhte Quotient wird korrigiert, indem die Zählung nicht bei 0 sondern bei -1 (0FFFFH) beginnt. Eine Bereichsüberschreitung ist nur gegeben, wenn eine Division durch 0 versucht wird, dieser Fall wird vorher geprüft und die Fehlermarkierung CY=1 bei seinem Auftreten gesetzt.

Die Unterprogramme für Eingabe und Reservfunktion müssen das Fehler-Flag CY rücksetzen, damit kein Fehler vorgetäuscht wird. Die Reservfunktion (Taste F) führt mit dem angegebenen Unterprogramm noch überhaupt nichts aus. Hier können Sie eigene Funktionen einsetzen wie beispielsweise Quadrieren oder Radizieren. Da Sie mit der Bedienung Ihres POLY-COMPUTERS 880 nun hinreichend vertraut sind, wird es Ihnen keine Schwierigkeiten bereiten, das Programm einzugeben und zum Laufen zu bringen. Beachten Sie, daß die jeweils ganz links in der Programmliste stehenden Speicheradressen einzuhalten sind, da andernfalls beispielsweise die Zieladressen bei absoluten Sprüngen und Aufrufen nicht mehr richtig sind. Zwischen den einzelnen Programmteilen wurden jeweils einige Bytes freigelassen, dies erleichtert es Ihnen, bei Bedarf noch geringfügige Änderungen vorzunehmen, ohne daß sich der gesamte Programmrest verschiebt. Wenn Sie sich überzeugt haben, daß Sie das Programm korrekt eingegeben haben, können Sie es auf der Adresse vor der Marke TARECH starten.

Die korrekte Funktion können Sie mit Testbeispielen überprüfen, ebenso die Reaktion auf falsche Eingaben und Überläufe. Sollte das Programm nicht arbeiten, überprüfen Sie noch einmal den Maschinencode und die Stellung des Stapelzeigers SP (er muß auf einen freien RAM-Bereich zeigen). Wenn Sie sich intensiver mit dem Programm beschäftigen, werden Sie sicher einiges bemerken, was verbesserungsfähig ist (beispielsweise wird beim Starten erst einmal irgendein Wert in X/Y angezeigt). Versuchen Sie, von Ihnen gewünschte Verbesserungen in das Programm einzubringen. Sie können auch das Funktionsprinzip und den gegebenen Rahmen beibehalten und Änderungen grundlegender Natur erproben, beispielsweise Rechnen mit BCD-kodierten Zahlen, negativen und positiven Zahlen, Zahlen höherer Genauigkeit (in 3 und mehr Bytes im RAM dargestellt), schnellere Algorithmen für Multiplikation und Division. Sie werden feststellen, daß mit steigenden Ansprüchen auch der Speicherbedarf der Programme und (nicht zu unterschätzen) der Aufwand an Programmierarbeit schnell zunehmen. So benötigen beispielsweise Arithmetikpakete mit Gleitkommadarstellung (beispielsweise 1 Byte Exponent, 3 Byte Mantisse) für die vier Grundrechenarten sowie die dazugehörigen Ein- und Ausgabekonvertierungen bereits mehr als 1KByte Programmspeicher. Es ist empfehlenswert, für Arithmetikanwendungen fertige Standard-Programmpakete zu benutzen, es sei denn, daß ganz spezielle Anforderungen dagegen sprechen.

ADR	OBJ-KODE	NR	TRECH QUELLANWEISUNG
		1	
		2	PROGRAMMIERBEISPIEL "TASCHENRECHNER"
		3	
		4	MODUL "ZAHL EINLESEN"
		5	UEBERGABE REGISTER X - HL
		6	(REGISTER Y - DE: NICHT AENDERN)
		7	AUFGABE: ANZEIGE DES X-REGISTERS
		8	EINLESEN EINES NEUEN X-INHALTS
		9	BIS OPERATIONSTASTE ERKANNT
		10	
4000		11	ORG 4000H
4000	D5	12	ZAHL PUSH DE
4001	E5	13	PUSH HL ; X, Y IN STACK RETTEN
4002	CD6C40	14	CALL LOANZ ; ANZEIGE LOESCHEN
4005	3E05	15	LD A, 5 ; 5 STELLEN ANZEIGEN
4007	010A00	16	ZAHL1 LD BC, 10 ; FAKTOR 10 DEZIMAL
400A	11FFFF	17	LD DE, 0FFFFH ; QUOTIENT:= -1
400D	13	18	ZAHL2 INC DE
400E	A7	19	AND A ; CY:=0
400F	ED42	20	SBC HL, BC
4011	30FA	21	JRNC ZAHL2
4013	09	22	ADD HL, BC ; IN HL DIV.-REST
4014	E5	23	PUSH HL ; ABSPEICHERN
4015	EB	24	EX DE, HL
4016	3D	25	DEC A
4017	20EE	26	JRNC ZAHL1
		27	; KONVERTIERTE ZAHL ANZEIGEN
4019	0E05	28	LD C, 5 ; 5 STELLEN
401B	D1	29	ZAHL3 POP DE ; ZIFFER IN E
401C	7B	30	LD A, E
401D	0607	31	LD B, 7
401F	217F40	32	LD HL, ANZ
4022	CD0003	33	CALL ZIFANZ ; ZIFFER IN ANZEIGE
4025	0D	34	DEC C
4026	20F3	35	JRNC ZAHL3
		36	; TASTATURABFRAGE, NEUES X ODER OPERATION
4028	117F40	37	ZAHL4 LD DE, ANZ
402B	CD4E01	38	CALL KONSOL
402E	28F8	39	JRZ ZAHL4 ; WARTEN BIS TASTE BETAET.
4030	79	40	LD A, C ; HEX. TASTENKODE
4031	FE0A	41	CMP 0AH
4033	302D	42	JRNC ZAHL6 ; SPRUNG, WENN TASTE A..F
4035	CD6C40	43	CALL LOANZ
4038	210000	44	LD HL, 0 ; X - ANFANGSWERT
403B	E5	45	ZAHL5 PUSH HL
403C	F5	46	PUSH AF
403D	0607	47	LD B, 7
403F	217F40	48	LD HL, ANZ
4042	CD0003	49	CALL ZIFANZ ; EINGEG. ZIFFER ANZEIGEN
4045	F1	50	POP AF
4046	E1	51	POP HL
4047	5F	52	LD E, A
4048	1600	53	LD D, 0 ; NEUE ZIFFER NACH DE
404A	060A	54	LD B, 10 ; FAKTOR 10
404C	EB	55	EX DE, HL
404D	19	56	ZAHL7 ADD HL, DE ; X:=(10*X)+ZIFFER
404E	3813	57	JRC ZAHLF ; ZAHL ZU GROSS
		58	; RUECKKEHR MIT FEHLERMARKIERUNG CY=1

ADR	OBJ-KODE	NR	TRECH QUELLANWEISUNG	SEITE 2 POLY-880 ASM 1.0
4050	10FB	59	DJNZ ZAHL7	
4052	E5	60	PUSH HL	
		61	TASTATURABFRAGE NACH WEITEREN ZIFFERN	
4053	117F40	62	ZAHL8 LD DE, ANZ	
4056	CD4E01	63	CALL KONSOL	
4059	28F8	64	JRZ ZAHL8	
405B	E1	65	POP HL	
405C	79	66	LD A, C	
405D	FE0A	67	CMP 0AH	
405F	38DA	68	JRC ZAHL5	SPRUNG, WENN WEITERE ZIFFER
4061	E3	69	EX (SP), HL	NEUES X ERSETZT ALTES
4062	A7	70	AND A	CY1=0, KEIN FEHLER
4063	E1	71	ZAHLF POP HL	
4064	D1	72	POP DE	X, Y IN HL, DE
4065	C9	73	RET	
406C		74	ORG m+6	
		75		
		76	HILFSPROZEDUR "LOESCHEN DER ANZEIGE"	
406C	E5	77	LOANZ PUSH HL	HL RETTEN
406D	0608	78	LD B, 8	8 ANZEIGESTELLEN
406F	217F40	79	LD HL, ANZ	
4072	3600	80	LO1 LD (HL), 0	
4074	23	81	HL INC HL	
4075	10FB	82	DJNZ LO1	
4077	E1	83	POP HL	
4078	C9	84	RET	
407F		85	ORG m+6	
		86		
		87	ANZEIGEBEREICH	
407F		88	ANZ BER 8	8 BYTE ANZEIGEBEREICH
		89		
		90	STARTADRESSEN DER MONITOR-UNTERPROGRAMME	
		91	(SIEHE BEDIENBESCHREIBUNG)	
		92	KONSOL EQU 14EH	
		93	ZIFANZ EQU 300H	

ADR	OBJ-KODE	NR	TRECH QUELLANWEISUNG	SEITE 3 POLY-880 ASM 1.0
		97	MODUL "OPERATION RUSHERTEN"	
		98	UEBERGABE: A - FUNKTIONSKODIERUNG 0A-0F	
		99	RUECKGABE: IX - STARTADRESSE DES FUNKTIONS-UP	
		100	DE, HL DUERFEN NICHT VERRENDERT WERDEN	
		101	ALGORITHMUS: AUSLESEN EINER LISTE DER START-	
		102	ADRESSEN MIT EINEM ZEIGER IN	
		103	(KODIERUNG-0AH) * 2 + LISTENANFANGSADRESSE	
4087	E5	105	OPAU5W PUSH HL	
4088	21A040	106	LD HL, LISTE	
4088	D60A	107	SUB 0AH	RI=KODIERUNG-0AH
408D	07	108	RLCA	RI=20A
408E	4F	109	LD C, A	
408F	0600	110	LD B, 0	ERG. NACH BC
4091	09	111	ADD HL, BC	
4092	4E	112	LD C, (HL)	NIEDERWERT. TEIL
4093	23	113	INC HL	
4094	46	114	LD B, (HL)	HOEHERWERT. TEIL
4095	C5	115	PUSH BC	
4096	FDE1	116	POP IV	STARTADRESSE IN IV
4098	E1	117	POP HL	
4099	C9	118	RET	
40A0		119	ORG 6+6	
		120		
		121	LISTE DER STARTADRESSEN DER FUNKTIONEN	
		122	IN DER REIHENFOLGE DER TASTENKODIERUNGEN	
		123		
40A0	D940	124	LISTE DA	ADD TASTE "A"
40A2	E140	125	DA	SUBW TASTE "B"
40A4	EC40	126	DA	MULW TASTE "C"
40A6	FF40	127	DA	DIVW TASTE "D"
40A8	1941	128	DA	EINGAB TASTE "E"
40AA	2241	129	DA	FREI TASTE "F"
		130		
		131	HAUPTPROGRAMM TASCHEURECHNER	
		132		
40AC	CD0040	133	TARECH CALL ZAHL	
40AF	380B	134	JRC FEHLER	
40B1	CD8740	135	CALL OPAU5W	
40B4	01BA40	136	LD BC, TARE1	RUECKKEHRADRESSE DER UP
40B7	C5	137	PUSH BC	AUF STACK ABSPEICHERN
40B8	FDE9	138	JMP (IV)	UP ANSPRINGEN
40BA	30F0	139	TARE1 JRNC TARECH	WEITERRECHNEN WENN
		140		KEIN FEHLER
		141	ANZEIGE EINES FEHLERS-	
40BC	E5	142	FEHLER PUSH HL	
40BD	D5	143	PUSH DE	
40BE	217F40	144	LD HL, ANZ	
40C1	117371	145	LD DE, 07173H	KODE FUER ANZEIGE "FE"
40C4	CDAF02	146	CALL FUNKAN	ANZEIGEN
40C7	117F40	147	LD DE, ANZ	
40CA	CD4E01	148	CALL KONSOL	WARTEN AUF TASTATURBETRIETIGUNG
40CD	28F8	149	JRZ FE1	ZUR FEHLERQUITTIERUNG
40CF	D1	150	POP DE	
40D0	E1	151	POP HL	
40D1	18D9	152	JR TARECH	
		153	UNTERPROGRAMM FUNKAN DES MONITORS	
		154	FUNKAN EQU 2AFH	
40D9		155	ORG 6+6	



ADR	OBJ-KODE	NR	TRECH QUELLANWEISUNG	SEITE 4 POLY-880 ASM 1.0
		160	: MODUL "ARITHMETIK"	
		161	: ALLGEMEINE MERKMALE:	
		162	: 1. OPERAND - DE	
		163	: 2. OPERAND - HL	
		164	: ERGEBNIS - HL	
		165	: BEI UNZULAESSIGER OPERATION: CY:=1	
		166		
		167	: 16-BIT-ADDITION	
40D9	19	168	ADDW ADD HL, DE	
40DA	C9	169	RET	
40E1		170	ORG \$+6	
		171		
		172	: 16-BIT-SUBTRAKTION	
40E1	EB	173	SUBW EX DE, HL	
40E2	A7	174	AND A	: CY:=0
40E3	ED52	175	SBC HL, DE	
40E5	C9	176	RET	
40EC		177	ORG \$+6	
		178		
		179	: 16-BIT-MULTIPLIKATION	
40EC	44	180	MULW LD B, H	
40ED	4D	181	LD C, L	: FAKTOR NACH BC
40EE	210000	182	LD HL, 0	: ANF. WERT PRODUKT
40F1	78	183	MULW1 LD A, B	
40F2	B1	184	OR C	: A=0, WENN BC=0
40F3	C8	185	RZ	
40F4	08	186	DEC BC	: FAKTOR HERUNTERZAEHLEN
40F5	19	187	ADD HL, DE	
40F6	D8	188	RC	: UEBERLAUF
40F7	18F8	189	JR MULW1	
40FF		190	ORG \$+6	
		191		
		192	: 16-BIT-DIVISION	
40FF	EB	193	DIVW EX DE, HL	
4100	7A	194	LD A, D	
4101	B3	195	OR E	: A=0, WENN DIV. DURCH 0
4102	280D	196	JRZ DIVW0	
4104	01FFFF	197	LD BC, 0FFFFH	: ANF. WERT QUOTIENT
4107	03	198	DIVW1 INC BC	
4108	A7	199	AND A	: CY:=0
4109	ED52	200	SBC HL, DE	
410B	30FA	201	JRNC DIVW1	
410D	60	202	LD H, B	
410E	69	203	LD L, C	: QUOTIENT NACH HL
410F	A7	204	AND A	: CY:=0
4110	C9	205	RET	
4111	37	206	DIVW0 SCF	: CY:=1, FEHLER
4112	C9	207	RET	
4119		208	ORG \$+6	
		209		
		210	: EINGABE, REGISTERTAUSCH	
4119	EB	211	EINGAB EX DE, HL	
411A	A7	212	AND A	: CY:=0
411B	C9	213	RET	
4122		214	ORG \$+6	
		216	: RESERVEFUNKTION	
4122	A7	219	FREI AND A	: CY:=0
4123	C9	220	RET	

### 10.3.2. Der POLY-COMPUTER macht Musik

#### 1. Problemstellung

In diesem Abschnitt soll ein Programm entwickelt werden, das die "Ausgabe" von Musik über den Magnetbandanschluß ermöglicht. Nach entsprechender Programmierung spielt der Rechner eine vorher eingegebene Melodie.

Der Magnetbandanschluß ermöglicht bekanntlich die Abgabe von Signalen, die in ihrer Amplitude diskret sind, d.h. sie können nur eine endliche Anzahl von Werten annehmen. (In diesem Fall sind es nur zwei Werte: +100mV und -100mV). Damit lassen sich im Prinzip nur Rechteckschwingungen erzeugen. Die Klangfarbe eines Instruments ist jedoch durch den Oberwellengehalt (n-fache Frequenzen der Grundfrequenz  $f_0$  (z.B.: 440Hz), also  $2f_0$ ,  $3f_0$  usw.) der abgegebenen Schwingungen bestimmt. Durch die Einschränkung, nur Rechteckschwingungen erzeugen zu können, ist die Klangfarbe also bereits festgelegt. Weiterhin ist nur die Erzeugung monophoner Musik möglich, d.h. am Ausgang ist für die Dauer eines Tones nur eine Frequenz (sowie ihre Oberwellen) vorhanden. Diese beiden Einschränkungen könnten durch die Ausgabe analoger Signale (jeder beliebige Amplitudenwert möglich) umgangen werden:

- Es können beliebige Kurvenformen und damit entsprechende Oberwellenanteile erzeugt werden. Die Nachbildung der Klangfarbe eines bestimmten Instruments (z.B. Konzertorgel) wäre möglich.
- Die Addition der Signale zweier Frequenzen (z.B.: Terz, Quinte) ergibt wiederum ein analoges Signal, das ausgegeben werden kann. Damit kann polyphone Musik erzeugt werden.

Die Erzeugung analoger Signale unter Benutzung der Hardware des Magnetbandanschlusses ist durch Anwendung des Verfahrens der Pulsdauermodulation denkbar. In Bild 10.6. wird dies Prinzip erläutert.

Signal am  
Magnetbandausgang  
Signal nach Durch-  
laufen eines Tief-  
passes:

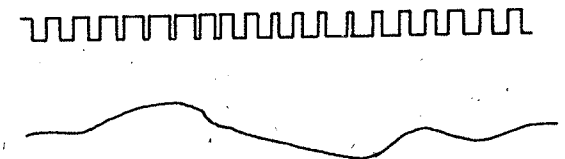


Bild 10.6. Pulsdauermodulation

Dabei wird das diskrete Ausgangssignal über einen Tiefpaß (Integration der Signalamplituden) "gemittelt". Die dabei entstehende Ausgangsspannung ergibt sich aus der Differenz der Zeiten von "Hoch" und "Tief". Da diese Differenz beliebige Werte (analoge) annehmen kann, ist auch die Erzeugung beliebiger Amplitudenwerte nach dem Tiefpaß möglich. Als Tiefpaß kann bereits das im POLY-COMPUTER vorhandene RC-Glied von 10 nF und 1 kOhm am Magnetbandausgang verwendet werden.

Weitere Forderungen an das zu erstellende Programm sind: Der Tonempfang sollte etwa 3 Oktaven betragen. Die Tondauer sollte von  $\frac{1}{16}$  bis zu einer ganzen Note variierbar sein. Es sollten auch Pausen möglich sein.

Damit sind die Grundprobleme der Aufgabenstellung umrissen.

2. Problemanalyse

2.1. Grundfrequenzen der Töne

Es ist bekannt, daß eine Oktave aus 12 Halbtönschritten besteht. Eine Tonleiter wird durch Auswahl von 8 Tönen aus den 12 möglichen gebildet (2 Halbtönschritte, 6 Ganztönschritte). Die Frequenzen der 12 Halbtönschritte bilden eine geometrische Reihe, d.h. die Frequenz eines Tones ergibt sich aus der Frequenz des darunterliegenden Tones durch Multiplikation mit dem Faktor  $\sqrt[12]{2}$  (wohltemperiertes Klavier). Als Näherung wird hier verwendet:

$\sqrt[12]{2} \approx 1,05946$ . Weiterhin ist bekannt, daß der Kammerton a einer Frequenz von 440 Hz entspricht. Für die Implementierung ist nicht so sehr die Frequenz, sondern die Periodendauer einer Schwingung von Bedeutung. Als Zeitbasis findet im POLY-COMPUTER der Maschinentakt mit der Frequenz von 921,60 kHz Verwendung. Dessen Periodendauer beträgt 1,08507 µs. Sinnvollerweise werden deshalb die Periodendauern der einzelnen Töne durch die Anzahl der Maschinentakte ausgedrückt.

Dabei entsteht folgendes Resultat:

Ton	T/2 in Maschinentakten	Frequenz
a	2094	220 Hz
ais	1977	
h	1866	
c	1761	
cis	1662,4	
d	1569	
dis	1398	
⋮	⋮	

Ton	T/2 in Maschinentakten	Frequenz
a'	1047,3	440 Hz
ais'	988,5	
h'	933,0	
⋮	⋮	
gis''	554,6	880 Hz
a''	523,6	
ais''	494,3	
h''	466,5	
c''	440,3	
⋮	⋮	
a'''	261,8	1760 Hz

Es ist auch bekannt, daß der tatsächlich erzeugte Ton in seiner Frequenz nicht mehr als ein Promille vom theoretischen Wert abweichen darf, damit die Verstimmlung in erträglichen Grenzen bleibt. Aus den oben gefundenen Werten für die einzelnen Töne wird ersichtlich, daß sich diese Forderung bei Nutzung des Maschinentaktes des POLY-COMPUTER als Zeitbasis nur etwa bis zum Ton a'' einhalten läßt. Darüber kann der Fehler > 1‰ werden. Es ist also zu fordern, daß das zu erstellende Programm die Erzeugung von Halbschwingungen am Magnetbandausgang ermöglicht, die sich in ihrer Dauer exakt auf die entsprechende Anzahl von Maschinentakten einstellen lassen.

2.2. Erzeugung analoger Signale

Zur Erzeugung analoger Signale mittels Pulsdauermodulation ist die Ausgabe einer Grundschwingung erforderlich, die mindestens über dem Hörfrequenzbereich ( $\approx 15$  Hz bis  $\approx 17$  kHz) liegt. Die halbe Periodendauer einer 17 kHz-Frequenz beträgt jedoch nur 27 Maschinentakte. Das entspricht etwa 3 Befehlen! Damit ist die Berechnung der Zeitdauer, sowie die erforderliche Ausgabe (Ende der Halbschwingungen) völlig unmöglich. Es ist deshalb nur die Erzeugung von Rechteckschwingungen einer Frequenz (Nomenklatur) möglich (im Rahmen dieses Lösungsansatzes).

2.3. Die Erzeugung der Grundfrequenz eines Tones

Wie bereits unter 2.1. festgestellt wurde, muß die Tonhöhenzeugung so erfolgen, daß sich praktisch jede mögliche Anzahl von Maschinentakten für eine Halbschwingung einstellen läßt (immerhalb bestimmter Grenzen).

Eine sinnvolle Lösung, wäre der Einsatz eines programmierbaren Teilers, der  $n$  Maschinentakte abzählt und anschließend ein geeignetes Signal (Ende der Halbperiode) abgibt.  $n$  ist dabei die Anzahl der Maschinentakte pro Halbperiode und kann programmiert werden. Der im POLY-COMPUTER vorhandene CTC ist hierfür ungeeignet, da er nicht in der Lage ist, einzelne Maschinentakte zu zählen (Vorteiler bzw.  $0 \neq \text{Zählfrequenz} \neq \text{Maschinentaktfrequenz}/2$ ). Das Problem läßt sich jedoch prinzipiell auch durch ein Programm lösen. Der entsprechende Programmablaufplan ist im Bild 10.7. dargestellt.

Zur Erzeugung der Zeit für  $T/2$  werden 2 Zeitschleifen verwendet, die jeweils eine Verzögerung von  $n \times$  Schleifenkonstante ( $A$  oder  $B$ ) bewirken.

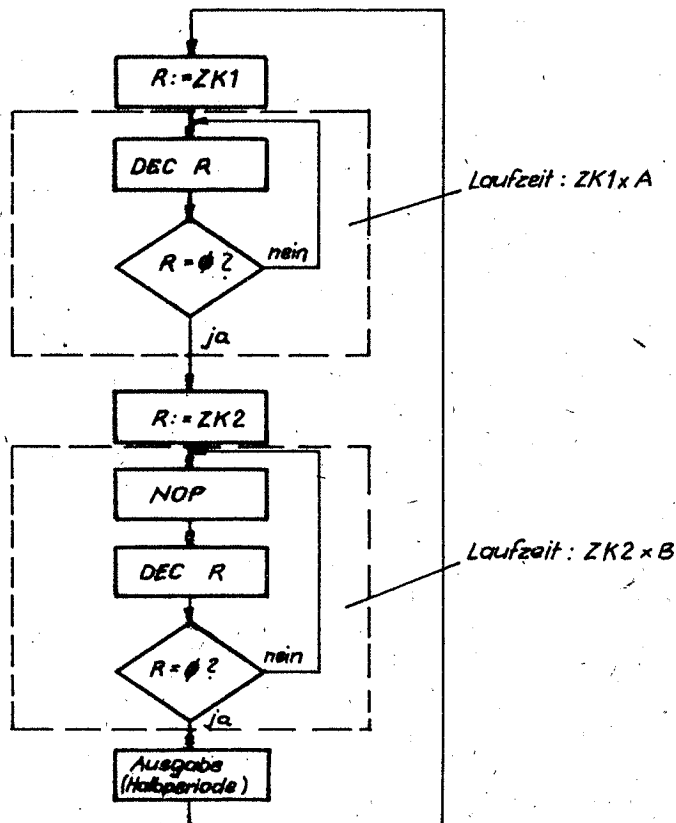


Bild 10.7. Erzeugung von  $T/2$

Die Laufzeiten  $A$  und  $B$  sind so zu wählen, daß  $A$  und  $(A - B)$  keine gemeinsamen Primfaktoren enthalten. Dann läßt sich durch geeignete Kombination von  $ZK1$  und  $ZK2$  jede beliebige Taktanzahl einstellen, die größer als  $(A \times B) + C$  ist.  $C$  ist dabei die Laufzeit der Routinen für die Ausgabe bei Ende der Halbperiode, Sprung zum Schleifenanfang und für das Laden der Zeitkonstanten.

#### 2.4. Erzeugung von Tönen definierter Dauer

Mit der Schleife an Bild 10.7. läßt sich ein Ton bestimmter (wählbarer) Höhe erzeugen. Nach einer durch die Note ( $\frac{1}{15}$ ,  $\frac{1}{8}$ ,  $\frac{1}{4}$  etc.) vorgeschriebenen Zeit ist jedoch der Übergang zum Ton der nächsten Note erforderlich, d.h. im Programm muß irgendwie die seit Beginn der Erzeugung des entsprechenden Tones vergangene Zeit erkennbar sein. Dazu würde sich prinzipiell folgendes Verfahren anbieten: Die Anzahl der ausgegebenen Halbperioden wird gezählt und bei Erreichen eines bestimmten Wertes erfolgt der Übergang zur nächsten Note. Die auftretende Schwierigkeit ist offensichtlich die, daß die Anzahl der abzuwartenden Halbperioden von der Tonhöhe abhängt. Damit wäre bei jeder Note erst eine Berechnung der Zählgröße erforderlich. Um diese Berechnung möglichst einfach zu halten, müßte eine Tabelle mit entsprechenden Konstanten für die einzelnen Tonhöhen vorhanden sein. In diesem Beispiel wird eine andere Lösung gewählt. Es wird eine zweite unabhängige Zeitbasis in Form eines CTC-Kanals eingeführt. Dazu wird der CTC-Kanal 1 als Timer (Zeitgeber) programmiert. Durch die Wahl der Zeitkonstanten liefert der Timer einen Takt mit einer Periode von etwa 17 Millisekunden. Die Nutzung dieser Basis zur Realisierung der richtigen Tondauer kann auf zwei verschiedene Weisen erfolgen:

1. Der Timer erzeugt Unterbrechungen und bewirkt den Übergang zur nächsten Note. Hierbei tritt jedoch folgender Nachteil auf: Durch die Unterbrechungen ist nicht mehr gewährleistet, daß die Dauer einer Halbperiode exakt stimmt, d.h. die seitliche Transparenz geht verloren. Das wäre nicht schlimm, wenn durch die Unterbrechung generell ein Übergang zur nächsten Note stattfinden würde. Das ist jedoch nicht gegeben, da sich die erforderlichen Tondauern nicht direkt durch Programmierung des CTC erreichen lassen. Die Zählkapazität des CTC muß also in der Software erweitert werden. Damit bewirkt dann aber nicht jede CTC-Unterbrechung einen Übergang zur jeweils nächsten Note, so daß der beschriebene Effekt zu erwarten ist.

2. Der Timer wird zu bestimmten Zeitpunkten abgefragt (z.B. nach einer Halbperiode des Tones) und es werden Nulldurchgänge durch Vergleich mit abgespeicherten Werten erkannt. Das Erkennen dieser Nulldurchgänge stellt praktisch einen unabhängigen Takt dar, der zur Ermittlung der Tondauer verwendet werden kann. Die dabei auftretenden unterschiedlichen Laufzeiten in den einzelnen Zweigen (Nulldurchgang erkannt oder nicht) lassen sich durch Zeitschleifen kompensieren, so daß die seitliche Transparenz voll erhalten bleibt (Bild 10.8.).

Aus dem genannten Gründen und wegen der einfacheren Realisierung wurde der 2. Variante der Vorzug gegeben. Es soll nicht verschwiegen werden, daß die Nachteile der 1. Variante gegenstandslos werden, wenn zwei CTC-Kanäle in Reihe geschaltet werden (1. Kanal als Zeitgeber ohne Unterbrechung, 2. Kanal als Zähler der Ausgangsimpulse des 1. Kanals mit Unterbrechungen).

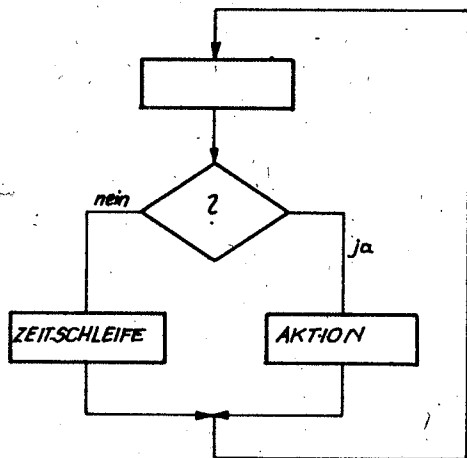


Bild 10.8. Erhaltung der Zeittransparenz durch gleiche Laufzeiten in allen Zweigen

**2.5. Darstellung von Noten im Rechner**

Eine Note enthält Informationen über

- Tonhöhe und
- Tondauer.

Der Sonderfall "Pause" kann als spezielle Tonhöhe aufgefaßt werden.

Analog ist eine Kodierung als Endezeichen zu vereinbaren, wenn nicht die Länge der Melodie am Anfang (als Kopfinformation) erscheinen soll, was aber unpraktischer wäre.

Es ist festzustellen, daß 5 Tondauern (ganze,  $\frac{1}{2}$ ,  $\frac{1}{4}$ ,  $\frac{1}{8}$ ,  $\frac{1}{16}$ ) und 37 Tonhöhen (3Oktaven) zu kodieren sind.

Damit ist eine Kodierung in einem Byte nicht möglich. 2 Byte sind dagegen für eine Note ausreichend. Zweckmäßigerweise wird 1 Byte für die Tondauer und 1 Byte für die Tonhöhe reserviert. Im Speicher wird für die 2 Bytes einer Note willkürlich folgende Reihenfolge festgelegt:

1. Tondauer
2. Tonhöhe

Darauf kann die nächste Note folgen. Für die Kodierung der Tondauer bietet sich folgende einfache Lösung an: Das Byte enthält direkt die Anzahl der CTC-Takte für die Tondauer (siehe 2.4.). Dabei ergibt sich folgende Zuordnung:

ganze Note :	COH
$\frac{1}{2}$ Note :	60H
$\frac{1}{4}$ Note :	30H
$\frac{1}{8}$ Note :	18H
$\frac{1}{16}$ Note :	0CH
$\frac{1}{32}$ Note :	06H

Zusätzlich wird festgelegt, daß der Kode 00H dem Schlußzeichen entspricht.

Für die Kodierung der Tonhöhe wäre analog eine direkte Interpretation des 2. Bytes als Zählwert für die Zeitschleife denkbar. Es zeigt sich jedoch, daß sich die beiden erforderlichen Zeitschleifenwerte nicht in einem Byte unterbringen lassen. Ein Ausweg wäre die Verwendung von 2 Bytes für die Kodierung der Tonhöhe. In diesem Programmierbeispiel wurde eine andere Lösung gewählt: Es wird eine relativ willkürliche Zuordnung der Tonhöhen zu Codes durchgeführt, wobei gilt, daß sich der Kode des nächsthöheren Tones aus dem des tieferen Tones durch Addition von 2 ergibt. Für die Implementierung ist damit der Aufbau einer Tabelle erforderlich, die für jede Tonhöhe die 2 Werte für die Zeitschleifen enthält.

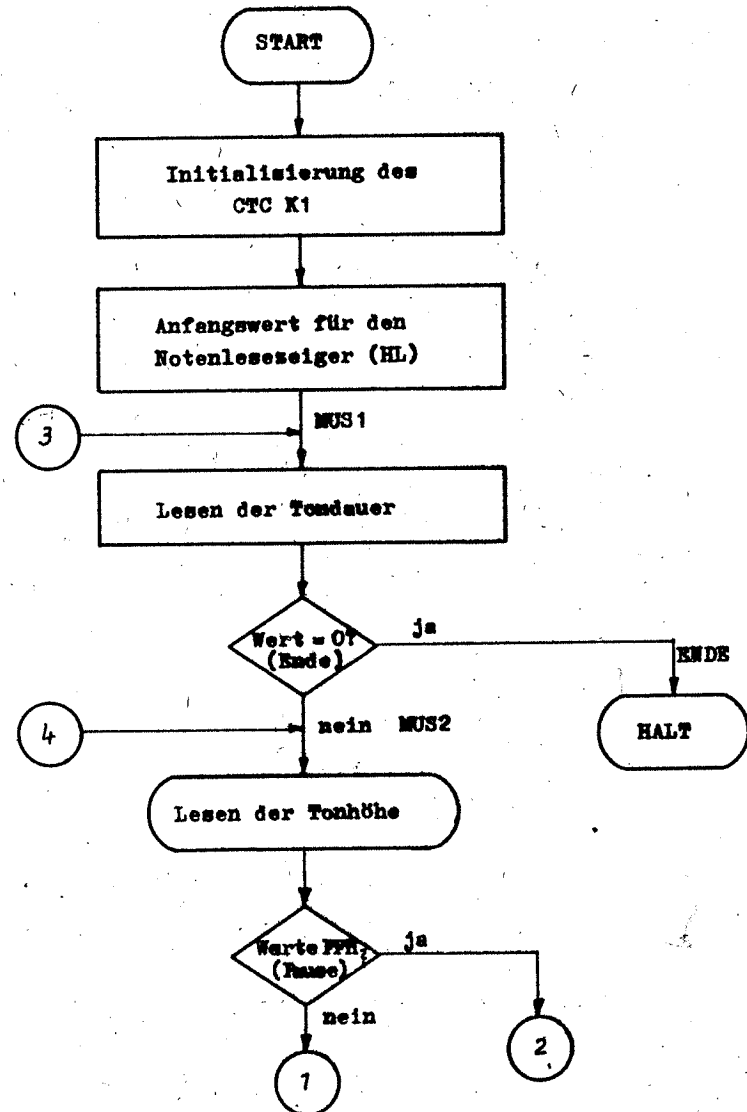
Folgende Kodierungen wurden für die Tonhöhe gewählt:

- a : 00H
- ais : 02H
- h : 04H
- o' : 06H
- cis' : 08H
- d' : 0AH
- dis' : 0CH
- e' : 0EH
- f' : 10H
- fis' : 12H
- g' : 14H
- gis' : 16H
- a' : 18H
- ais' : 1AH
- ...
- gis'' : 46H
- a''' : 48H
- Pause : FFH

Die Tabelle mit den Zeitschleifenwerten ist natürlich sinnvollerweise so aufzubauen, daß eine Adressierung mit den angegebenen Codes direkt möglich ist. Für jede Note sind 2 Byte als Zeitschleifenwerte erforderlich. Die Tabelle beginnt also mit den Werten für die Note a und endet mit den Werten für die Note a'''. Die direkte Adressierung dieser Tabelle ist möglich, da für die Codes der Tonhöhen nur gerade Zahlen verwendet werden, so daß mit jedem Code 2 aufeinanderfolgende Bytes adressierbar sind. Die Zeitschleifenwerte sind dem angefügten Programm ab Adresse 4080H (Werte für A) zu entnehmen.

3. Das Programm "Musik"

Der Programmablaufplan ist in Bild 10.9. dargestellt.



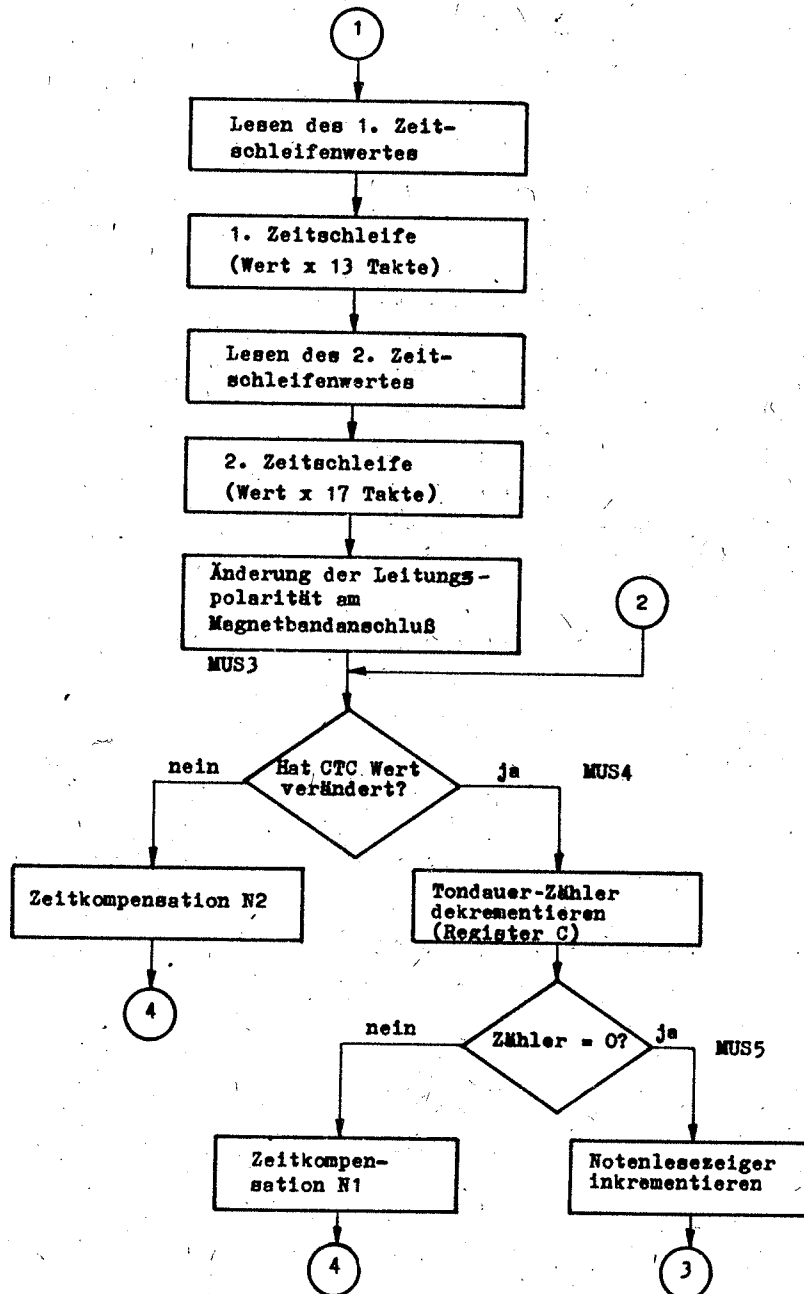
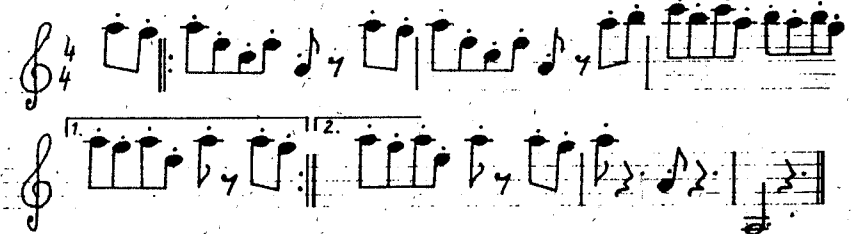


Bild 10.9. Programmablaufplan für Programm Musik

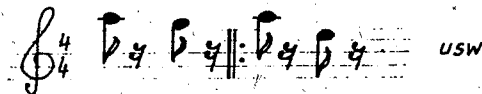
Die dabei angeführten symbolischen Marken sind mit denen im angefügten Programm "Musik" identisch. Die Abspeicherung der Noten beginnt auf der Adresse 4100H (durch Initialisierung für Lesenzeiger HL festgelegt). Die verwendeten Kodierungen entsprechen genau den oben vereinbarten (1. Byte: Tondauer, 2. Byte: Tonhöhe). Es ist zu bemerken, daß durch die Zeitkompensationen N1 und N2 für alle Alternativen im Programm gleiche Laufzeiten erreicht wurden. Eine Ausnahme wird nur bei Erkennung einer Pause gemacht, wo ja die Zeittransparenz nicht erforderlich ist, da ohnehin kein Ton erzeugt wird. Es ist zu empfehlen, wenigstens für eine Note die entsprechende Anzahl von Maschinentakten für eine Halbperiode zu errechnen und mit der unter 2.1. angegebenen Anzahl von Maschinentakten zu vergleichen. Im Folgenden wird die Kodierung für eine Melodie angegeben, mit der das Programm ausprobiert werden kann.

Beispiel:

Dem "Elektronikklang" des POLY-COMPUTERS Rechnung tragend wurde als Beispiel die Melodie "Pop corn" ausgewählt (Ausschnitte daraus), deren Notenzeile nachfolgend aufgeführt ist.



Punktierte Noten sind staccato ("abgehackt") zu spielen, so daß im Gegensatz zur obigen Notation folgendes programmiert werden muß:



D.h. zwischen zwei  $\frac{1}{16}$  Noten ist eine  $\frac{1}{16}$  Pause einzufügen.

Die ab Adresse 4100H einzutragende Notentabelle sieht demnach folgendermaßen aus: (bei der Eingabe kann aufgrund der Wiederholungen das **ME** -Kommando sinnvoll genutzt werden).

Adressen (hex)				Inhalt (hex) (Dauer, Tonhöhe)
4100	411E	417A	4198	0C,30
4102	4120	417C	419A	0C,FF
4104	4122	417E	419C	0C,2C
4106	4124	4180	419E	0C,FF
4108	4126	4182	41A0	0C,30
410A	4128	4184	41A2	0C,FF
410C	412A	4186	41A4	0C,26
410E	412C	4188	41A6	0C,FF
4110	412E	418A	41A8	0C,1E
4112	4130	418C	41AA	0C,FF
4114	4132	418E	41AC	0C,26
4116	4134	4190	41AE	0C,FF
4118	4136	4192	41B0	0C,18
411A	4138	4194	41B2	0C,FF
411C	413A	4196	41B4	18,FF
	413C		41B6	0C,30
	413E		41B8	0C,FF
	4140		41BA	0C,34
	4142		41BC	0C,FF
	4144		41BE	0C,36
	4146		41C0	0C,FF
	4148		41C2	0C,34
	414A		41C4	0C,FF
	414C		41C6	0C,36
	414E		41C8	0C,FF
	4150		41CA	0C,30
	4152		41CC	0C,FF
	4154		41CE	0C,34
	4156		41D0	0C,FF
	4158		41D2	0C,30
	415A		41D4	0C,FF
	415C		41D6	0C,34
	415E		41D8	0C,FF
	4160		41DA	0C,2C
	4162		41DC	0C,FF
	4164		41DE	0C,30
	4166		41E0	0C,FF
	4168		41E2	0C,2C
	416A		41E4	0C,FF
	416C		41E6	0C,30

Adressen (hex)		Inhalt (hex) (Dauer, Tonhöhe)
416E	41E8	0C,FF
4170	41EA	0C,26
4172	41EC	0C,FF
4174	41EE	0C,30
4176	41F0	0C,FF
4178	41F2	18,FF
	41F4	0C,30
	41F6	0C,FF
	41F8	0C,2C
	41FA	0C,FF
	41FC	0C,30
	41FE	54,FF
	4200	0C,18
	4202	54,FF
	4204	90,00
	4206	00

4. Schlußfolgerungen

Es konnte gezeigt werden, daß die Aufgabenstellung prinzipiell lösbar ist. Dabei mußten allerdings einige Einschränkungen gemacht werden (größere Toleranzen bei höheren Tönen, monophone Musik, reine Rechteckschwingung).

Der Rechner wurde mit der Erzeugung der Ausgangssignale am Magnetbandanschluß voll ausgelastet. Es war nicht mehr möglich, Unterbrechungen zu gestatten, da diese die Töne zu stark verfälscht hätten.

Die "Intelligenz" des Rechners wurde kaum genutzt. Die Interpretation des Speicherinhaltes wäre auch durch eine einfache Folgesteuerung möglich gewesen (bei größerer Genauigkeit).

Die Programmierung war recht kompliziert, da die Laufzeiten ganz genau berechnet werden mußten. Insbesondere mußte dieser Prozeß bei jeder geringfügigen Änderung erneut durchgeführt werden.

Abschließend wird deshalb festgestellt, daß die eigentliche Ton-erzeugung besser durch spezielle programmierbare Teiler ausgeführt werden sollte. Der Oberwellengehalt könnte durch (programmierbare) Filter geeignet beeinflußt werden. Der Rechner selbst würde die Steuerung des Teilers übernehmen. Außerdem wäre der Rechner dann zeitlich so weit entlastet, daß er gleichzeitig eine größere Anzahl von Teilern bedienen könnte, so daß auch polyphone Musik möglich wird.

SEITE 1  
POLY-880 ASM 1.0

MUSIK  
QUELLANWEISUNG

ADR	OBJ-KODE	NR	QUELLANWEISUNG	MUSIK
4000		1	*****	*****
4000		2	*****	*****
4002		3	*****	*****
4004		4	*****	*****
4006		5	*****	*****
4008		6	MUSIKPROGRAMM	
4008		7	*****	*****
4008		8	*****	*****
400C		9	*****	*****
400C		10	*****	*****
400D		11	*****	*****
400F		12	*****	*****
4011		13	*****	*****
4011		14	*****	*****
4012		15	*****	*****
4013		16	*****	*****
4013		17	*****	*****
4015		18	*****	*****
4017		19	*****	*****
4018		20	*****	*****
401A		21	*****	*****
401C		22	*****	*****
401D		23	*****	*****
401E		24	*****	*****
4020		25	*****	*****
4021		26	*****	*****
4021		27	*****	*****
4022		28	*****	*****

```

4000 ORG
4000 LD A,25H
4002 OUT CTC1
4004 LD A,40H
4006 CTC1
4008 LD HL,4100H
4008 LD A,<HL>
400C HL
400D INC HL
400D CMP 00H
400F JR2 ENDE
4011 LD C,A
4011 LD A,<HL>
4013 CMP 0FFH
4015 JR2 MUS3
4017 PUSH HL
4018 LD H,40H
401A ADD 80H
401C LD L,A
401D LD B,<HL>
401E X B,<HL>
4020 DJNZ HL
4021 INC LD
4021 LD B,<HL>
4022 NOP

```

```

; CTC-KANAL 1 ALS TIMER PROGRAMMIEREN
; ZUR BESTIMMUNG DER TONDAUER
; ZEITKONSTANTE FUER 1/128 SEKUNDE
;
; ZEIGER ZUM LESEN DER "NOTEN"
; TONDAUER LADEN
;
; ENDE DER MELODIE?
; FERTIG!
; DAUER DES TONES IN C
; KODE FUER DIE TONHOEHE
; PAUSE?
; PAUSE!
;
; ZEIGER FUER TONHOEHENTABELLE
;
; ZEITKONSTANTE M13
; ZEITSCHLEIFE Z13
;
; ZEITKONSTANTE M17

```



4023	10FD	29		DJNZ	MUSZ	
4025	E1	30		POP	HL	; ZEIGER FUER "NOTEN"
4026	D882	31		IN	PI002	; MAGNETBANDANSCHLUSS
4028	EE04	32		XOR	04H	; T/2 VERGANGEN
402A	D382	33		OUT	PI002	
402C	D889	34	MUS3	IN	CTC1	; TONDAUER
402E	AA	35		XOR	D	; VERGLEICH MIT LETZTEM WERT
402F	CB6F	36		BIT	5, A	; AENDERUNG?
4031	2006	37		JRNZ	MUS4	; JA!
4033	0605	38		LD	B, N2	; LAUFZEITKOMPENSATION
4035	10FE	39		DJNZ	x	
4037	1809	40		JR	MUS2	; NAECHSTES T/2
4039	AA	41	MUS4	XOR	D	
403A	57	42		LD	D, A	; CTC-WERT IN D ABSPEICHERN
403B	00	43		DEC	C	; TONDAUERZAEHLER
403C	2006	44		JRZ	MUS5	; FERTIG MIT DIESER "NOTE"
403E	0603	45		LD	B, N1	; LAUFZEITKOMPENSATION
4040	10FE	46		DJNZ	x	
4042	18CE	47		JR	MUS2	
4044	23	48	MUS5	INC	HL	
4045	18C4	49		JR	MUS1	
4047	76	50	ENDE	HALT		; FERTIG MIT INTERPRETATION DER "NOTEN"
		51	CTC1	EQU	89H	; ADRESSE CTC-KANAL 1
		52	PI002	EQU	82H	; ADRESSE MAGNETBANDANSCHLUSS
		53	N1	EQU	3	
		54	N2	EQU	5	
		55				
		56				TABELLE DER TONHOEHEN
		57				
4080		58	ORG	4080H		

- 238 -

ADR	OBJ-KODE	NR	MUSIK QUELLANWEISUNG
4080	8C	59	DB 8CH
4081	04	60	DB 04H
4082	83	61	DB 83H
4083	04	62	DB 04H
4084	70	63	DB 70H
4085	0C	64	DB 0CH
4086	75	65	DB 75H
4087	02	66	DB 02H
4088	5F	67	DB 5FH
4089	00	68	DB 00H
408A	61	69	DB 61H
408B	06	70	DB 06H
408C	56	71	DB 56H
408D	0A	72	DB 0AH
408E	4C	73	DB 4CH
408F	0C	74	DB 0CH
4090	53	75	DB 53H
4091	02	76	DB 02H
4092	4C	77	DB 4CH
4093	03	78	DB 03H
4094	44	79	DB 44H
4095	05	80	DB 05H
4096	3B	81	DB 3BH
4097	08	82	DB 08H
4098	31	83	DB 31H
4099	0C	84	DB 0CH
409A	33	85	DB 33H
409B	07	86	DB 07H

- 239 -

409C	34	87	DB	34H
409D	03	88	DB	03H
409E	2C	89	DB	2CH
409F	06	90	DB	06H
40A0	22	91	DB	22H
40A1	0A	92	DB	0AH
40A2	25	93	DB	25H
40A3	05	94	DB	05H
40A4	20	95	DB	20H
40A5	07	96	DB	07H
40A6	22	97	DB	22H
40A7	03	98	DB	03H
40A8	1F	99	DB	1FH
40A9	03	100	DB	03H
40AA	13	101	DB	13H
40AB	0A	102	DB	0AH
40AC	0F	103	DB	0FH
40AD	08	104	DB	08H
40AE	13	105	DB	13H
40AF	06	106	DB	06H
40B0	0E	107	DB	0EH
40B1	08	108	DB	08H
40B2	11	109	DB	11H
40B3	04	110	DB	04H
40B4	07	111	DB	07H
40B5	0A	112	DB	0AH
40B6	01	113	DB	01H
40B7	0D	114	DB	0DH
40B8	07	115	DB	07H
40B9	07	116	DB	07H

ADR	OBJ-KODE	NR	MUSIK QUELLANWEISUNG
40BA	0D	117	DB 0DH
40BB	01	118	DB 01H
40BC	0A	119	DB 0AH
40BD	02	120	DB 02H
40BE	02	121	DB 02H
40BF	07	122	DB 07H
40C0	03	123	DB 03H
40C1	05	124	DB 05H
40C2	03	125	DB 03H
40C3	04	126	DB 04H
40C4	03	127	DB 03H
40C5	03	128	DB 03H
40C6	03	129	DB 03H
40C7	02	130	DB 02H
40C8	03	131	DB 03H
40C9	01	132	DB 01H
		133	END

SEITE 3  
POLY-880 ASM 1.0

Hersteller:



VEB Kombinat Polytechnik und  
Präzisionsgeräte Karl-Marx-Stadt  
Stammbetrieb VEB Polytechnik  
Karl-Marx-Stadt

DDR - 9023 Karl-Marx-Stadt  
Melanchthonstraße 4/8, PSF 93

Deutsche Demokratische Republik

Exporteur:

**MLW intermed-export-import**

Volkseigener Außenhandelsbetrieb der  
Deutschen Demokratischen Republik  
DDR - 1020 Berlin, Schicklerstraße 5/7, PSF 17  
Deutsche Demokratische Republik