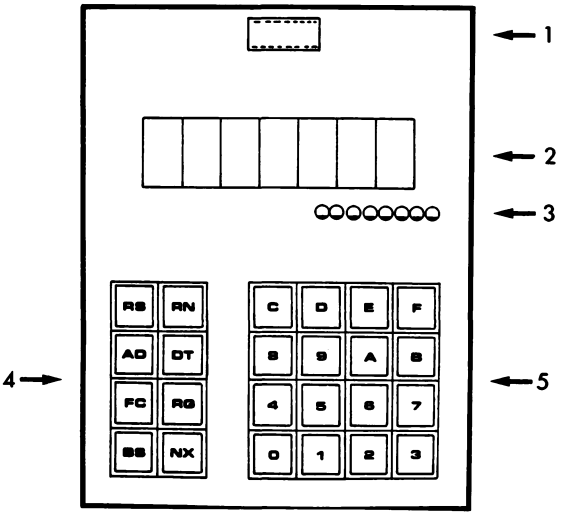



WICHTIGER HINWEIS: Das Computersystem CCS 85 wird als Bausatz ohne Gehäuse geliefert. Auf der Rechnerplatine sind Leiterbahnen frei zugänglich, die die Netzspannung von 220 V führen. Zumindest die Rechnerplatine muß deshalb in ein vollständig isolierendes Kunststoffgehäuse eingebaut werden, so daß der Berührungsschutz gewährleistet ist. Ein Betrieb ohne ausreichende Schutzmaßnahmen ist nicht zulässig!

1 DIL-Fassung zur Verbindung der Tastaturplatine mit der Rechnerplatine über das mitgelieferte Flachkabel. Bei der Rechnerplatine muß das Flachkabel so eingesteckt werden, daß es neben dem E/A-Baustein 8255 vorbeiläuft.

2 Siebenstellige Siebensegmentanzeige zur Anzeige von Adressen und Daten. Die vier linken Stellen zeigen Adressen an, die beiden rechten Stellen zeigen die unter der angezeigten Adresse gespeicherten Daten. Nach Einschalten des Gerätes erscheint 0800 XX Hierin ist 0800 die erste Adresse des Schreib/Lese-Speichers (RAM) und XX der - nach dem Einschalten zufällige und daher nicht definierte - Inhalt.

3 Acht Leuchtdioden (LED's) zur Anzeige von Daten im Dualcode. Sie werden wie ein Port angesprochen. Der entsprechende Ausgabebefehl ist OUT 00.

4 Acht Funktionstasten zur Kommunikation mit dem System über den Monitor. Diese Tasten haben im einzelnen bei Betätigung folgende Wirkung:

RS RESET. Diese Taste hat dieselbe Wirkung, wie wenn das Gerät eingeschaltet wird. Es wird ein definierter Anfangszustand hergestellt. Alle Register im E/A-Baustein werden gelöscht, und alle Ports werden als Eingänge definiert. Der Monitor startet bei Adresse 0000 und wartet auf Betätigung einer Taste.

RN RUN. Es wird mit der Ausführung des Programms bei der Adresse begonnen, die im Adressenfeld erscheint.

AD ADDRESS. Neben der vierten Ziffer im Adressenfeld erscheint ein leuchtender Punkt. Das bedeutet, daß über die Hexadezimaltastatur nun eine Adresse eingegeben werden kann. Das geschieht durch Eintippen der vier Ziffern, die der gewünschten Adresse entsprechen.

DT DATA. Ganz rechts im Datenfeld der Anzeige erscheint ein leuchtender Punkt. Das bedeutet, daß über die Hexadezimaltastatur nun Daten eingegeben werden können, die unter der links erscheinenden Adresse abgespeichert werden sollen. Die Übernahme der Daten in die Speicherzelle erfolgt jedoch erst bei Betätigen der Taste NX (siehe dort).

FN FUNCTION. Mit dieser Taste ist ein vereinfachter Start von Programmen möglich. In diesem Fall wird nicht die Startadresse und darauffolgend RUN eingegeben, sondern es wird die Tastenfolge FN N NX gedrückt. Dabei bedeutet N eine Hexadezimalziffer von 0 bis 7. Es ist also auf diese Weise möglich, acht verschiedene Programme anzustarten. Die zugehörigen Anfangsadressen sind im Monitor in den Zellen 03D0 bis 03DF abgelegt.

N=0 ist bereits belegt, und zwar für die Kassetten-Ausgabe-Routine (Anfangsadresse 0400). N=1 ist für die Kassetten-Eingaberoutine (Anfangsadresse 0487) reserviert. Diese beiden Adressen findet man also in den Zellen 03D0 bis 03D3. In den restlichen 12 Zellen steht FF; hier können beliebige Anfangsadressen von Eigenprogrammen einprogrammiert werden.

RG REGISTER. Bei Drücken dieser Taste erscheint L (für L-Register) und im Datenfeld der Inhalt dieses Registers. Jeweils bei Betätigung der Taste NX wird nun der Kennbuchstabe eines anderen Registers zusammen mit dessen Inhalt dargestellt, und zwar in der Reihenfolge L - H - A - B - C - D - E - F. (F steht für flag-word, Prozessorzustandswort. Siehe Beschreibung des Befehls PUSH PSW.) Es ist möglich, den Inhalt eines bestimmten Registers zu ändern. Dazu muß zunächst die Taste DT gedrückt werden. Dann können die Daten eingegeben und durch Betätigen von NX in das Register gebracht werden.

BS BACKSTEP. Die im Adressenfeld erscheinende Anzeige wird um 1 vermindert. Der Inhalt der neuen Adresse wird im Datenfeld angezeigt.

NX NEXT. Diese Taste führt eine Doppelfunktion aus:

1) Die im Datenfeld dargestellten Daten werden unter der im Adressenfeld dargestellten Adresse bzw. in dem dort angegebenen Register abgespeichert.

2) Die Adresse im Adressenfeld wird automatisch um 1 erhöht (bzw. es wird automatisch auf das nächste Register umgeschaltet), und im Datenfeld erscheint der zugehörigr Inhalt.

5 Hexadezimales Tastenfeld mit den Tasten von 0 bis F zur Eingabe von Daten oder Adressen in hexadezimal codierter Form.

Benutzung der Routinen für das Kassetten-Interface

AUSGABE:

1) Anfangs- und Endadresse des auszugebenden Speicherbereichs in die Zellen 0BFB/C und 0BFD/E laden.

2) Bei angeschlossenem, jedoch in Ruhe befindlichem Tonbandgerät das Ausgabeprogramm starten, indem die Tastenfolge FN 0 NX gedrückt wird. Start über AD 0 4 0 0 RN ist natürlich auch möglich.

3) Das Ausgabeprogramm sendet nun einen 10 Sekunden langen Vorspann (hoher Dauerton). In diesem Zeitraum muß das Tonbandgerät in Stellung „Aufnahme" gestartet werden. Die Anzeige meldet sich nach beendeter Ausgabe mit 0800 XX wieder.

EINGABE:

1) Anfangs- und Endadresse wie oben beschrieben eingeben. Es müssen nicht unbedingt die gleichen Adressen sein wie bei der Ausgabe; jedoch Vorsicht, wenn das Programm Sprungadressen enthält!

2) Tonbandgerät in Stellung „Wiedergabe" im Bereich des Vorspanns (hoher Dauerton) starten.

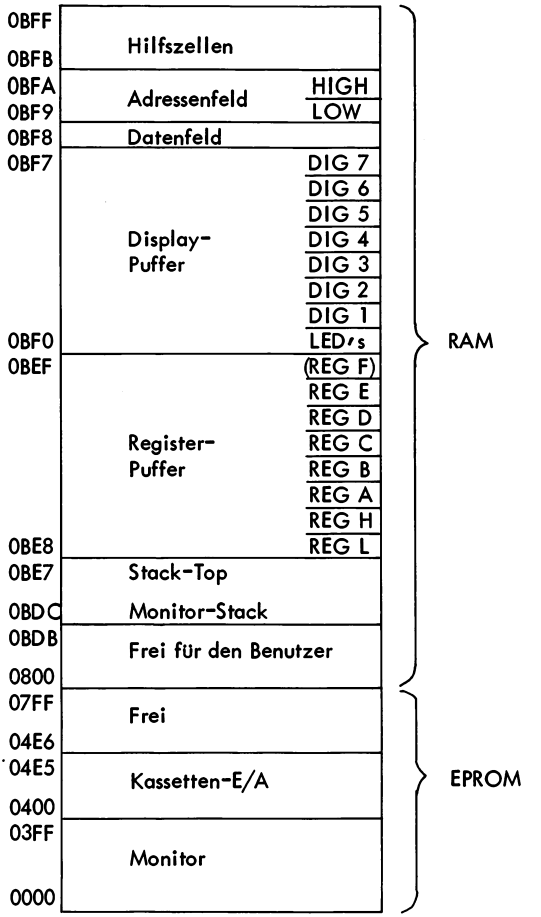
3) Sofort, jedenfalls auch innerhalb des Vorspanns, die Tastenfolge FN 1 NX drücken (oder AD 0 4 8 7 RN). Nach beendeter Eingabe meldet sich der Monitor wieder.

Es ist offensichtlich ratsam, bei der Ausgabe einen möglichst langen Vorspann aufzuzeichnen.

Das Ausgabeprogramm erzeugt eine Prüfsumme der ausgegebenen Daten und zeichnet diese ebenfalls auf das Band auf. Dieser Wert wird vom Einleseprogramm geprüft. Liegt keine Übereinstimmung vor, dann erscheint „Err" (Error, Fehler) in der Anzeige. Der Einlesevorgang sollte dann wiederholt werden.

Die Datenübertragungsrate ist 25 Bytes pro Sekunde.

## Speicherorganisation




Die ersten der nachfolgenden Beispielprogramme sind für den Fall geschrieben, daß die Leitungen von Port A über Treiber mit Leuchtdioden verbunden sind, so daß der Zustand der Bits dieses Ports angezeigt werden kann. Man kann diese Programme aber auch leicht für die LED-Zeile auf der Tastaturplatine verwenden, indem man folgende kleine Änderung vornimmt:

1) Die Befehle
```
MVI A,80
OUT 07
```
entfallen.

2) Der Befehl
```
OUT 04
```
wird durch
```
OUT 00
```
ersetzt.

Programm Nr. 7 erfordert leichte Zugänglichkeit der Ports (Buchsen o.ä.)

Der Befehlssatz des 8085 umfaßt fünf verschiedene Arten von Befehlen:

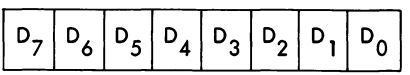
- Datentransportgruppe - Daten werden zwischen Registern oder zwischen Registern und Speicherplätzen bewegt.
- Arithmetische Gruppe - Daten in Registern oder in Speicherzellen werden addiert, subtrahiert, um 1 erhöht oder erniedrigt.
- Logische Gruppe - Daten in Registern oder Speicherzellen werden durch UND, ODER, EXKLUSIV ODER miteinander verknüpft oder sie werden verglichen,zyklisch verschoben oder komplementiert.
- Verzweigungsgruppe - Dies sind bedingte und unbedingte Sprungbefehle, Unterprogramm-Rufbefehle und Rücksprungbefehle.
- Stack, I/O- und Maschinenkontrollbefehle - Diese umfassen I/O-Befehle, Befehle zur Stackmanipulation und Befehle, die sich auf die Flags beziehen.

Wie bei den meisten Mikroprozessorsystemen ist der Speicher in Gruppen zu 8 bits organisiert. Eine solche Gruppe nennt man Byte. Jedes Byte hat seine spezifische 16-bit-Binäradresse, die seine Position in der Aufeinanderfolge der Speicherzellen im Speicher angibt.

Im 8085-System können direkt bis zu 65536 Speicherzellen (zu je 1 Byte) adressiert werden. Der Speicher kann sowohl aus Nur-Lese-Speicher (Read only memory, ROM) als auch aus Schreib/Lese-Speicher (Random access memory, RAM) oder aus beiden Typen bestehen.

Daten in einem 8085-System werden in der Form eines ganzzahligen 8-bit-Wortes gespeichert:

Datenwort

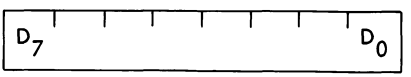



MSB LSB

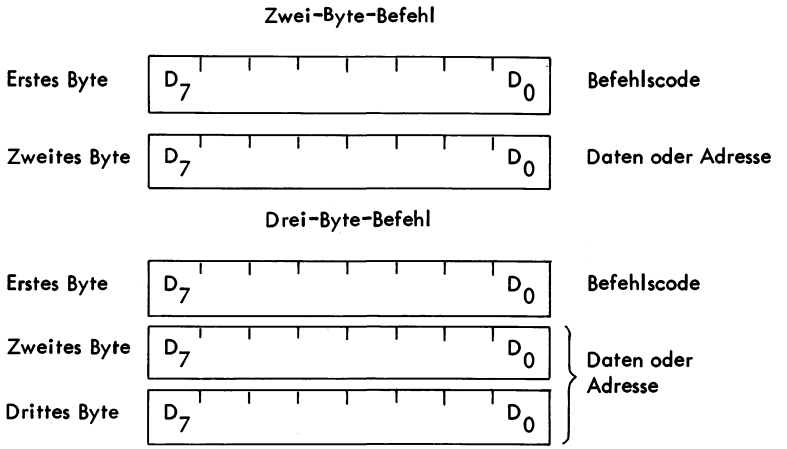
Datenbit 0 wird als das niederwertigste bit (LSB, least significant bit) bezeichnet und Datenbit 7 als das höchstwertigste bit (MSB, most significant bit).

8085-Befehle können ein, zwei oder drei Byte lang sein. Befehle, die aus mehreren Bytes bestehen, müssen in aufeinanderfolgenden Speicherzellen abgespeichert sein; die Adresse des ersten Byte wird immer als die Adresse des Befehls betrachtet.

Ein-Byte-Befehl




Befehlscode




Die zu verarbeitenden Daten sind oft im Speicher abgelegt. Wenn Daten aus mehreren Bytes bestehen, müssen sie, wie Befehle, in aufeinanderfolgenden Zellen untergebracht sein. Dabei kommt zuerst das niedrigstwertige Byte, gefolgt von den weiteren Bytes in aufsteigender Wertigkeit. Der 8085 kennt vier verschiedene Arten, die in Speicherzellen oder Registern abgelegten Daten zu adressieren:

- Direkt: Byte 2 und 3 des Befehls enthalten die genaue Speicheradresse des dem Befehl zu unterziehenden Datenwortes. Byte 2 enthält das niederwertige Adressenbyte, Byte 3 das höherwertige.
- Register: Der Befehl gibt das Register oder das Registerpaar an, in dem sich die Daten befinden.
- Register indirekt: Der Befehl gibt ein Registerpaar an, welches die Speicheradresse enthält, unter der die gewünschten Daten zu finden sind. Das höherwertige Byte der Adresse ist im ersten Register des Paares, das niederwertige Byte im zweiten.
- Sofort (engl. immediate): Der Befehl enthält die Daten selbst, entweder ein 8-bit-Wort oder ein 16-bit-Wort, und zwar erst das niederwertige Byte und dann das höherwertige Byte.

Sofern keine Unterbrechung (Interrupt) oder ein Sprungbefehl vorliegt, werden die Befehle so ausgeführt, wie sie in aufeinanderfolgenden Speicherzellen abgelegt sind. Ein Sprungbefehl kann die Adresse des als nächstes auszuführenden Befehls auf zweierlei Weise angeben:

- Direkt: Der Sprungbefehl enthält die Adresse des als nächstes auszuführenden Befehls. (Abgesehen von dem RST-Befehl enthält Byte 2 den niederwertigen Adreßteil und Byte 3 den höherwertigen Adreßteil.)
- Register indirekt: Der Sprungbefehl gibt ein Registerpaar an, welches die Adresse des als nächstes auszuführenden Befehls enthält. Der höherwertige Teil der Adresse ist im ersten Register und der niederwertige Teil im zweiten Register des Paares.

Der RST-Befehl ist ein Ein-Byte-Befehl, der meistens bei der Interrupt-Verarbeitung verwendet wird. Er enthält eine spezielle Gruppe von 3 bits. Der Programmablauf wird bei demjenigen Befehl fortgesetzt, dessen Adresse dem achtfachen Inhalt der genannten Bitgruppe entspricht.

Bedingungsbits

Es gibt fünf Bedingungsbits („Flags"), die im Zusammenhang mit der Ausführung eines Befehls beeinflußt werden können. Es sind dies Null, Vorzeichen, Gerade, Übertrag und Hilfsübertrag (Zero, Sign, Parity, Carry, Auxiliary carry). Jedes Flag wird in der CPU durch ein Ein-bit-Register dargestellt. Ein Flag wird gesetzt, indem das zugehörige bit auf den Wert 1 gebracht wird, und gelöscht, indem das bit auf den Wert 0 gebracht wird.

Wenn nicht anders angegeben, beeinflußt ein Befehl ein Flag in der folgenden Weise:

Zero: Wenn die Ausführung eines Befehls den Wert 0 ergibt, wird dieses Flag gesetzt, sonst gelöscht.

Sign: Wenn das höchstwertige bit des Ergebnisses einer Operation den Wert 1 hat, wird dieses Flag gesetzt, sonst gelöscht.

Parity: Wenn die modulo 2-Summe des Ergebnisses einer Operation den Wert 0 hat (d.h. wenn das Ergebnis gerade Parität hat), wird dieses Flag gesetz. Es wird gelöscht, wenn das Ergebnis ungerade Parität hat.

Carry: Falls die Ausführung eines Befehls einen Überlauf (carry) zur Folge hat (z.B. bei einer Addition) oder sich ein bit borgt (z.B. bei einer Subtraktion oder einem Vergleich), wird dieses Flag gesetzt, sonst gelöscht.

Auxiliary Carry: Wenn die Befehlsausführung im Ergebnis einen Übertrag von bit 3 nach bit 4 verursacht, wird dieses Flag gesetzt, sonst gelöscht. Dieses Flag wird beeinflußt von Additionen mit einfacher Genauigkeit, Subtraktionen, beim Inkrementieren und Dekrementieren, Vergleichen und bei logischen Operationen. Grundsätzlich wird dieses Flag jedoch bei Additionen benutzt und bei Inkrementierungen, die einem DAA-Befehl vorangehen.

Symbole und Abkürzungen

Die folgenden Symbole und Abkürzungen werden bei der Beschreibung des Befehlssatzes des 8085 verwendet:

| Symbol | Bedeutung |
|---|---|
| Akku | Register A |
| Adr | 16-bit-Adresse |
| Daten | 8-bit-Daten |
| Daten 16 | 16-bit-Daten |
| Byte 2 | Das zweite Byte des Befehls |
| Byte 3 | Das dritte Byte des Befehls |

| Symbol | Bedeutung |
|---|---|
| Port | 8-bit-Adresse eines I/O-Bausteins |
| r, r1, r2 | Eines der Register A,B,C,D,E,H,L |
| DDD, SSS | Das Bitmuster, welches eins der Register A,B,C, D,E,H,L kennzeichnet. (DDD = destiation, Zielregister; SSS = source, Quellregister.) |

| DDD oder SSS | Register |
|---|---|
| 1 1 1 | A |
| 0 0 0 | B |
| 0 0 1 | C |
| 0 1 0 | D |
| 0 1 1 | E |
| 1 0 0 | H |
| 1 0 1 | L |
| 1 1 0 | eine Speicherzelle |

| | |
|---|---|
| rp | Eines der Registerpaare. B steht für das Registerpaar B,C, wobei B das höherwertige und C das niederwertige Register ist. In gleicher Weise steht D für DE, H für HL und SP für den 16-bit-Stapelzeiger (stack pointer). |
| RP | Das Bitmuster, welches eins der Registerpaare B, D, H oder SP kennzeichnet: |

| RP | Registerpaar |
|---|---|
| 0 0 | BC |
| 0 1 | DE |
| 1 0 | HL |
| 1 1 | SP |

| | |
|---|---|
| rh | Das erste (höherwertige) eines Registerpaares. |
| rl | Das zweite (niederwertige) Register eines Paares. |
| PC | Befehlszähler (program counter), 16 bit. PCH kennzeichnet die höherwertigen und PCL die niederwertigen 8 bits des PC. |
| SP | Das 16-bit-Stapelzeigerregister mit dem höherwertigen Byte SPH und dem niederwertigen SPL. |
| $r_m$ | Bit m des Registers r. Die bits haben von links nach rechts die Nummern 7 bis 0. |
| m | Eine Speicherzelle. |

| Symbol | Bedeutung |
|---|---|
| Z,S,P,C,AC | Die Bedingungs-Flags:<br>Zero<br>Sign<br>Parity<br>Carry<br>Auxiliary carry |
| ( ) | Der Inhalt des Registers oder der Speicherzelle, die von den Klammern eingeschlossen wird |
| ← | ... wird nach ... transportiert |
| ∧ | Logisches UND |
| ⊻ | EXKLUSIV ODER |
| ∨ | ODER |
| + | Addition |
| - | Subtraktion im Zweierkomplement |
| * | Multiplikation |
| ⟷ | ... wird mit ... ausgetauscht |
| ¯ | Einerkomplement |
| n | Die Restart-Kennziffer (0 bis 7) |
| NNN | Die Binärdarstellung der Restart-Kennziffer |

Format der Befehlsbeschreibung

Auf den folgenden Seiten wird der Befehlsvorrat des 8085 ausführlich beschrieben. Jeder Befehl wird wie folgt beschrieben:

1) Die mnemonische Abkürzung wird mit fetten Großbuchstaben angegeben.

2) Neben 1) steht die englische Befehls-Kurzbeschreibung in Klammern, darunter die deutsche Kurzbeschreibung.

3) Die nächste Zeile enthält die Wirkung des Befehls in symbolischer Darstellungsweise.

4) Daran anschließend wird die Wirkung des Befehls ausführlich beschrieben.

5) Anschließend wird der Maschinencode als Bitmuster dargestellt.

6) Die letzten vier Zeilen enthalten zusätzliche Informationen zur Ausführung des Befehls. Zuerst wird die Anzahl der Maschinenzyklen und der zur Befehlsausführung erforderlichen Zustände (Taktimpulse) angegeben. Falls zwei Ausführungszeiten möglich sind, wie z.B. bei einem bedingten Sprung, werden beide Zeiten angegeben und durch einen Schrägstrich getrennt. Dann ist die Adressierungsart angegeben, und schließlich werden diejenigen Flags aufgeführt, die bei der Ausführung des Befehls beeinflußt werden können.

Datentransportbefehle:

## MOV r1, r2

(Move register)
Transportiere von Register zu Register

(r1) ← (r2)

Der Inhalt des Registers r2 wird in das Register r1 gebracht.

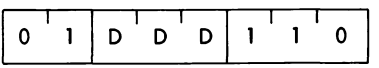

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | keine |

## MOV r, m

(Move from memory)
Lade Register aus dem Speicher

(r) ← ((H) (L))

Der Inhalt der Speicherzelle, deren Adresse im HL-Register steht, wird in das Register r geladen.



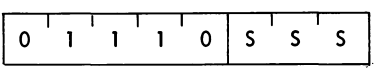

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | keine |

## MOV m, r

(Move to memory)
Bringe Registerinhalt in eine Speicherzelle

((H) (L)) ← (r)

Der Inhalt des Registers r wird in diejenige Speicherzelle gebracht, deren Adresse im HL-Register steht.



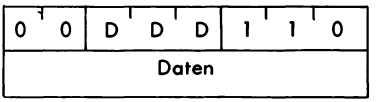

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | keine |

## MVI r, Daten

(Move immediate)
Lade direkt

(r) ← (Byte 2)

Der Inhalt von Byte 2 des Befehls wird in das Register r gebracht.



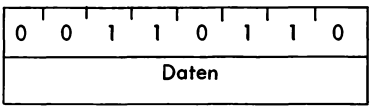

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | keine |

## MVI m, Daten

(Move to memory immediate)
Lade Speicherzelle direkt

((H) (L)) ← (Byte 2)

Der Inhalt von Byte 2 des Befehls wird in diejenige Speicherzelle gebracht, deren Adresse im HL-Register steht.



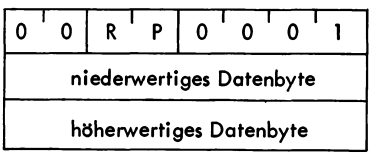

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | sofort/Register indirekt |
| Flags: | keine |

## LXI rp, Daten 16

(Load register pair immediate)
Lade Registerpaar direkt

(rh) ← (Byte 3), (rl) ← (Byte 2)

Byte 3 des Befehls wird in das höherwertige Register des Registerpaars rp gebracht; Byte 2 des Befehls wird in das niederwertige Register (rl) des Registerpaars rp gebracht.



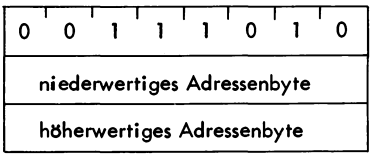

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | sofort |
| Flags: | keine |

## LDA Adr

(Load Accumulator direct)
Lade Akku direkt aus dem Speicher

(A) ← ((Byte 3) (Byte 2))

Der Inhalt derjenigen Speicherzelle, deren Adresse in Byte 2 und Byte 3 des Befehls angegeben ist, wird in den Akku gebracht.



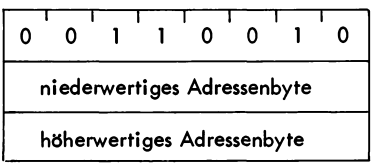

| | |
|---|---|
| Zyklen: | 4 |
| Zustände: | 13 |
| Adressierung: | direkt |
| Flags: | keine |

## STA Adr

(Store Accumulator direct)
Speichere Akkuinhalt direkt in die Speicherzelle

((Byte 3) (Byte 2)) ← (A)

Der Akkuinhalt wird in diejenige Speicherzelle gebracht, deren Adresse in Byte 2 und Byte 3 des Befehls angegeben ist.



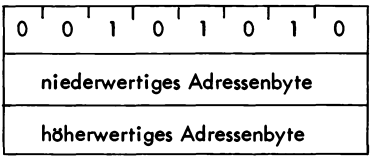

| | |
|---|---|
| Zyklen: | 4 |
| Zustände: | 13 |
| Adressierung: | direkt |
| Flags: | keine |

## LHLD Adr

(Load H and L direct)
Lade das HL-Registerpaar direkt aus dem Speicher

(L) ← ((Byte 3) (Byte 2)
(H) ← ((Byte 3) (Byte 2) + 1)

Der Inhalt der Speicherzelle, deren Adresse in Byte 2 und 3 des Befehls angegeben ist, wird in das L-Register gebracht. Der Inhalt der Speicherzelle mit der darauffolgenden Adresse wird in das H-Register gebracht.



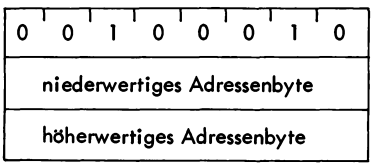

| | |
|---|---|
| Zyklen: | 5 |
| Zustände: | 16 |
| Adressierung: | direkt |
| Flags: | keine |

## SHLD Adr

(Store H and L direct)
Bringe den Inhalt des HL-Registers in den Speicher

((Byte 3) (Byte 2)) ← (L)
((Byte 3) (Byte 2) + 1) ← (H)

Der Inhalt des Registers L wird in diejenige Speicherzelle gebracht, deren Adresse in Byte 2 und 3 des Befehls angegeben ist. Der Inhalt des H-Registers wird in die darauffolgende Speicherzelle gebracht.



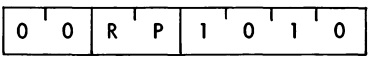

| | |
|---|---|
| Zyklen: | 5 |
| Zustände: | 16 |
| Adressierung: | direkt |
| Flags: | keine |

## LDAX rp

(Load Accumulator indirect)
Lade den Akku indirekt

(A) ← ((rp))

Der Inhalt derjenigen Speicherzelle, deren Adresse im Registerpaar rp enthalten ist, wird in den Akku geladen. ACHTUNG: Bei diesem Befehl dürfen nur die Registerpaare BC und DE verwendet werden!



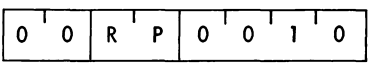

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | keine |

## STAX rp

(Store Accumulator indirect)
Speichere den Akkuinhalt indirekt

((rp)) ← (A)

Der Inhalt des Akkus wird in diejenige Speicherzelle gebracht, deren Adresse im Registerpaar rp steht. ACHTUNG: Nur Registerpaar BC oder DE darf verwendet werden!



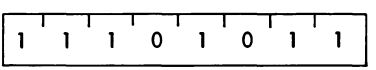

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | keine |

## XCHG

(Exchange H and L with D and E)
Vertausche H und L mit D und E

(H) ↔ (D)
(L) ↔ (E)

Der Inhalt der Register H und L wird mit dem Inhalt der Register D und E vertauscht.



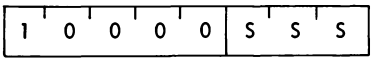

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | keine |

## ADD r

(Add register)
Addiere Register

(A) ← (A) + (r)

Der Inhalt des Registers r wird zum Akkuinhalt addiert. Das Ergebnis steht im Akku.



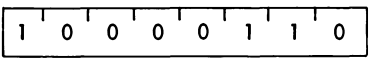

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,C,AC |

## ADD m

(Add memory)
Addiere Speicherzelle

(A) ← (A) + ((H) (L))

Der Inhalt der Speicherzelle, deren Adresse im HL-Register steht, wird zum Akkuinhalt addiert. Das Ergebnis steht im Akku.



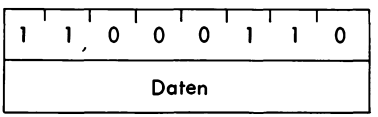

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,C,AC |

## ADI Daten

(Add immediate)
Addiere direkt

(A) ← (A) + (Byte 2)

Der Inhalt des zweiten Bytes des Befehls wird zum Akkuinhalt addiert. Das Ergebnis steht im Akku.



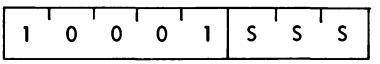

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | Z,S,P,C,AC |

## ADC r

(Add register with carry)
Addiere Register und Übertrag

(A) ← (A) + (r) + (C)

Der Inhalt des Registers r und das Carrybit werden zum Akkuinhalt addiert. Das Ergebnis steht im Akku.



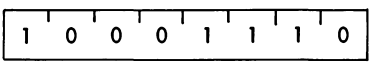

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,C,AC |

## ADC m

(Add memory with carry)
Addiere Speicherzelle und Übertrag

(A) ← (A) + ((H) (L)) + (C)

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, und das Carrybit werden zum Akkuinhalt addiert. Das Ergebnis steht im Akku.



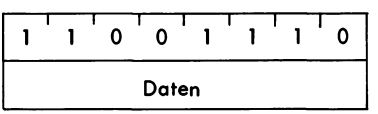

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | Z, S, P, C, AC |

## ACI Daten

(Add immediate with carry)
Addiere direkt mit Übertrag

(A) ← (A) + (Byte 2) + (C)

Der Inhalt des zweiten Bytes des Befehls und das Carrybit werden zum Akkuinhalt addiert. Das Ergebnis steht im Akku.



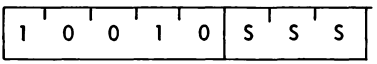

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | Z,S,P,C,AC |

## SUB r

(Subtract register)
Subtrahiere Register

(A) ← (A) - (r)

Der Inhalt des Registers r wird vom Akkuinhalt subtrahiert. Das Ergebnis steht im Akku.



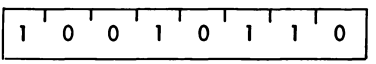

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,C,AC |

## SUB m

(Subtract memory)
Subtrahiere Speicherzelle

(A) ← (A) - ((H) (L))

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird vom Akkuinhalt subtrahiert. Das Ergebnis steht im Akku.



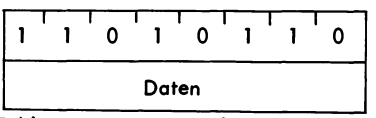

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,C,AC |

## SUI Daten

(Subtract immediate)
Subtrahiere sofort

(A) ← (A) - (Byte 2)

Der Inhalt des zweiten Bytes des Befehls wird vom Akkuinhalt subtrahiert. Das Ergebnis steht im Akku.



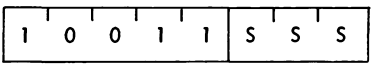

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | Z,S,P,C,AC |

## SBB r

(Subtract register with borrow)
Subtrahiere Register und Übertragsbit

(A) ← (A) - (r) - (C)

Der Inhalt des Registers r und das Carrybit werden vom Akkuinhalt subtrahiert. Das Ergebnis steht im Akku.



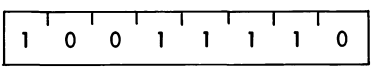

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,C,AC |

## SBB m

(Subtract memory with borrow)
Subtrahiere Speicherzelle und Übertragsbit

(A) ← (A) - ((H) (L)) - (C)

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, sowie das Carrybit werden vom Akkuinhalt subtrahiert. Das Ergebnis steht im Akku.



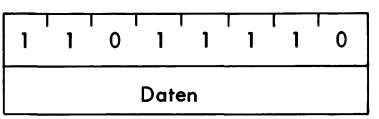

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,C,AC |

## SBI Daten

(Subtract immediate with borrow)
Subtrahiere sofort mit Übertrag

(A) ← (A) - (Byte 2) - (C)

Der Inhalt des zweiten Bytes des Befehls sowie das Carrybit werden vom Akkuinhalt subtrahiert. Das Ergebnis steht im Akku.



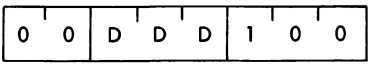

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | Z,S,P,C,AC |

## INR r

(Increment register)
Inkrementiere Register

(r) ← (r) + 1

Der Inhalt des Registers r wird um 1 erhöht. C wird nicht beeinflußt!



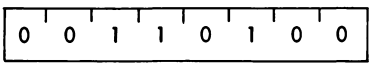

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,AC |

## INR m

(Increment memory)
Inkrementiere Speicherzelle

((H) (L)) ← ((H) (L)) + 1

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird um 1 erhöht. Das C-Flag wird nicht beeinflußt!



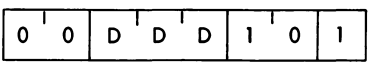

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,AC |

## DCR r

(Decrement register)
Dekrementiere Register

(r) ← (r) - 1

Der Inhalt des Registers r wird um 1 vermindert. C wird nicht beeinflußt!



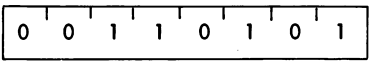

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,AC |

## DCR m

(Decrement memory)
Dekrementiere Speicherzelle

((H) (L)) ← ((H) (L)) - 1

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird um 1 vermindert. Das C-Flag wird nicht beeinflußt!



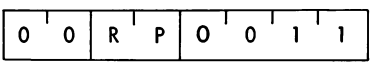

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,AC |

## INX rp

(Increment register pair)
Inkrementiere Registerpaar

(rh) (rl) ← (rh) (rl) + 1

Der Inhalt des Registerpaares rp wird um 1 erhöht. Es wird kein Flag beeinflußt!



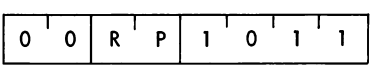

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 6 |
| Adressierung: | Register |
| Flags: | keine |

## DCX rp

(Decrement register pair)
Dekrementiere Registerpaar

(rh) (rl) ← (rh) (rl) - 1

Der Inhalt des Registerpaares rp wird um 1 vermindert. Es wird kein Flag beeinflußt!



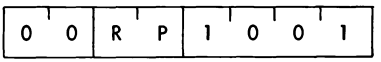

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 6 |
| Adressierung: | Register |
| Flags: | keine |

## DAD rp

(Add register pair to H and L)
Addiere Registerpaar zum HL-Register

(H) (L) ← (H) (L) + (rh) (rl)

Der Inhalt des Registerpaares rp wird zum Inhalt des Registerpaares HL addiert. Das Ergebnis steht im Registerpaar HL. Nur das C-Flag wird beeinflußt. Es wird gesetzt, wenn es bei der Addition einen Überlauf gibt, sonst gelöscht.




| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | Register |
| Flags: | C |

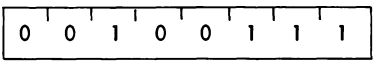
## DAA

(Decimal adjust accumulator)
Dezimaleinstellung des Akkus

Die 8-bit-Zahl im Akku wird so modifiziert, daß zwei BCD-kodierte Zahlen entstehen. Das geschieht wie folgt:

1) Falls der Wert der niederwertigen 4 bits im Akku größer als 9 ist oder wenn das AC-Flag gesetzt ist, wird 6 zum Akkuinhalt hinzuaddiert.

2) Falls jetzt der Wert der höchstwertigen 4 bits im Akku größer als 9 ist oder falls das C-Flag gesetzt ist, wird zu den 4 höchstwertigen bits 6 hinzuaddiert.



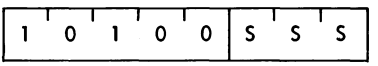

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Flags: | Z,S,P,C,AC |

## ANA r

(AND register)
Bilde UND mit Register

(A) ← (A) ∧ (r)

Der Inhalt des Registers r wird mit dem Akkuinhalt UND-verknüpft. Das Ergebnis steht im Akku. Das C-Flag wird gelöscht; das AC-Flag wird gesetzt.



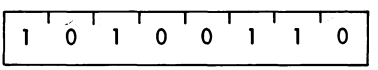

Zyklen: 1
Zustände: 4
Adressierung: Register
Flags: Z,S,P,C,AC

## ANA m

(AND memory)
Bilde UND mit Speicherzelle

(A) ← (A) ∧ ((H) (L))

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird mit dem Akku UND-verknüpft. Das Ergebnis steht im Akku. Das C-Flag wird gelöscht; das AC-Flag wird gesetzt.



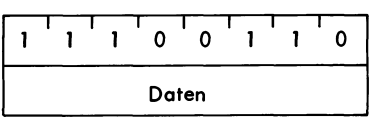

Zyklen: 2
Zustände: 7
Adressierung: Register indirekt
Flags: Z,S,P,C,AC

## ANI Daten

(AND immediate)
Bilde UND direkt

(A) ← (A) ∧ (Byte 2)

Der Inhalt des zweiten Byte des Befehls wird mit dem Akkuinhalt UND-verknüpft. Das Ergebnis steht im Akku. Das C-Flag wird gelöscht und das AC-Flag wird gesetzt.



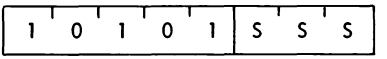

Zyklen: 2
Zustände: 7
Adressierung: sofort Flags: Z,S,P,C,AC

## XRA r

(Exclusive OR register)
Bilde EXKLUSIV ODER mit Register

(A) ← (A) ∀ (r)

Der Inhalt des Registers r wird mit dem Inhalt des Akkus EXKLUSIV ODER verknüpft. Das Ergebnis steht im Akku. Das C- und AC-Flag wird gelöscht.



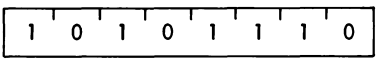

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,C,AC |

## XRA m

(Exclusive OR memory)
Bilde EXKLUSIV ODER mit Speicherzelle

(A) ← (A) ∀ ((H) (L))

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird mit dem Inhalt des Akkus EXKLUSIV ODER verknüpft. Das Ergebnis steht im Akku. Das C- und AC-Flag wird gelöscht.



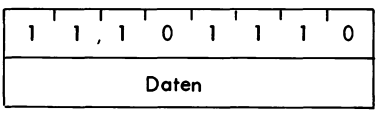

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,C,AC |

## XRI Daten

(Exclusive OR immediate)
Bilde EXKLUSIV ODER direkt

(A) ← (A) ∀ (byte 2)

Der Inhalt des zweiten Befehlsbytes wird mit dem Inhalt des Akkus EXKLUSIV ODER verknüpft. Das Ergebnis steht im Akku. Das C- und AC-Flag wird gelöscht.



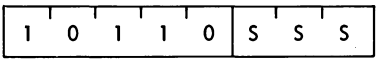

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | Z,S,P,C,AC |

## ORA r

(OR register)
Bilde ODER mit Register

(A) ← (A) ∨ (r)

Der Inhalt des Registers r wird mit dem Akkuinhalt ODER-verknüpft. Das Ergebnis steht im Akku. Das C- und AC-Flag wird gelöscht.



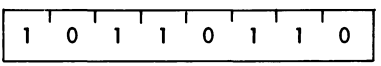

Zyklen: 1
Zustände: 4
Adressierung: Register
Flags: Z,S,P,C,AC

## ORA m

(OR memory)
Bilde ODER mit Speicherzelle

(A) ← (A) ∨ ((H) (L))

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird mit dem Akkuinhalt ODER-verknüpft. Das C- und AC-Flag wird gelöscht. Das Ergebnis steht im Akku.



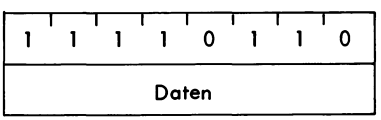

Zyklen: 2
Zustände: 7
Adressierung: Register indirekt
Flags: Z,S,P,C,AC

## ORI Daten

(OR immediate)
Bilde ODER direkt

(A) ← (A) ∨ (Byte 2)

Der Inhalt des zweiten Bytes des Befehls wird mit dem Inhalt des Akkus ODER-Verknüpft. Das Ergebnis steht im Akku. Das C- und AC-Flag wird gelöscht.



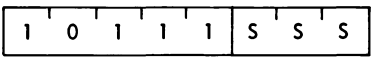

Zyklen: 2
Zustände: 7
Adressierung: sofort
Flags: Z,S,P,C,AC

## CMP r

(Compare register)
Vergleiche mit Register

(A) - (r)

Der Inhalt des Registers r wird vom Inhalt des Akkus abgezogen, jedoch bleibt der Akkuinhalt unverändert. Das Z-Flag wird gesetzt, wenn (A) = (r). Das C-Flag wird gesetzt, wenn (A) < (r).



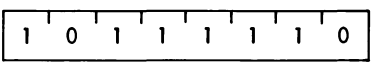

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 4 |
| Adressierung: | Register |
| Flags: | Z,S,P,C,AC |

## CMP m

(Compare Memory)
Vergleiche mit Speicherzelle

(A) - ((H) (L))

Der Inhalt derjenigen Speicherzelle, deren Adresse im HL-Register steht, wird vom Akkuinhalt abgezogen. Der Akkuinhalt bleibt unverändert. Das Z-Flag wird gesetzt, wenn (A) = ((H) (L)). Das C-Flag wird gesetzt, wenn (A) < ((H) (L)).



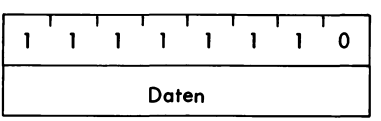

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,C,AC |

## CPI Daten

(Compare immediate)
Vergleiche direkt

(A) - (Byte 2)

Der Inhalt des zweiten Befehlsbytes wird vom Inhalt des Akkus abgezogen. Der Akkuinhalt bleibt unverändert. Das Z-Flag wird gesetzt, wenn (A) = (Byte 2). Das C-Flag wird gesetzt, wenn (A) < (Byte 2).



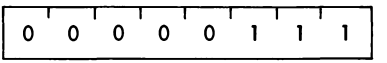

| | |
|---|---|
| Zyklen: | 2 |
| Zustände: | 7 |
| Adressierung: | sofort |
| Flags: | Z,S,P,C,AC |

## RLC

(Rotate left)
Schiebe links

$(A_{n+1}) \leftarrow (A_n)$; $(A_0) \leftarrow (A_7)$; $(C) \leftarrow (A_7)$

Der Akkuinhalt wird um eine Stelle linksverschoben. Das niedrigstwertige bit und das Carrybit erhalten den Wert, der aus dem höchstwertigen bit herausgeschoben wird. Nur das C-Flag wird beeinflußt.



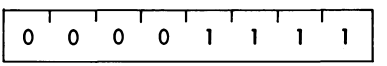

Zyklen: 1
Zustände: 4
Flags: C

## RRC

(Rotate right)
Schiebe rechts

$(A_n) \leftarrow (A_{n+1})$; $(A_7) \leftarrow (A_0)$; $(C) \leftarrow (A_0)$

Der Akkuinhalt wird um eine Stelle rechtsverschoben. Das höchstwertige bit und das Carrybit erhalten den Wert, der aus dem niedrigsten bit herausgeschoben wird. Nur das C-Flag wird beeinflußt.



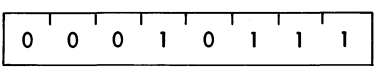

Zyklen: 1
Zustände: 4
Flags: C

## RAL

(Rotate left through carry)
Schiebe links durch das C-bit

$(A_{n+1}) \leftarrow (A_n)$; $(C) \leftarrow (A_7)$; $(A_0) \leftarrow (C)$

Der Akkuinhalt wird um eine Stelle linksverschoben. Das niedrigstwertige bit erhält den Wert des C-Flag, und das C-Flag erhält den Wert, der aus dem höchstwertigen bit des Akkus herausgeschoben wird. Nur das C-Flag wird beeinflußt.



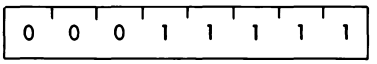

Zyklen: 1
Zustände: 4
Flags: C

## RAR

(Rotate right through carry)
Schiebe rechts durch das C-bit

$(A_n) \leftarrow (A_{n+1})$; $(C) \leftarrow (A_0)$; $(A_7) \leftarrow (C)$

Der Akkuinhalt wird um eine Stelle rechtsverschoben. Das höchstwertige bit erhält den Wert des C-Flag, und das C-Flag erhält den Wert, der aus dem niedrigstwertigen bit des Akkus herausgeschoben wird.



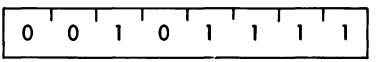

Zyklen: 1
Zustände: 4
Flags: C

## CMA

(Complement accumulator)
Invertiere Akkumulator

$(A) \leftarrow (\overline{A})$

Der Akkuinhalt wird invertiert. Kein Flag wird beeinflußt.



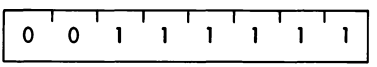

Zyklen: 1 Zustände: 4

## CMC

(Complement carry)
Invertiere das Carry-Flag

$(C) \leftarrow (\overline{C})$

Das Carry-Flag wird invertiert. Kein anderes Flag wird beeinflußt.



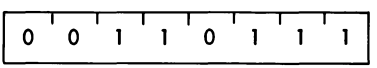

Zyklen: 1 Zustände: 4

## STC

(Set carry)
Setze das Carry-Flag

$(C) \leftarrow 1$

Das Carry-Flag erhält den Wert 1. Kein anderes Flag wird beeinflußt.




Zyklen: 1 Zustände: 4

Verzweigungsbefehle (Sprungbefehle)

Diese Gruppe von Befehlen verändert den normalen sequentiellen Programmablauf.

Die Bedingungsbits werden von keinem dieser Befehle beeinflußt.

Es gibt unbedingte und bedingte Sprungbefehle. Unbedingte Sprungbefehle führen einfach die spezifizierte Operation mit dem Befehlszähler (PC, program counter) aus. Bedingte Sprungbefehle untersuchen zuerst den Zustand der vier Flags (das AC-Flag spielt hier keine Rolle) und entscheiden dann, ob der Sprung ausgeführt wird oder nicht. Die Bedingungen, die vorgegeben werden können, sind folgende:

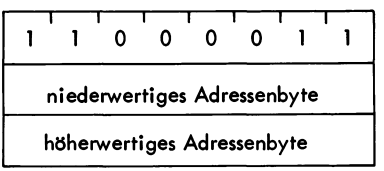
| Bedingung | Bedingungscode (CCC) |
|---|---|
| NZ - not zero (Z = 0) | 0 0 0 |
| Z - zero (Z = 1) | 0 0 1 |
| NC - no carry (C = 0) | 0 1 0 |
| C - carry (C = 1) | 0 1 1 |
| PO - parity odd (P = 0) | 1 0 0 |
| PE - parity even (P = 1) | 1 0 1 |
| P - plus (S = 0) | 1 1 0 |
| M - minus (S = 1) | 1 1 1 |

## JMP Adr

(Jump)
Springe unbedingt

(PC) ← (Byte 3) (Byte 2)

Das Programm wird mit dem Befehl fortgesetzt, dessen Adresse in Byte 2 und 3 des Sprungbefehls angegeben ist.



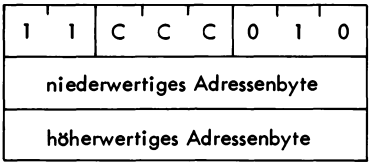

Zyklen: 3
Zustände: 10
Adressierung: sofort
Flags: keine

## J condition Adr

(Conditional jump)
bedingter Sprung

IF (CCC), (PC) ← (Byte 3) (Byte 2)

Falls die durch den Bedingungscode CCC gekennzeichnete Bedingung erfüllt ist, wird das Programm mit dem Befehl fortgesetzt, dessen Adresse in Byte 2 und 3 des hier beschriebenen Befehls steht. Andernfalls wird der sequentielle Programmablauf nicht unterbrochen.



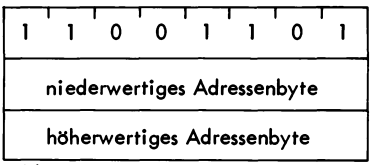

Zyklen: 2/3
Zustände: 7/10
Adressierung: sofort
Flags: keine

## CALL Adr

(Call) Rufe

((SP) - 1) ← (PCH)
((SP) - 2) ← (PCL)
(SP) ← (SP) - 2
(PC) ← (Byte 3) (Byte 2)

Die höherwertigen 8 bits der Adresse des nächsten Befehls werden in diejenige Speicherzelle gebracht, deren Adresse aus dem Inhalt des Stapelzeigers minus 1 gebildet wird. Die niederwertigen 8 bits der Adresse des nächsten Befehls werden in diejenige Speicherzelle gebracht, deren Adresse aus dem Inhalt des Stapelzeigers minus 2 gebildet wird. Sodann wird der Inhalt des Stapelzeigers um 2 vermindert. Das Programm wird mit dem Befehl fortgesetzt, dessen Adresse in Byte 2 und 3 des hier beschriebenen Befehls enthalten ist.



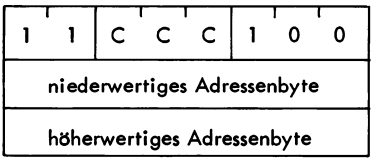

Zyklen: 5
Zustände: 18
Adressierung: sofort/Register indirekt
Flags: keine

## C condition Adr

(Conditional call)
Rufe unter Bedingung

IF (CCC), ((SP) - 1) ← (PCH)
((SP) - 2) ← (PCL)
(SP) ← (SP) - 2
(PC) ← (Byte 3) (Byte 2)

Falls die durch den Bedingungscode CCC angegebene Bedingung erfüllt ist, werden die Aktivitäten ausgeführt, die in dem oben beschriebenen CALL-Befehl spezifiziert wurden. Andernfalls wird der sequentielle Programmablauf nicht unterbrochen.



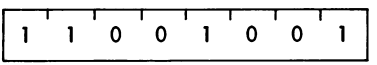

Zyklen: 2/5
Zustände: 9/18
Adressierung: sofort/Register indirekt
Flags: keine

## RET

(Return)
Rücksprung

(PCL) ← ((SP)); (PCH) ← ((SP) + 1); (SP) ← (SP) + 2

Der Inhalt derjenigen Speicherzelle, deren Adresse der Stapelzeiger angibt, wird in die niederwertigen 8 bits des Befehlszählers (PC) geladen. Der Inhalt der Speicherzelle mit der nächsthöheren Adresse wird in die höherwertigen 8 bits des Befehlszählers geladen. Der Inhalt des Stapelzeigers wird um 2 erhöht.



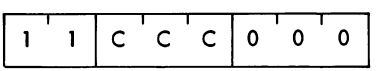

Zyklen: 3
Zustände: 10
Adressierung: Register indirekt
Flags: keine

## Rcondition

(Conditional return)
Bedingter Rücksprung

IF (CCC), (PCL) ← ((SP));
(PCH) ← ((SP) + 1);
(SP) ← (SP) + 2

Wenn die durch den Bedingungscode CCC spezifizierte Bedingung erfüllt ist, werden die Aktivitäten ausgeführt, die bei dem oben beschriebenen RET-Befehl angegeben wurden. Andernfalls wird der sequentielle Programmablauf nicht unterbrochen.




Zyklen: 1/3
Zustände: 6/12
Adressierung: Register indirekt
Flags: keine

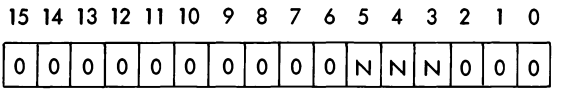
## RST n

(Restart)
Wiederstart bei angegebener Adresse

((SP) - 1) ← (PCH)
((SP) - 2) ← (PCL)
(SP) ← (SP) - 2
(PC) ← 8 (NNN)

Dies ist ein Sprungbefehl, der nur 1 Byte im Speicher belegt. Die Zieladresse ist im Befehlscode enthalten. Der Ablauf ist wie folgt: Die höherwertigen 8 bits der nächsten Befehlsadresse werden in der Speicherzelle abgespeichert, deren Adresse aus dem Inhalt des Stapelzeigers minus 1 gebildet wird. Die niederwertigen 8 bits der nächsten Befehlsadresse werden in der Speicherzelle abgelegt, deren Adresse sich aus dem Inhalt des Stapelzeigers minus 2 ergibt. Der Inhalt des Stapelzeigers wird um 2 vermindert. Das Programm wird bei dem Befehl fortgesetzt, dessen Adresse acht Mal dem Wert von NNN entspricht.


Zyklen: 3
Zustände: 12
Adressierung: Register indirekt
Flags: keine



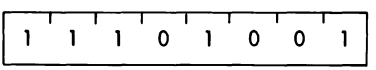

Inhalt des Befehlszählers nach RST

## PCHL

(Jump H and L indirect - move H and L to PC)
Springe mit dem Inhalt von HL als Sprungziel

(PCH) ← (H)
(PCL) ← (L)

Dieser wichtige Befehl ermöglicht es, relative Sprünge auszuführen, d.h. Sprünge, die unabhängig vom aktuellen Stand des Befehlszählers sind. Der Inhalt des H-Registers wird in die höherwertigen 8 bits des Befehlszählers geladen; der Inhalt des L-Registers wird in die niederwertigen 8 bits des Befehlszählers (PC) geladen.




Zyklen: 1
Zustände: 6
Adressierung: Register
Flags: keine

## PUSH rp

(Push)

((SP) - 1) ← (rh); ((SP) - 2) ← (rl); (SP) ← (SP) - 2

Der Inhalt des höherwertigen Registers des Registerpaares rp wird in die Speicherzelle gebracht, deren Adresse sich aus dem Inhalt des Stapelzeigers minus 1 ergibt. Der Inhalt des niederwertigen Registers wird in die Speicherzelle gebracht, deren Adresse sich aus dem Inhalt des Stapelzeigers minus 2 ergibt. Der Inhalt des Stapelzeigers wird um 2 vermindert. ACHTUNG: rp = SP darf nicht verwendet werden!

| 1 | 1 | R | P | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

Zyklen: 3
Zustände: 12
Adressierung: Register indirekt
Flags: keine

## PUSH PSW

(Push processor status word)
Speichere die Flag-bits

((SP) - 1) ← (A)
$((SP) - 2)_0$ ← (C), $((SP) - 2)_1$ ← X
$((SP) - 2)_2$ ← (P), $((SP) - 2)_3$ ← X
$((SP) - 2)_4$ ← (AC), $((SP) - 2)_5$ ← X
$((SP) - 2)_6$ ← (Z), $((SP) - 2)_7$ (S)
(SP) ← (SP) - 2 X = unbestimmt

Der Inhalt des Akkus wird in die Speicherzelle gebracht, deren Adresse sich aus dem Inhalt des Stapelzeigers minus 1 ergibt. Die fünf Flags werden zu einem Wort, dem „Prozessor-Zustandswort", zusammengefaßt. Dieses Wort wird in die Speicherzelle gebracht, deren Adresse sich aus dem Inhalt des Stapelzeigers minus 2 ergibt. Der Inhalt des Stapelzeigers wird um 2 vermindert.

| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

Zyklen: 3
Zustände: 12
Adressierung: Register indirekt
Flags: keine

Flag-Wort

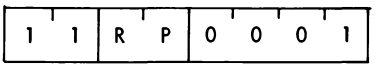
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| S | Z | X | AC | X | P | X | C |

## POP

(Pop)

(rl) ← ((SP)); (rh) ← ((SP) + 1); (SP) ← (SP) + 2

Der Inhalt derjenigen Speicherzelle, deren Adresse im Stapelzeiger enthalten ist, wird in das niederwertige Register des Registerpaares rp gebracht. Der Inhalt der Speicherzelle, deren Adresse gleich dem Inhalt des Stapelzeigers plus 1 ist, wird in das höherwertige Register des Registerpaares rp gebracht. Der Inhalt des Stapelzeigers wird um 2 erhöht. ACHTUNG: rp = SP darf nicht verwendet werden!



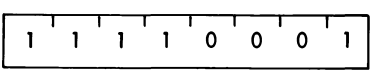

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | Register indirekt |
| Flags: | keine |

## POP PSW

(Pop processor status word)
Hole das Prozessor-Zustandswort vom Speicher

(C) ← ((SP))$_0$
(P) ← ((SP))$_2$
(AC) ← ((SP))$_4$
(Z) ← ((SP))$_6$
(S) ← ((SP))$_7$
(A) ← ((SP) + 1)
(SP) ← (SP) + 2

Der Inhalt der Speicherzelle, deren Adresse im Stapelzeiger enthalten ist, wird als Zustandswort interpretiert, und die Flags werden entsprechend gesetzt. Der Inhalt der Speicherzelle, deren Adresse um 1 höher als der Inhalt des Stapelzeigers ist, wird in den Akku gebracht. Der Inhalt des Stapelzeigers wird um 2 erhöht.



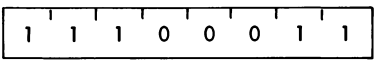

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | Register indirekt |
| Flags: | Z,S,P,C,AC |

## XTHL

(Exchange stack top with H and L)
Tausche die oberen Worte des Stapelspeichers mit HL

(L) ↔ ((SP))
(H) ↔ ((SP) + 1)

Der Inhalt des L-Registers wird mit dem Inhalt derjenigen Speicherzelle ausgetauscht, deren Adresse im Stapelzeiger steht. Der Inhalt des H-registers wird mit dem Inhalt der Speicherzelle ausgetauscht, deren Adresse sich aus dem Inhalt des Stapelzeigers plus 1 ergibt.



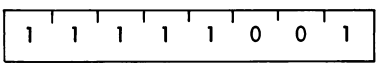

| | |
|---|---|
| Zyklen: | 5 |
| Zustände: | 16 |
| Adressierung: | Register indirekt |
| Flags: | keine |

## SPHL

(Move HL to SP)
Lade SP mit dem Inhalt des HL-Registers

(SP) ← (H) (L)

Der Inhalt der Register H und L wird in das Stapelzeiger-Register gebracht.



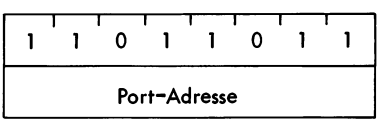

| | |
|---|---|
| Zyklen: | 1 |
| Zustände: | 6 |
| Adressierung: | Register |
| Flags: | keine |

## IN port

(Input)
Eingabe

(A) ← (Daten)

Die Daten, die vom angegebenen Port auf den Datenbus gebracht werden, werden in den Akku geladen.



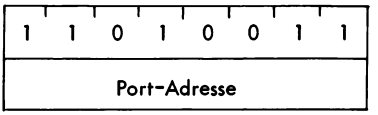

| | |
|---|---|
| Zyklen: | 3 |
| Zustände: | 10 |
| Adressierung: | direkt |
| Flags: | keine |

## OUT port

(Output)
Ausgabe

(Daten) ← (A)

Der Inhalt des Akkus wird auf den Datenbus gegeben und in den angegebenen Port gebracht.

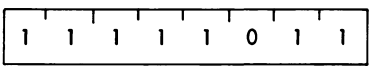



Zyklen: 3
Zustände: 10
Adressierung: direkt
Flags: keine

## EI

(Enable interrupts)
Interrupts ermöglichen

Nach Ausführung des nächsten Befehls werden Interrupts ermöglicht. Während der Ausführung von EI werden keine Interrupts erkannt.

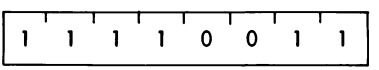



Zyklen: 1 Zustände: 4

## DI

(Disable interrupts)
Interrupts verhindern

Sofort nach Ausführung des DI-Befehls wird das Interruptsystem gesperrt. Während der Ausführung von DI werden keine Interrupts erkannt.

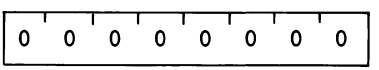



Zyklen: 1 Zustände: 4

## NOP

(No operation)
Keine Operation

Es wird keinerlei Operation ausgeführt.




Zyklen: 1 Zustände: 4

## RIM

(Read interrupt mask)
Interrupt-Maske lesen

Nach Ausführung des RIM-Befehls enthält der Akku die RESTART-Interruptmasken, das Interrupt-Ermöglichungs-Flag, etwa anstehende Interrupts und den Inhalt des seriellen Dateneingangs (SID).

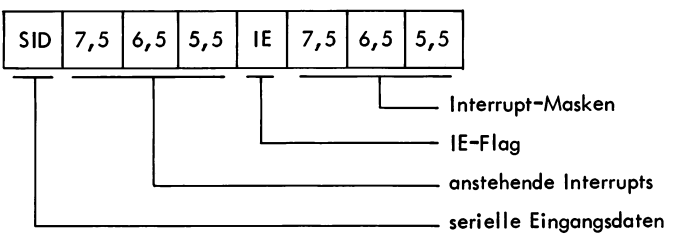
Wenn der RIM-Befehl zum ersten Mal nach einem erfolgten TRAP-Interrupt ausgeführt wird, gibt das IE-bit im Akku den Zustand v o r dem TRAP-Interrupt an. Ferner wird das IE-Zustandsbit freigegeben, so daß alle folgenden RIM-Befehle den jeweils vorliegenden IE-Zustand wiedergeben.

WICHTIG: Nach einer TRAP-Unterbrechung muß RIM ausgeführt werden, um den korrekten IE-Zustand herbeizuführen.

| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Zyklen: 1
Zustände: 4
Flags: keine

Akkuinhalt nach Ausführung von RIM:




## SIM

(Set interrupt masks)
Interruptmasken setzen

Während der Ausführung des SIM-Befehls wird der Akkuinhalt dazu benutzt, um die RESTART-Interruptmasken zu programmieren. Bits 0 bis 2 besetzen die Maskenbits für RST 5.5, 6.5 und 7.5 im Interrupt-Maskenregister, falls bit 3 besetzt ist. Bit 3 ist also ein Steuerbit für die Möglichkeit, Maskenbits zu setzen.

Wenn ein Maskénbit gesetzt ist, wird der zugehörige Interrupt verhindert.

| | gesetzt | gelöscht |
|---|---|---|
| Maske für RST 5.5 | wenn bit 0 = 1 | wenn bit 0 = 0 |
| Maske für RST 6.5 | bit 1 = 1 | bit 1 = 0 |
| Maske für RST 7.5 | bit 2 = 1 | bit 2 = 0 |

Das interne flankengetriggerte Anforderungsflipflop für RST 7.5 wird gelöscht, wenn bit 4 des Akkus den Wert 1 hat, unabhängig davon, ob RST 7.5 maskiert ist oder nicht.

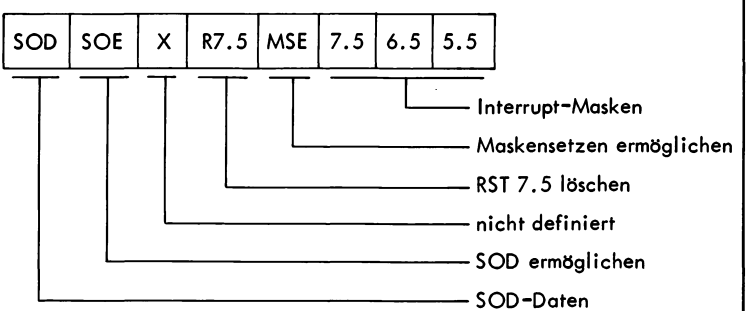
Bei einem hardware-RESET werden alle RST-Masken gesetzt und alle Interrupts verhindert.

Mit SIM kann man auch den Ausgangspuffer für den seriellen Datenausgang SOD laden. Bit 7 des Akkus wird in den SOD-Puffer gebracht, falls bit 6 gesetzt ist. Der Puffer wird nicht beeinflußt, wenn bit 6 den Wert 0 hat. Ein hardware-RESET setzt den Puffer auf 0.

| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Zyklen: 1
Zustände: 4
Flags: keine

Akkuinhalt vor Ausführung von SIM:




## HLT

Halt

Der Prozessor wird angehalten. Keine Register oder Flags werden beeinflußt.

| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Zyklen: 1
Zustände: 5

## 8085-Befehlsliste in Hex-Folge

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | NOP | | 33 | INX | SP | 66 | MOV | H,m | 99 | SBB | C | CC | CZ | |
| 01 | LXI | BC | 34 | INR | m | 67 | MOV | H,A | 9A | SBB | D | CD | CALL | |
| 02 | STAX | BC | 35 | DCR | m | 68 | MOV | L,B | 9B | SBB | E | CE | ACI | |
| 03 | INX | BC | 36 | MVI | m | 69 | MOV | L,C | 9C | SBB | H | CF | RST | 1 |
| 04 | INR | B | 37 | STC | | 6A | MOV | L,D | 9D | SBB | L | D0 | RNC | |
| 05 | DCR | B | 38 | --- | | 6B | MOV | L,E | 9E | SBB | m | D1 | POP | D |
| 06 | MVI | B | 39 | DAD | SP | 6C | MOV | L,H | 9F | SBB | A | D2 | JNC | |
| 07 | RLC | | 3A | LDA | | 6D | MOV | L,L | A0 | ANA | B | D3 | OUT | |
| 08 | --- | | 3B | DCX | SP | 6E | MOV | L,m | A1 | ANA | C | D4 | CNC | |
| 09 | DAD | B | 3C | INR | A | 6F | MOV | L,A | A2 | ANA | D | D5 | PUSH | D |
| 0A | LDAX | BC | 3D | DCR | A | 70 | MOV | m,B | A3 | ANA | E | D6 | SUI | |
| 0B | DCX | BC | 3E | MVI | A | 71 | MOV | m,C | A4 | ANA | H | D7 | RST | 2 |
| 0C | INR | C | 3F | CMC | | 72 | MOV | m,D | A5 | ANA | L | D8 | RC | |
| 0D | DCR | C | 40 | MOV | B,B | 73 | MOV | m,E | A6 | ANA | m | D9 | --- | |
| 0E | MVI | C | 41 | MOV | B,C | 74 | MOV | m,H | A7 | ANA | A | DA | JC | |
| 0F | RRC | | 42 | MOV | B,D | 75 | MOV | m,L | A8 | XRA | B | DB | IN | |
| 10 | --- | | 43 | MOV | B,E | 76 | HLT | | A9 | XRA | C | DC | CC | |
| 11 | LXI | DE | 44 | MOV | B,H | 77 | MOV | m,A | AA | XRA | D | DD | --- | |
| 12 | STAX | DE | 45 | MOV | B,L | 78 | MOV | A,B | AB | XRA | E | DE | SBI | |
| 13 | INX | DE | 46 | MOV | B,m | 79 | MOV | A,C | AC | XRA | H | DF | RST | 3 |
| 14 | INR | D | 47 | MOV | B,A | 7A | MOV | A,D | AD | XRA | L | E0 | RPO | |
| 15 | DCR | D | 48 | MOV | C,B | 7B | MOV | A,E | AE | XRA | m | E1 | POP | H |
| 16 | MVI | D | 49 | MOV | C,C | 7C | MOV | A,H | AF | XRA | A | E2 | JPO | |
| 17 | RAL | | 4A | MOV | C,D | 7D | MOV | A,L | B0 | ORA | B | E3 | XTHL | |
| 18 | --- | | 4B | MOV | C,E | 7E | MOV | A,m | B1 | ORA | C | E4 | CPO | |
| 19 | DAD | D | 4C | MOV | C,H | 7F | MOV | A,A | B2 | ORA | D | E5 | PUSH | H |
| 1A | LDAX | DE | 4D | MOV | C,L | 80 | ADD | B | B3 | ORA | E | E6 | ANI | |
| 1B | DCX | DE | 4E | MOV | C,m | 81 | ADD | C | B4 | ORA | H | E7 | RST | 4 |
| 1C | INR | E | 4F | MOV | C,A | 82 | ADD | D | B5 | ORA | L | E8 | RPE | |
| 1D | DCR | E | 50 | MOV | D,B | 83 | ADD | E | B6 | ORA | m | E9 | PCHL | |
| 1E | MVI | E | 51 | MOV | D,C | 84 | ADD | H | B7 | ORA | A | EA | JPE | |
| 1F | RAR | | 52 | MOV | D,D | 85 | ADD | L | B8 | CMP | B | EB | XCHG | |
| 20 | RIM | | 53 | MOV | D,E | 86 | ADD | m | B9 | CMP | C | EC | CPE | |
| 21 | LXI | HL | 54 | MOV | D,H | 87 | ADD | A | BA | CMP | D | ED | --- | |
| 22 | SHLD | | 55 | MOV | D,L | 88 | ADC | B | BB | CMP | E | EE | XRI | |
| 23 | INX | HL | 56 | MOV | D,m | 89 | ADC | C | BC | CMP | H | EF | RST | 5 |
| 24 | INR | H | 57 | MOV | D,A | 8A | ADC | D | BD | CMP | L | F0 | RP | |
| 25 | DCR | H | 58 | MOV | E,B | 8B | ADC | E | BE | CMP | m | F1 | POP | PSW |
| 26 | MVI | H | 59 | MOV | E,C | 8C | ADC | H | BF | CMP | A | F2 | JP | |
| 27 | DAA | | 5A | MOV | E,D | 8D | ADC | L | C0 | RNZ | | F3 | DI | |
| 28 | --- | | 5B | MOV | E,E | 8E | ADC | m | C1 | POP | B | F4 | CP | |
| 29 | DAD | H | 5C | MOV | E,H | 8F | ADC | A | C2 | JNZ | | F5 | PUSH | PSW |
| 2A | LHLD | | 5D | MOV | E,L | 90 | SUB | B | C3 | JMP | | F6 | ORI | |
| 2B | DCX | HL | 5E | MOV | E,m | 91 | SUB | C | C4 | CNZ | | F7 | RST | 6 |
| 2C | INR | L | 5F | MOV | E,A | 92 | SUB | D | C5 | PUSH | B | F8 | RM | |
| 2D | DCR | L | 60 | MOV | H,B | 93 | SUB | E | C6 | ADI | | F9 | SPHL | |
| 2E | MVI | L | 61 | MOV | H,C | 94 | SUB | H | C7 | RST | 0 | FA | JM | |
| 2F | CMA | | 62 | MOV | H,D | 95 | SUB | L | C8 | RZ | | FB | EI | |
| 30 | SIM | | 63 | MOV | H,E | 96 | SUB | m | C9 | RET | | FC | CM | FD --- |
| 31 | LXI | SP | 64 | MOV | H,H | 97 | SUB | A | CA | JZ | | FE | CPI | |
| 32 | STA | | 65 | MOV | H,L | 98 | SBB | B | CB | --- | | FF | RST | 7 |

DIPL.-ING. H. WEIDNER, D-3300 BRAUNSCHWEIG, KRÖGERSTRASSE 69

In diesem Abschnitt wird an Beispielprogrammen gezeigt, was man mit dem MICRO-MAX II machen kann und wie er programmiert wird. Am Anfang stehen einige bewußt einfache, ja geradezu primitive Programme. Sie sind für Anwender gedacht, für die das Programmieren von Mikrocomputern völliges Neuland ist. Auch für die ersten Stunden eines Programmierlehrgangs oder einer Arbeitsgemeinschaft können diese Programme als Beispiele dienen.

Der Hauptteil dieses Abschnittes widmet sich der Programmierung von Tastatur und Anzeige, denn diese stellen die Ein- und Ausgabemedien des Rechners dar. Man sollte sich jedoch vor Augen halten, daß immer die Programmentwicklung die Hauptsache ist, die Verwendung bestimmter Befehle und das Verständnis ihrer Wirkungsweise. Daß sich diese Programme - notgedrungen - z.B. auf die Anzeige beziehen, ist zweitrangig.

Zum Schluß werden einige Routinen angegeben, die häufig benötigt werden.

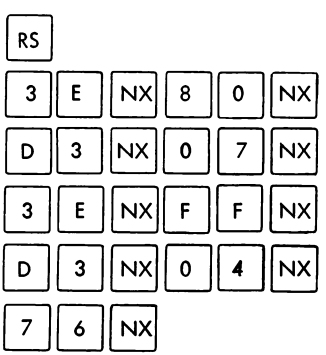
## Die Programmierung der LED-Zeile

Einer der drei Ports, nämlich Port A, ist an Leuchtdioden angeschlossen. Damit kann der Pegel an den 8 bits dieses Ports optisch angezeigt werden. Zum Verständnis der folgenden Programme ist es erforderlich, die „Programmierung des E/A-Bausteins 8255" zu beherrschen, zumindest den Inhalt der Seiten S1 bis S3.

**1.** Programm: Alle Leuchtdioden zum Aufleuchten bringen

```
0800   3E 80      MVI A,80     ;der Akku wird mit der Konstanten 80 geladen
0802   D3 07      OUT 07       ;alle Ports sollen Ausgänge sein
0804   3E FF      MVI A,FF     ;im Akku wird die Konstante FF erzeugt
0806   D3 04      OUT 04       ;alle bits im Port A werden gesetzt
0808   76         HLT          ;Programmende
```

Um dieses Programm ab Adresse 0800 in den Speicher einzuschreiben, drücken Sie folgende Tasten:



Die Praxis zeigt immer wieder, daß es angebracht und wichtig ist, den Speicherinhalt nach erfolgter Programmeingabe auf seine Richtigkeit hin zu überprüfen. Deshalb drücken Sie zunächst wieder [RS] und dann wiederholt die Taste [NX] so lange, bis Sie sich den Inhalt aller vom Programm belegten Speicherzellen angesehen haben. Vergleichen Sie nach jedem Drücken von NX den Speicherinhalt mit dem Programmprotokoll! Ist das Programm korrekt eingegeben, dann drücken Sie

[RS] [RN]

Nun muß die Anzeige über der Tastatur dunkel sein, und alle Leuchtdioden müssen leuchten.

**2.** Programm: Die linke äußere und die rechte äußere LED soll leuchten

Dies ist eine Variation des 1. Programms. Wir müssen dafür sorgen, daß im Port A nur die bits 0 und 7 gesetzt werden. Man muß also im Akku statt FF die Konstante 81 erzeugen. Wir brauchen nun keineswegs das gesamte Programm neu einzugeben, sondern es muß lediglich der Inhalt der Speicherzelle 0805 von FF in 81 umgeändert werden. Das erreicht man durch Betätigung der Tasten

[RS] [AD] [0] [8] [0] [5] [DT] [8] [1] [NX]

[RS] [RN]

Nun dürfen nur die beiden äußeren LED's leuchten. Erzeugen Sie andere Muster in der Leuchtdiodenzeile, indem Sie den Inhalt der Zelle 0805 ändern!

Machen Sie sich an diesen zwei Programmbeispielen auch klar, was bei der Betätigung der Tasten passiert. Im ersten Beispiel haben wir außer den Datentasten die Funktionstasten RS, NX und RN benutzt. Das zweite Beispiel zeigt, wie die Tasten AD und DT funktionieren. Speichern Sie zur Übung das ganze Programm einmal ab Zelle 0900! Um dieses Programm zu starten, müssen Sie drücken:

[RS] [AD] [0] [9] [0] [0] [RN]

Sie können ab Zelle 0800 ein Programm speichern, welches eine bestimmte LED-Konfiguration zum Leuchten bringt, und ab Zelle 0900 kann ein Programm stehen, welches eine andere Konfiguration zum Leuchten bringt. Je nach dem,welches Programm Sie „anstarten", wird in der LED-Zeile das entsprechende Leuchtbild erscheinen. Dieses Beispiel zeigt, daß im Speicher mehrere unabhängige Programme stehen können.

**3.** Programm: Nach Programmende soll wieder der Monitor gerufen werden.

Dazu braucht in Zelle 0808 statt HLT nur RST 0 (Hex-Code C7) eingespeichert zu werden. Ändern Sie den Zelleninhalt in der beschriebenen Weise ab! Wenn Sie nun RS und RN drücken, erscheint in der LED-Zeile wieder das gewünschte Muster; außerdem aber leuchtet in der Siebensegmentanzeige 0800 3E . Der Befehl RST 0 wirkt also wie eine Betätigung der Taste RS, o h n e das System in den Grundzustand zu versetzen. Das Beispiel demonstriert auch die speichernde Eigenschaft

der Ports: Obwohl das Monitorprogramm läuft, bleiben die LED's erleuchtet.

Der Umgang mit der Tastatur sollte Ihnen nun geläufig sein. Nachfolgend werden daher die zu betätigenden Tasten nicht mehr abgebildet, sondern es wird nur noch die Befehlsliste eines Programms angegeben.

**4.** Programm: Akku fortlaufend inkrementieren und anzeigen.

Mit dem Befehl OUT 04 wird ja der Inhalt des Akkus in den Port A gebracht, d.h. die Leuchtdioden zeigen an, welches bit im Akku besetzt ist und welches nicht. So können wir sehr schön die Wirkung von Befehlen, die sich auf den Akku beziehen, untersuchen. Fangen wir mit dem Befehl INR A an.

Dieser Befehl erhöht bekanntlich den Akkuinhalt um 1. Nun wissen wir ja, daß die Ausführungszeit eines Befehls nur wenige Mikrosekunden beträgt; das ist so schnell, daß sich in der LED-Anzeige für das menschliche Auge überhaupt nichts ändern würde. Wir müssen daher nach jedem Inkrementieren eine Verzögerung einbauen, die es möglich macht,den neuen Akkuinhalt vom alten zu unterscheiden. Solche Zeitverzögerungsprogramme sind im Monitor enthalten und können vom Programmierer benutzt werden. So bewirkt z.B. der Aufruf

```
CD BD 03      CALL 03BD
```

eine Verzögerung von 0,1 Sekunde. Nachdem diese Zeit vergangen ist, soll erneut inkrementiert werden. Wir werden den Befehl INR A nun nicht fortlaufend wiederholen, sondern an die Stelle im Programm, wo inkrementiert wird, zurückspringen. Wir lernen in diesem Beispiel also einen Unterprogrammaufruf und einen Sprungbefehl kennen.

```
0800    3E 80       MVI A,80
0802    D3 07       OUT 07      ;diese Vorbereitung kennen wir schon
0804    3E 00       MVI A,00    ;fangen wir bei 0 an!
0806    3C          INR A       ;Akkuinhalt um 1 erhöhen
0807    D3 04       OUT 04      ;und anzeigen
0809    CD BD 03    CALL 03BD   ;Anzeige 0,1 s lang stehen lassen
080C    C3 06 08    JMP 0806    ;und weitermachen
```

Geben Sie das Programm ein (auch das letzte NX nicht vergessen!) und starten Sie! Sie sehen, wie sich der Akku durch das fortlaufende Inkrementieren allmählich füllt, bis das Spiel schließlich von vorn beginnt. Dieses Programm hört von allein nicht auf; es läuft, wie man sagt, in einer Endlosschleife. Erst ein Betätigen der Taste RS unterbricht das Programm und aktiviert wieder den Monitor.

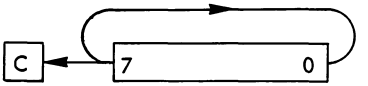
Dieses Programm kann in zweierlei Weise abgewandelt werden, und solange es im Speicher steht, wollen wir das auch tun. Einmal kann der Befehl INR A durch den Befehl DCR A ersetzt werden; der Akku wird dann „leergezählt". Führen Sie die Übung aus! Eine andere Variante ist, statt einer Verzögerung von 0,1 s eine solche von 1 s vorzusehen. Dazu brauchen Sie den Rufbefehl nur in CALL 03B1 abzuändern. Achten Sie auf die richtige Reihenfolge beim Eingeben der zwei Adressenbytes!

Schließlich ist noch etwas zu dem Programm zu bemerken: Mit dem Befehl MVI A,00 wird der Akku gelöscht. Der Befehl XRA A bewirkt dasselbe, belegt aber im Speicher nur 1 byte. Versuchen Sie sich zu erklären, wieso das kommt!

## 5. Programm: Akkuinhalt verschieben

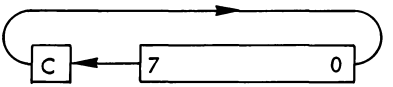
Der 8085 kennt die Rotationsbefehle RLC und RAL (Schieberichtung nach links) und RRC und RAR (Schieberichtung nach rechts). Ergänzend zu den Beschreibungen dieser Befehle in der Befehlssatzbeschreibung kann man sich ihre Wirkungsweise noch an Hand einer kleinen Skizze klarmachen:

RLC:



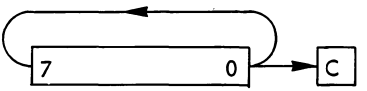

Bei jeder Anwendung des Befehls RLC geht bit 7 des Akkus sowohl ins Carrybit als auch nach bit 0.

RAL:



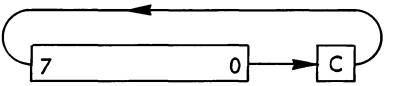

Es wird „durch das Carrybit hindurch" geschoben. Man nennt das auch „arithmetische Rotation".

RRC:




Bei jeder Anwendung des Befehls RRC geht bit 0 des Akkus sowohl ins Carrybit als auch nach bit 7.

RAR:




Dieser Befehl wirkt wie der Befehl RAL, nur eben „rechts herum".

Mit Hilfe eines kleinen Testprogramms kann man sich diese Zusammenhänge sehr schön vor Augen führen. Als Anzeigeverzögerung wählen wir jetzt 1 Sekunde, da die Vorgänge sonst zu schnell ablaufen würden.

```
0800   3E 80       MVI A,80
0802   D3 07       OUT 07      ;Ports vorbereiten
0804   3E 01       MVI A,01    ;wir wollen ein einzelnes bit verschieben
0806   D3 04       OUT 04
0808   CD B1 03    CALL 03B1   ;eine Sekunde Verzögerung
080B   07          RLC         ;um eine Position linksverschieben
080C   C3 06 08    JMP 0806    ;und weitermachen
```

Wenn Sie dieses Programm richtig eingegeben haben und starten, sehen Sie genau, wie das bit bei jedem Durchlauf einer Programmschleife um eine Position weiter nach links geschoben wird. Wenn das bit aus der Position 7 im Akku herausgeschoben wird, erscheint es sofort wieder in der Position 0. Ein bit im Akku ist also immer besetzt.

Ändern Sie nun den Befehl in Zelle 080B in RAL (Hex-Code 17). Wenn unser Testbit jetzt aus dem Akku herausgeschoben wird, bleiben die Leuchtdioden für eine Weile dunkel. Der Akku ist nämlich leer, dafür ist das Carrybit besetzt. Erst beim nächsten Programmdurchlauf erscheint das bit wieder in der unteren Position.
Füllen Sie nun einmal Zelle 0805 mit 7F statt 01. Wir haben jetzt kein wanderndes Bit, sondern eine wandernde Lücke. Wenn sie von rechts in den Akku hineingeschoben wird, sind plötzlich zwei Leuchtdioden dunkel. Wie kommt das?

Wenn die „Lücke" den Akku verläßt, wird von rechts das Carrybit nachgeschoben. Dies ist nicht besetzt, und die rechte Leuchtdiode erlischt. Beim nächsten Programmdurchlauf wird dann die Lücke, die sich vorher im Carrybit aufgehalten hatte, nachgeschoben, und so kommt es, daß zwei bits nebeneinander nicht besetzt sind.

Das Carrybit spielt hier also eine große Rolle. Es gibt deshalb Befehle, die auf das Carrybit wirken und mit denen man sicherstellen kann, daß dieses bit einen definierten Anfangszustand einnimmt. So kann man das Carrybit mit dem Befehl STC setzen. Um das Carrybit zu löschen, könnte man die Befehlsfolge

```
STC
CMC
```

anwenden. Eleganter ist es jedoch, den Befehl ANA A oder ORA A zu geben. Beide Befehle ändern den Akkuinhalt nicht; das Carrybit aber wird gelöscht. Der Vorteil ist, daß man so nur einen Speicherplatz benötigt; außerdem geht es schneller.

**6.** Programm: Wie Programm 4, jedoch soll angehalten werden, wenn die obersten 4 bits im Akku besetzt sind

Wenn im Akku die obersten 4 bits besetzt sind, dann ist sein Inhalt in hexadezimaler Schreibweise F0. Mit diesem Wert müssen wir den Akkuinhalt nach jeder Inkrementierung vergleichen; das geht mit dem Befehl CPI F0. Bei Gleichheit wird das Z-bit gesetzt, und dann können wir das Programm anhalten.

```
0800    3E 80       MVI A,80
0802    D3 07       OUT 07      ;Ports vorbereiten
0804    AF          XRA A       ;alle Akkubits löschen
0805    3C          INR A       ;inkrementieren
0806    D3 04       OUT 04      ;zur Anzeige bringen
0808    CD BD 03    CALL 03BD   ;und eine Zehntelsekunde warten
080B    FE F0       CPI F0      ;Ziel erreicht?
080D    CA 13 08    JZ 0813     ;ja, anhalten
0810    C3 05 08    JMP 0805    ;nein, weitermachen
0813    76          HLT
```

Nur wenn das Z.bit gesetzt ist, wird der bedingte Sprung JZ ausgeführt, andernfalls wird die Programmschleife erneut durchlaufen.

In diesem Beispiel haben wir einen Vergleichsbefehl und einen bedingten Sprung kennengelernt. Wenn im Akku bis zum Wert F0 hochgezählt wird, muß der Vergleich knapp 240 Mal ausgeführt werden, also recht oft. Nun kommt es bei unseren Beispielen auf die Programmlaufzeit nicht an; im Gegenteil, wir bauen Verzögerungen ein, um gut verfolgen zu können, was sich da tut. Bei zeitkritischen Programmen würde man anders programmieren. So könnte man gleich am Programmanfang ein Register mit der abzufragenden Konstanten laden, etwa mit MVI B,F0. Der Vergleich würde dann mit dem Befehl CMP B ausgeführt, und das geht fast doppelt so schnell. Wir wollen uns grundsätzlich merken: Häufig auszuführende Operationen immer in den Registern ablaufen lassen!

**7.** Programm: Legt man an einen Eingang von Port B logisch 1 an, dann soll das zugehörige Bit in Port A gesetzt werden.

Aus der Aufgabenstellung ergibt sich, daß Port A als Ausgang und Port B als Eingang bestimmt werden muß. Logisch 1, also H-Pegel, steht auf der Frontplatte des Gerätes nicht zur Verfügung. Wir definieren deshalb Port C ebenfalls als Ausgang und schreiben FF hinein, dann haben wir logisch 1 gleich achtmal zur Verfügung.

```
0800    3E 82       MVI A,82    ;A und C = Ausgang, B = Eingang
0802    D3 07       OUT 07
0804    3E FF       MVI A,FF
0806    D3 06       OUT 06      ;alle bits von Port C = 1
0808    AF          XRA A       ;zu Anfang sei alles 0, auch das
0809    47          MOV B,A     ;B-Register, das wir als Hilfsreg. benötigen
080A    DB 05       IN 05       ;Port B abfragen
080C    B0          ORA B       ;alt und neu mit ODER verknüpfen
080D    47          MOV B,A     ;das neue Ergebnis wegspeichern
080E    D3 04       OUT 04      ;und anzeigen
0810    C3 0A 08    JMP 080A    ;das Spiel kann weitergehen
```

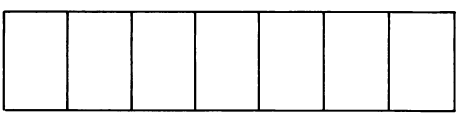
Um mit diesem Programm zu arbeiten, benötigen wir eine kurze Verbindungsschnur mit 2-mm-Steckern. Wir stecken ein Ende in irgendeine Buchse von Port C. Berühren wir nun mit dem anderen Ende eine Buchse von Port B, dann wird das entsprechende bit von Port A gesetzt, und die zugehörige LED leuchtet.

In diesem Programm habe wir die Benutzung des IN-Befehls, der ODER-Verknüpfung und eines Hilfsregisters kennengelernt. Wir wollen uns nun der Programmierung der Siebensegmentanzeige zuwenden und weitere Befehle und Monitor-Routinen kennenlernen. Fahren Sie aber bitte mit den Programmierübungen nur dann fort, wenn Sie bis hierher alles restlos verstanden haben!

## Die Programmierung der Siebensegmentanzeige

Die Ports (besonders Port A) sind für die ersten Programmier-Gehversuche gut geeignet; wichtigstes und natürlich auch vielseitigeres Ausgabemedium ist aber die Siebensegmentanzeige. Mit ihrer Programmierung wollen wir uns deshalb eingehend beschäftigen.

Jede der sieben Siebensegmentanzeigen wird wie ein Port angesprochen. In der nachfolgenden Skizze sind die Anzeigen und darunter die zugehörigen Port-Adressen dargestellt.

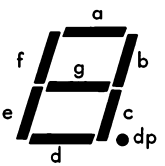


70 60 50 40 30 20 10

Es wird also z.B. mit dem Befehl OUT 10 der Inhalt des Akkus in die rechte Anzeige gebracht, d.h. die Anzeige leuchtet entsprechend dem Akkuinhalt. Nun ist es allerdings nicht so, daß etwa in der Anzeige eine 5 erscheint, wenn der Akkuinhalt 05 ist. Der Akku kann ja maximal FF (oder dezimal 255) enthalten, und wie will man das in einer einzigen Stelle anzeigen?

Der Zusammenhang ist anders, und zwar wie folgt: (Betrachten Sie bitte die Abbildung auf der nächsten Seite oben.)

Die sieben Segmente einer Siebensegmentanzeige werden fortlaufend mit den Buchstaben a bis g bezeichnet; der ebenfalls vorhandene Dezimalpunkt bekommt die Bezeichnung dp. Das ergibt insgesamt acht einzelne Elemente, die unabhängig voneinander




nander leuchten können. Im Akku sind aber gerade acht bits vorhanden, und es besteht folgende Zuordnung zwischen Akku-bit und Element einer Siebensegmentanzeige:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| dp | g | f | e | d | c | b | a |

Wir wollen uns das gleich mit einem Programmbeispiel klarmachen. Zuvor muß bemerkt werden, daß die Anzeigen k e i n e speichernde Eigenschaft haben; der OUT-Befehl muß deshalb in eine Schleife eingebaut werden, damit das gewünschte Leuchtbild auch für längere Zeit erscheint.

**8.** Programm: Erzeugung eines Leuchtbildes in der Siebensegmentanzeige

```
0800    3E 01       MVI A,01    ;Segment a soll leuchten
0802    D3 10       OUT 10
0804    C3 00 08    JMP 0800
```

Dieses Prograämmchen bringt, wie zu erwarten war, ein einsames Segment in der rechten Anzeige zum Leuchten. Spielen Sie nun ein bißchen herum: Ändern Sie die Konstante in Zelle 0801 und beobachten Sie, wie das zugehörige Leuchtbild aussieht. Ändern Sie auch die Adresse in 0803 in 20, 30 usw., um auch andere als die rechte Anzeige anzusprechen. Nun verstehen Sie sicher auch, wie das folgende Programm arbeitet:

**9.** Programm: Erzeugung einer Anzeige

```
0800    AF          XRA A       ;die Befehle dieses Programms werden ausnahms-
0801    D3 10       OUT 10      ;weise einmal nicht kommentiert. Wenn Sie das
0803    D3 70       OUT 70      ;Programm eingegeben und gestartet haben, se-
0805    3E 79       MVI A,79    ;hen Sie ja, was dabei herauskommt. Machen
0807    D3 20       OUT 20      ;Sie sich klar, wie die Anzeige zustande kommt!
0809    3E 71       MVI A,71
080B    D3 30       OUT 30
080D    3E 38       MVI A,38
080F    D3 40       OUT 40
0811    3E 06       MVI A,06
0813    D3 50       OUT 50
0815    3E 76       MVI A,76
0817    D3 60       OUT 60
0819    C3 00 08    JMP 0800
```

Nun kommt es beim Umgang mit dem Rechner immer wieder vor, daß auf der Anzeige etwas dargestellt werden soll. Der Monitor enthält deshalb ein Unterprogramm, wel-

ches uns die Programmierung der Siebensegmentanzeige abnimmt. Es wird mit

CALL 011F

aufgerufen. Dieses Programm benutzt einen Hifsspeicher, und zwar die Zellen 0BF1 bis 0BF7. Hier muß das abgespeichert sein, was in den Anzeigen 1 bis 7 dargestellt werden soll (von rechts an gezählt). Der Zusammenhang zwischen den im Hilfsspeicher besetzten bits und den Segmenten der Anzeige ist dabei derselbe wie auf der vorigen Seite oben beschrieben. Sehen wir uns ein ganz einfaches Beispiel an:

**10.** Programm: In der Anzeige Nr. 4 soll eine 0 dargestellt werden

```
0800    3E 3F       MVI A,3F    ;der Siebensegmentcode für 0
0802    32 F4 0B    STA 0BF4    ;kommt in die Hilfszelle für die Anzeige 4
0805    CD 1F 01    CALL DISPL  ;Monitor-Unterprogramm DISPL aufrufen
0807    C3 05 08    JMP 0805    ;und für stehendes Bild sorgen
```

Als neuen Befehl lernen wir hier STA kennen. Außerdem wollen wir uns von nun an daran gewöhnen, Unterprogramme mit ihrem Namen anzusprechen und nicht mit ihrer Adresse, wie hier z.B. DISPL. Da die Namen ja tunlichst so gewählt sind, daß sie etwas über die Funktion des Unterprogramms aussagen, wird die Verständlichkeit der Programme sehr erhöht. Eingeben müssen wir natürlich immer die tatsächliche Adresse! Die Monitor-Unterprogramme sind am Schluß des Monitor-Listings (S. M12) zusammengestellt.

Bei dem eben angegebenen Beispielprogramm fällt auf, daß die Anzeigen rechts und links von der 0 dunkel sind, d.h. die Hilfszellen 0BF1 bis 0BF3 und 0BF5 bis 0BF7 müssen leer sein. Dafür sorgt das Monitor-Unterprogramm CLRDS (Adresse 0029), das jedesmal bei Betätigen der Taste RN aufgerufen wird. Es löscht die Zellen 0BF1 bis 0BF7. Wir haben natürlich von Anfang an beobachtet, daß die Anzeige bei Drücken von RN erlischt. Hier ist jedoch die passende Stelle, darüber zu sprechen.

Nun wollen wir einmal unter Benutzung von DISPL und der Hilfszellen etwas auf der Anzeige darstellen. Suchen Sie sich doch mal selbst ein Wort aus! Für das Beispielprogramm wird HALLO gewählt; dieses Wort soll in den Anzeigen 6 bis 2 erscheinen.

**11.** Programm: Darstellung eines Wortes, z.B. HALLO

Die Anzeigen 1 und 7 sollen dunkel bleiben; dafür sorgt ja automatisch CLRDS. Wir müssen nur die Zellen 0BF2 bis 0BF6 nacheinander mit dem richtigen Code füllen.

```
0800    21 F2 0B    LXI H,0BF2  ;Adresse der ersten Zelle ins HL-Register
0803    36 3F       MVI m,3F    ;Code für O nach 0BF2
0805    23          INX H       ;HL-Register um 1 erhöhen
0806    36 38       MVI m,38    ;Code für L nach 0BF3
0808    23          INX H       ;HL-Register um 1 erhöhen
0809    36 38       MVI m,38    ;Code für L nach 0BF4
080B    23          INX H       ;HL-Register um 1 erhöhen
080C    36 77       MVI m,77    ;Code für A nach 0BF5
080E    23          INX H       ;HL-Register um 1 erhöhen
080F    36 76       MVI m,76    ;Code für H nach 0BF6
0811    CD 1F 01    CALL DISPL  ;alles zur Anzeige bringen
0814    C3 11 08    JMP 0811
```

Dieses Beispiel bringt eine neue - und äußerst wichtige - Programmiertechnik, die nun besprochen werden soll.

Wenn man die Beschreibung der Befehle aufmerksam durchliest, fällt auf, daß das HL-Register eine besondere Stellung einnnimmt. Viele Befehle, die sich auf eine Speicherzelle beziehen, setzen voraus, daß die Adresse eben dieser Zelle im HL-Register enthalten ist. Sollen nun, was sehr häufig vorkommt, hintereinanderliegende Speicherzellen angesprochen werden (d.h. Zellen, deren Adressen aufeinanderfolgen), dann braucht man nur das HL-Register mit der ersten Adresse zu füllen und jedesmal, wenn eine andere Zelle angesprochen werden soll, um 1 zu erhöhen. Das ist auch im letzten Beispiel geschehen:

Mit dem LXI-Befehl wird die Adresse der ersten zu füllenden Zelle des Display-Hilfsspeichers ins HL-Register geladen. Nun ist der Befehl MVI m anwendbar, undfortlaufendes Inkrementieren des HL-Registers macht die Zellen nacheinander zugänglich. Wir wollen auch festhalten, daß dieses Programm etwas kürzer ist als Programm Nr. 9, obwohl beide im Prinzip dasselbe bewirken.

**12.** Programm: Eingeben einer Ziffer

Hierfür läßt sich das Monitor-Unterprogramm NIBIN verwenden. Es liest ein Nibble (= ein halbes Byte, also eine einstellige Hexadezimalziffer) ein, stellt es auf der Anzeige dar und schreibt es nach Drücken von NX in den Akku ein. Das wollen wir mit einem Testprogramm untersuchen.

```
0800    3E 80       MVI A,80
0802    D3 07       OUT 07      ;Port vorbereiten
0804    CD 40 03    CALL NIBIN  ;Unterprogramm aufrufen
0807    CD A2 03    CALL NEXT   ;Taste NX gedrückt?
080A    D2 04 08    JNC 0804    ;nein, zurück in die Schleife
080D    D3 04       OUT 04      ;ja, Akkuinhalt in den LEDs darstellen
080F    C3 04 08    JMP 0804    ;und in die Schleife zurückspringen
```

Wenn das Programm läuft, sieht man, daß der Wert der gedrückten Taste erst nach Betätigen von NX im Akku (und damit in der LED-Zeile) steht.

Ähnliches bewirkt das Unterprogramm INPUT, nur daß hier zwei statt einer Ziffer verarbeitet werden. Wir können dasselbe Testprogramm benutzen und brauchen nur den Befehl in 0804 in CALL INPUT (CD 5A 03) zu ändern.

In diesem Beispiel lernen wir auch eine interessante und nützliche Anwendung des Monitor-Unterprogrammes NEXT kennen. Wir wollen uns diesen Teil des Monitors einmal näher ansehen.

**13.** Programm: Analyse des Monitor-Unterprogramms NEXT

```
03A2    CD 65 01    CALL CCKEY  ;Funktionstasten abfragen
03A5    D0          RNC         ;Rücksprung, wenn Carrybit nicht gesetzt
03A6    FE 00       CPI 00      ;eine andere als die Taste NX gedrückt?
03A8    C0          RNZ         ;wenn ja, ebenfalls Rücksprung
03A9    3A F8 0B    LDA 0BF8    ;Umspeichern des Inhalts von 0BF8
03AC    32 FE 0B    STA 0BFE    ;nach 0BFE
03AF    37          STC         ;Carrybit setzen
03B0    C9          RET         ;und Rücksprung
```

Abgesehen davon, daß der Inhalt von 0BF8 nach 0BFE umgespeichert wird (der Monitor will das so), ist der einzige Effekt dieses Unterprogramms der, das Carrybit zu setzen. Das können wir nutzen, um die NX-Taste als Befehlstaste besonderer Art, z.B. als Abschluß einer Eingabe oder zum Eingriff in ein laufendes Programm, zu verwenden.

## **14.** Programm: Eingabe einer mehrstelligen Zahl

Dieses wird nun schon ein etwas längeres Programm. Wir werden später Programme kennenlernen, die eine Anfangs- und eine Endadresse benötigen, z.B. zum Füllen eines Speicherbereichs mit einer Konstanten. Diese Adressen müssen ja eingegeben werden; es sind vierstellige Hex-Zahlen. Aus diesem Grund ist das folgende Programm für eine vierstellige Zahl geschrieben. Unten wird noch angegeben, wie es leicht auf eine andere Stellenzahl modifiziert werden kann.

Das Programm benötigt für die eingegebenen Ziffern einen Hilfsspeicher; dieser ist hier auf 0BB0 bis 0BB3 gelegt worden; natürlich sind auch andere Adressen möglich. In jedem Hilfsspeicher steht nur e i n e Ziffer, nämlich im unteren Halbbyte (Nibble). Das obere Nibble ist =0.

Zusammengefaßt ist der Programmablauf folgender: Zunächst wird die Hexadezimaltastatur nach einer gedrückten Taste abgefragt. Ist eine Taste gedrückt, dann steht der Wert im Register B. Nun wird der gesamte Hifsspeicher um eine Zelle nach links verschoben, und der Tastenwert kommt in die freiwerdende Zelle. Schließlich wird der Inhalt der vier Hilfsspeicherzellen auf dem Display angezeigt.

```
0800  CD 61 00      CALL HHKEY  ;Hexadezimaltasten abfragen
0803  DC 10 08      CC SH       ;wenn Taste gedrückt, Hilfsspeicher shiften
0806  CD 28 08      CALL AN     ;Hilfsspeicher-Inhalt anzeigen
0809  C9            RET         ;Rücksprung. Das Ganze ist ein Unterprog.!

0810  21 B3 0B SH:  LXI H,0BB3  ;oberste Adresse des Hilfsspeichers ins HL-Reg.
0813  11 B2 0B      LXI D,0BB2  ;und die nächstniedrigere ins DE-Register
0816  0E 03         MVI C,03    ;Stellenzahl minus 1 ins C-Register
0818  1A       SL:  LDAX D      ;erste Ziffer laden
0819  77            MOV m,A     ;und einen Speicherplatz höher wieder wegsp.
081A  2B            DCX H       ;HL-Register um 1 erniedrigen
081B  1B            DCX D       ;das DE-Register auch
081C  0D            DCR C       ;ebenso das C-Register
081D  C2 18 08      JNZ SL      ;weitermachen, wenn C noch nicht leer ist
0820  70            MOV m,B     ;ansonsten die neue Zahl abspeichern
0821  CD 61 00 LS:  CALL HHKEY  ;mit diesen letzten drei Befehlen
0824  D0            RNC         ;wird darauf gewartet, daß die Taste
0825  C3 21 08      JMP LS      ;wieder losgelassen wird

0828  21 B0 0B AN:  LXI H,0BB0  ;Anfangsadresse des Ziffernspeichers nach HL
082B  11 F1 0B      LXI D,0BF1  ; anfangsadr. des Display-Puffers nach DE
082E  0E 04         MVI C,04    ;Anzahl der anzuzeigenden Ziffern nach C
0830  7E       AL:  MOV A,m     ;erste Ziffer holen
0831  E5            PUSH H      ;der nächste Befehl benutzt HL, deshalb retten
0832  CD E2 00      CALL SSGCV  ;in Siebensegment-Balkencode umwandeln
0835  E1            POP H       ;HL-Register restaurieren
0836  23            INX H       ;inkrementieren
0837  0D            DCR C       ;und C-Register dekrementieren
```

```
0838   C2 2E 08     JNZ AL       ;C nicht leer? Weitermachen! Ansonsten
083B   CD 1F 01     CALL DISPL   ;das Ganze anzeigen
083E   C9           RET          ;und zurückspringen
```

In diesem Programm benutzen wir zum ersten Mal symbolische Adressen. So bedeutet z.B. CALL AN natürlich CALL 0826. Da wir aber zu Anfang noch nicht wissen, an welcher Stelle im Speicher das Anzeige-Unterprogramm stehen wird, kennzeichnen wir die Adresse durch das Symbol AN. Später wird die betreffende Stelle durch das Symbol AN: markiert.

Das Monitor-Unterprogramm HHKEY werden wir uns noch ansehen; das Unterprogramm SSGCV sorgt dafür, daß der zu einer Hexadezimalzahl gehörende Balkencode entsprechend der Auf Seite P7 oben beschriebenen Vorschrift bereitgestellt wird.

Wenn Sie das Programm nun laufen lassen und z.B. die Zahl 12AB eintippen, dann haben die

| Hilfsspeicherzellen | 0BB3 | 0BB2 | 0BB1 | 0BB0 |
|---|---|---|---|---|
| den Inhalt | 01 | 02 | 0A | 0B |

wie Sie sich durch Anwählen der Adressen leicht überzeugen können. Wie aber bastelt man daraus eine vierstellige Adresse? Das soll in dem nächsten Programmbeispiel gezeigt werden.

**15.** Programm: Assemblieren der 4 eingegebenen Ziffern zu einer Adresse

Dazu müssen wir lediglich das höherwertige Adressenbyte aus den Zellen 0BB3 und 0BB2 zusammenbauen und das niederwertige Adressenbyte aus den Inhalten der Zellen 0BB1 und 0BB0. Die fertige Adresse wollen wir wieder in zwei Hilfsspeicherzellen ablegen und wählen dafür die Zellen 0BB4 und 0BB5.

Um ein Adressenbyte aus zwei der oben angegebenen Hilfsspeicher zusammenzufügen, braucht man nur das höherwertige Nibble zu holen, viermal linkszuschieben und das niederwertige Nibble hinzuzuaddieren. Da die komplette Adresse ja aus zwei Bytes besteht, muß dieser Vorgang ganz offensichtlich zweimal ausgeführt werden; wir machen deshalb ein Unterprogramm.

Dieses nun folgende Unterprogramm setzt voraus, daß sich die Adresse des höherwertigen Nibbles im HL-Register befindet.

```
0840   7E        AD: MOV A,m   ;höherwertiges Nibble in den Akku holen
0841   07            RLC       ;von
0842   07            RLC       ;unten
0843   07            RLC       ;nach
0844   07            RLC       ;oben schieben
0845   2B            DCX H     ;Adresse des anderen Halbbytes bilden
0846   86            ADD m     ;und dieses hinzuaddieren
0847   C9            RET       ;nun steht das eine Adressenbyte im Akku
```

Dieses Programm besteht zufällig aus lauter Ein-Byte-Befehlen. Damit haben wir aber erst e i n Adressenbyte aufgebaut. Es muß nun noch das niederwertige Byte aufgebaut werden, und die fertige Adresse soll in die Zellen 0BB4 und 0BB5. Das besorgt das Programm auf der nächsten Seite.

```
0848   21 B3 0B VA: LXI H,0BB3   ;oberste Adresse des Ziffernspeichers ins HL-Reg.
084B   CD 40 08      CALL AD     ;erstes Byte bilden
084E   57            MOV D,A     ;und im D-Register zwischenspeichern
084F   2B            DCX H       ;die nächsten Hilfszellen ansprechen
0850   CD 40 08      CALL AD     ;und zum zweiten Byte vereinen
0853   5F            MOV E,A     ;welches in E zwischengespeichert wird
0854   EB            XCHG        ;die komplette Adresse kommt ins HL-Register
0855   22 B4 0B      SHLD 0BB4   ;und wird in 0BB4 - 0BB5 abgelegt
```

Dieses Programm demonstriert wunderbar die Anwendung der Befehle XCHG und SHLD. XCHG bringt bekanntlich den Inhalt des DE-Registerpaares in das HL-Registerpaar, und SHLD speichert dann den Inhalt des HL-Registerpaares in zwei aufeinanderfolgende Adressen. Beide Befehle sind 16-bit-Operationen!

Aus Gründen, die später noch deutlich werden, wollen wir auch aus diesem Programm wieder ein Unterprogramm machen. Wir lassen den SHLD-Befehl weg und überlassen es dem rufenden Programm, den Speicherplatz für die abzulegende Adresse anzugeben. In Zelle 0855 kommt dann C9, der Hex-Kode für RET.

**16.** Programm: Aufruf des Eingabeprogramms (Nr. 14) und anschließende Abfrage von NX

Wenn wir mit Hilfe von Programm Nr. 14 eine Adresse eingegeben haben, müssen wir irgendwie angeben, daß die Eingabe beendet ist und daß nun etwas anderes getan werden soll. Das können wir, wie schon auf Seite P9 unten und P10 oben gesagt, mit der Taste NX erreichen. Dies erledigt das nachfolgende kleine Programm, welches auch wieder als Unterprogramm geschrieben ist:

```
0856   CD 00 08 CN: CALL 0800    ;rufe das Eingabeprogramm
0859   CD A2 03     CALL NEXT    ;und frage die NX-Taste ab
085C   D2  56 08     JNC 0856     ;rufe erneut(hauptsächlich wegen DISPL)
085F   CD 65 01      CALL CCKEY  ;jetzt ist NX gedrückt...
0862   D0            RNC         ;...und jetzt losgelassen.
0863   C3 5F 08      JMP 085F    ;noch nicht losgelassen - weiter warten
```

Jetzt wollen wir ganz komfortabel werden. Wenn wir ein Programm in einen anderen Speicherbereich verschieben wollen, dann müssen wir vom Programm die Anfangs- und Endadresse angeben sowie die Zieladresse. Ohne uns zunächst um das eigentliche Transferprogramm zu kümmern (das kommt später noch), machen wir ein Programm für die Adresseneingabe.

**17.** Programm: Eingabe von drei Adressen in die Zellen 0BB4 bis 0BB9

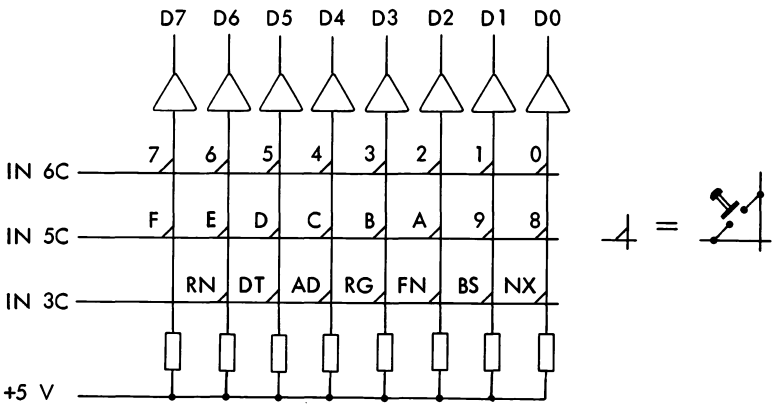
```
0870   3E 77     AE: MVI A,77      ;Leuchtbalkenmuster für A (Anfangsadresse)
0872   32 F7 0B      STA 0BF7     ;jetzt leuchtet ganz links ein A
0875   3E 48         MVI A,48     ;leuchtbalkenmuster für =
0877   32 F6 0B      STA 0BF6     ;in der zweiten Stelle von links leuchtet =

087A   CD 56 08      CALL CN      ;erste Adresse eingeben
087D   CD 48 08      CALL VA      ;assemblieren
0880   22 B4 0B      SHLD 0BB4    ;und an vorgegebner Stelle ablegen

0883   3E 79         MVI A,79     ;jetzt wird an der Stelle von A
0885   32 F7 0B      STA 0BF7     ;ein E (Endadresse) zum Leuchten gebracht
```

```
0888   CD 56 08   CALL CN    ;zweite Adresse eingeben
088B   CD 48 08   CALL VA    ;assemblieren
088E   22 B6 0B   SHLD 0BB6  ;und ablegen

0891   3E 1B      MVI A,1B   ;das Balkenmuster für ein etwas komisches Z
0893   32 F7 0B   STA 0BF7   ;leuchtet jetzt ganz links

0896   CD 56 08   CALL CN    ;dritte Adresse eingeben
0899   CD 48 08   CALL VA    ;assemblieren
089C   22 B8 0B   SHLD 0BB8  ;und ablegen

089F   C9         RET        ;und auch das ist wieder ein Unterprogramm!
```

Das Unterprogramm Nr. 14 ist ein sehr nützliches Programm zur Zifferneingabe allgemein. In der abgedruckten Form verarbeitet es, wie wir wissen, vier Ziffern. Es kann durch Modifikation von nur 4 Speicherplätzen für eine beliebige Anzahl von Ziffern umgeschrieben werden:

In Zelle 082F steht die Anzahl der einzugebenden Ziffern; in Zelle 0817 steht diese Anzahl minus 1. Der Ziffernspeicher reicht von 0BB0 bis zu der Adresse, die in den Zellen 0811 und 0812 steht. Um 1 verringert steht diese Adresse noch in den Zellen 0814 und 0815.

Am Anfang unserer Programmierübungen haben wir u.a. den Befehl OUT XX kennengelernt. Wie aber bekommt man Daten in den Rechner hinein? Das Drücken einer Taste muß doch irgendwelche Aktivitäten auslösen, die schließlich dazu führen, daß der Wert in der Anzeige erscheint. Wie schon bemerkt, wird diese Aufgabe von dem Monitor-Unterprogramm HHKEY, welches wir ja auch schon benutzt haben, übernommen. Dieses Programm wollen wir nun einmal unter die Lupe nehmen. Dazu ist es allerdings erforderlich, die zugehörige Hardware ebenfalls zu betrachten. Nachfolgend ist deshalb ein Schaltungsauszug des Rechners dargestellt, der die Tastatur betrifft.




Die Ausgänge D0 bis D7 sind an den Datenbus des Systems angeschlossen. Wenn keine Taste gedrückt ist, sind alle Ausgänge auf H-Pegel, d.h. das von der Tastatur ausgegebene Datenwort ist FF. Gibt man nun z.B. den Befehl IN 6C, dann wird die zugehörige Leitung auf L-Pegel gelegt. Wenn nun zu diesem Zeitpunkt auch eine Taste ge-

drückt ist, z.B. die Taste „4", geht auch das entsprechende Datenbit auf L. Das Datenwort wäre dann in unserem Beispiel EF oder in dualer Schreibweise 11101111.

**18.** Programm: Analyse des Monitor-Unterprogramms HKEY

```
0075  2E 00        MVI L,00    ;das L-Register enthält später den Tastencode
0077  DB 6C        IN 6C       ;Tasten 0 bis 7 abfragen
0079  00           NOP         ;Leitungsverzögerungen berücksichtigen
007A  EE FF        XRI FF      ;ist eine Taste gedrückt?
007C  C2 87 00     JNZ CN      ;ja, entsprechenden Code erzeugen
007F  2E 08        MVI L,08    ;die Tasten der 2. Reihe sind um 8 höherwertig
0081  DB 5C        IN 5C       ;vielleicht wurde hier eine gedrückt?
0083  00           NOP
0084  EE FF        XRI FF      ;das wollen wir feststellen!
0086  C8           RZ          ;nein, Routine verlassen
0087  0F        CN:RRC         ;verschiebe den Akkuinhalt so lange nach rechts,
0088  DA 8F 00     JC MV       ;bis das Carrybit gesetzt ist, dann sind wir fertig
008B  2C           INR L       ;ansonsten das L-Register inkrementieren
008C  C3 87 00     JMP CN      ;und weitermachen
008F  7D        MV:MOV A,L     ;nun enthält auch der Akku den Tastencode
0090  C9           RET         ;und wir sind fertig
```

Der Befehl XRI FF erfordert eine eingehendere Betrachtung. Wenn keine Taste gedrückt wurde, enthält der Akku nach der Abfrage FF. Wenn man nun FF mit FF exklusiv ODER verknüpft, kommt 0 heraus, und das Z-bit wird gesetzt. Dann jedoch wird der Befehl JNZ nicht ausgeführt, und die zweite Tastenreihe wird abgefragt.

Wenn aber eine Taste gedrückt wurde, hat genau ein bit im Akku den Wert 0. Jetzt wirkt der Befehl XRI FF so, als ob der Akkuinhalt komplementiert würde; aus dem 0-bit wird ein besetztes bit, und alle anderen werden zu 0. Das Z-bit wird nicht gesetzt, und es wird zu dem Programmteil CN (continue) gesprungen.

Hier wird das bit im Akku nun so lange rechtsverschoben, bis es in das Carrybit gelangt. Wurde z.B. die Taste „0" gedrückt, dann ist bit 0 im Akku besetzt, und schon nach der ersten Rechtsverschiebung wird das Carrybit gesetzt. In diesem Fall wird gesprungen, das L-Register wird nicht inkrementiert und bleibt dementsprechend null!

Ergibt die Abfrage der 1. Tastenreihe null, dann könnte ja immer noch in der 2. Reihe eine Taste gedrückt sein. Vor der Abfrage wird dann eine 8 ins L-Register gebracht, da die Tasten in dieser Reihe immer um 8 höherwertig sind als die korrespondierenden Tasten in der 1. Reihe.

Dieses Programm ist ein herrliches Beispiel für das Zusammenspiel von Hard- und Software und für eine äußerst sinnreiche Anwendung des XRI-Befehls. Nun können wir (was wir uns ja eigentlich vorgenommen haben) uns dem Programm HHKEY zuwenden.

Dieses Programm ruft das Unterprogramm HKEY zweimal auf; das Ergebnis des ersten Aufrufs wird mit dem des zweiten Aufrufs verglichen. Zwischen beiden Aufrufen ist eine Zeitverzögerung von einer Millisekunde eingebaut; damit wird berücksichtigt, daß der Tastenkontakt prellt. Erst wenn die Kontaktgabe einwandfrei ist und beide Aufrufe dasselbe Ergebnis liefern, wird das Carrybit gesetzt und die Routine beendet; ansonsten wird die Routine mit gelöschtem Carrybit verlassen. Der Tastencode steht bei erfolgreicher Abfrage in A, B und L.

```
0061  CD 75 00        CALL HKEY   ;Tastatur zum ersten Mal abfragen
0064  D0              RNC         ;kein Carrybit, keine Taste - fertig!
0065  47              MOV B,A     ;Ergebnis der 1. Abfrage sicherstellen
0066  CD C7 03        CALL DELY1  ;und eine Millisekunde warten, dann
0069  CD 75 00        CALL HKEY   ;die Tastatur zum zweiten Mal abfragen
006C  B8              CMP B       ;ist das Ergebnis dasselbe?
006D  CA 73 00        JZ SC       ;ja, Carrybit setzen und Routine beenden
0070  37              STC         ;nein, Carrybit
0071  3F              CMC         ;löschen und
0072  C9              RET         ;zurückspringen
0073  37          SC: STC
0074  C9              RET
```

Einige oft benötigte Routinen

**19.** Programm: Füllen eines Speicherbereichs mit einer Konstanten

Das Programm setzt voraus, daß in 0BB4/5 die Anfangsadresse, in 0BB6/7 die Endadresse und in 0BB8 die Konstante enthalten ist. Der erste Teil enthält eine 16-Bit-Subtraktion und wurde deshalb als Unterprogramm geschrieben.

```
0800  2A B4 0B DF: LHLD 0BB4  ;Anfangsadresse holen
0803  EB           XCHG       ;und nach DE umspeichern
0804  3A B6 0B     LDA 0BB6   ;unteres Halbbyte der Endadresse holen
0807  93           SUB E      ;davon das untere Halbbyte der Anfangsadr abz.
0808  4F           MOV C,A    ;und das Ergebnis ins C-Register bringen
0809  3A B7 0B     LDA 0BB7   ;oberes Halbbyte der Endadresse holen
080C  9A           SBB D      ;Anfangsadr.-Halbbyte abziehen, C berücksicht.
080D  47           MOV B,A    ;und das Ergebnis ins B-Register bringen
080E  C9           RET        ;die Differenz steht nun im BC-Register

0810  CD 00 08     CALL DF    ;hier startet das eigentliche Programm
0813  03           INX BC     ;wenn AA = EA, wird diese gefüllt!
0814  2A B4 0B     LHLD 0BB4  ;Anfangsadresse ins HL-Register
0817  3A B8 0B     LDA 0BB8   ;und die Konstante in den Akku
081A  77           MOV m,A    ;und von da aus in die Speicherzelle
081B  0B           DCX BC     ;Zähler erniedrigen
081C  F5           PUSH PSW   ;Akku retten, denn DCX ändert keine Zustands-
081D  78           MOV A,B    ;bits, und deshalb diese umständliche Art der
081E  B1           ORA C      ;Nullabfrage
081F  CA 27 08     JZ 0827    ;
0822  F1           POP PSW    ;Akkuinhalt wiederholen
0823  23           INX HL     ;nächste Speicherzelle ansprechen
0824  C3 1A 08     JMP 081A   ;und weitermachen
0827  76           HLT        ;geschafft!
```

**20.** Programm: Umspeichern eines Speicherbereichs

Das Programm setzt voraus, daß in 0BB4/5 die Anfangsadresse, in 0BB6/7 die Endadresse und in 0BB8/9 die Zieladresse enthalten ist. Das Unterprogramm des vorigen Beispiels zur 16-Bit-Subtraktion wird benutzt. Zu beachten ist, daß das vorige und dieses Programm für den gesamten Adressenbereich geeignet sind.

```
0830    CD 00 08        CALL DF         ;Anzahl berechnen
0833    03              INX BC          ;hier muß es eins mehr sein!
0834    2A B8 0B        LHLD 0BB8       ;hole die Zieladresse
0837    EB              XCHG            ;und packe sie ins DE-Register
0838    2A B4 0B        LHLD 0BB4       ;hole die Anfangsadresse
083b    7E              MOV A,m         ;Inhalt einer Zelle holen
083C    12              STAX DE         ;und an neuer Stelle ablegen
083D    0B              DCX BC          ;Zähler um 1 vermindern
083E    78              MOV A,B         ;und feststellen,
083F    B1              ORA C           ;ob wir fertig sind
0840    CA 48 08        JZ 0848         ;wenn ja, Ende der Vorstellung
0843    23              INX HL          ;wenn nein, die Register erhöhen,
0844    13              INX DE          ;die die Adressen enthalten
0845    C3 3B 08        JMP 083B        ;und weitermachen
0848    76              HLT
```

**21.** Programm: Multiplikation zweier vorzeichenloser 8-bit-Binärzahlen

Das Programm setzt voraus, daß der Multiplikand in Zelle 0BB4 und der Multiplikator in Zelle 0BB5 enthalten ist. Das Ergebnis wird in den Zellen 0BB6/7 abgelegt.

Wie im täglichen Leben wird der Multiplikand mit jeder einzelnen Stelle des Multiplikators multipliziert. Da es sich um Binärzahlen handelt, erfolgt die Multiplikation entweder mit 0 oder mit 1. Im ersten Fall ist das Ergebnis offensichtlich null; im zweiten Fall ist das Ergebnis gleich dem Multiplikanden. Damit reduziert sich die Multiplikation auf das einfache Verfahren:

Wenn das momentan multiplizierende bit im Multiplikator 1 ist, addiere den Multiplikanden zum Teilprodukt.

Zur Beachtung des Stellenwertes muß allerdings - auch wie bei der schriftlichen Multiplikation korrekt geschoben werden.

```
0800    21 B4 0B        LXI H,0BB4
0803    5E              MOV E,m         ;hole Multiplikand
0804    16 00           MVI D,00        ;erweitere auf 16 bits
0806    23              INX H
0807    7E              MOV A,m         ;hole Multiplikator
0808    21 00 00        LXI HL,0000     ;Voreinstellung Produkt = 0
080B    06 08           MVI B,08        ;Zähler für die acht Multiplikationen
080D    29          MU: DAD H           ;Produkt = Produkt mal 2 (Linksverschiebung!)
080E    17              RAL             ;Übertrag vom Multiplikator untersuchen
080F    D2 13 08        JNC CT          ;wenn =0, weitermachen
0812    19              DAD D           ;wenn =1, Produkt = Produkt + Multiplikand
0813    05          CT: DCR B           ;mal sehen, ob wir fertig sind
0814    C2 0D 08        JNZ MU          ;wenn nicht, auf ein neues!
0817    22 B6 0B        SHLD 0BB6       ;Ergebnis abspeichern
081A    C9              RET             ;das sollte man tunlichst zur Routine erklären!
```

Interessant an diesem Beispiel ist die Verwendung der 16-bit-Befehle LXI und DAD zur Verarbeitung von Daten anstatt von Adressen.

```
0000  31 E8 0B   INIT    LXI SP,0BE8
   3  E5                PUSH H
   4  CD 65 02          CALL SAVE
   7  E1                POP H
   8  22 E8 0B          SHLD 0BE8
   B  00                NOP
   C  3E 01       MAIN    MVI A,01
   E  32 FF 0B          STA 0BFF
  11  21 00 08          LXI H,0800
   4  22 F9 0B          SHLD 0BF9
   7  CD 29 00          CALL CLRDS
   A  CD 12 01          CALL READ
   D  CD 50 00    MAINA   CALL HEXKY
  20  CD 19 01          CALL READ2
   3  CD 5A 01          CALL CMDKY
   6  C3 1D 00          JMP MAINA
------------------------------------
0029  AF         CLRDS   XRA A
   A  0E 08             MVI C,08
   C  11 F0 0B          LXI D,0BF0
   F  12         CLRLP   STAX D
  30  1C                INR E
   1  0D                DCR C
   2  C2 2F 00          JNZ CLRLP
   5  C9                RET
------------------------------------
0036  31 E8 0B   SOFTM   LXI SP,0BE8
   9  C3 1D 00          JMP MAINA
003C  C3 3C 08   RST75   JMP ISR
   F  FF
------------------------------------
0040  49         REGTB   Code für ≡ ≡ ≡ ≡ ≡ ≡ ≡ ≡
   1  49
   2  49
   3  49
   4  49
   5  49
   6  49
   7  49
   8  38                Code für L H A b C d E F
   9  76
   A  77
   B  7C
   C  39
   D  5E
   E  79
004F  71
```

```
0050  CD 61 00  HEXKY   CALL HHKEY
   3  D0        END1    RNC
   4  CD 91 00          CALL SHIFT
   7  CD BC 00          CALL CONVT
   A  CD ED 00          CALL DECPT
   D  CD 08 01          CALL RELSH
  60  C9                RET
------------------------------------
0061  CD 75 00  HHKEY    CALL HKEY
   4  D0        END2    RNC
   5  47                MOV B,A
   6  CD C7 03          CALL DELY1
   9  CD 75 00          CALL HKEY
   C  B8                CMP B
   D  CA 73 00          JZ SETCY
  70  37        CLRCY   STC
   1  3F                CMC
   2  C9                RET
   3  37        SETCY   STC
   4  C9                RET
------------------------------------
0075  2E 00     HKEY    MVI L,00
   7  DB 6C             IN 6C
   9  00                NOP
   A  EE FF             XRI FF
   C  C2 87 00          JNZ CNT
   F  2E 08             MVI L,08
  81  DB 5C             IN 5C
   3  00                NOP
   4  EE FF             XRI FF
   6  C8        END3    RZ
   7  0F        CNT     RRC
   8  DA 8F 00          JC MOVE
   B  2C                INR L
   C  C3 87 00          JMP CNT
   F  7D        MOVE    MOV A,L
  90  C9                RET
------------------------------------
0091  11 F8 0B  SHIFT    LXI D,0BF8
   4  3A FF 0B          LDA 0BFF
   7  E6 05             ANI 05
   9  C2 AD 00          JNZ SFT1
   C  1C        SFT2    INR E
   D  CD B2 00          CALL SPLIT
  A0  1A                LDAX D
   1  E6 F0             ANI F0
   3  0F                RRC
   4  0F                RRC
   5  0F                RRC
   6  0F                RRC
   7  4F                MOV C,A
   8  78                MOV A,B
   9  12                STAX D
```

```
00AA  79                    MOV A,C
   B  47                    MOV B,A
   C  1C                    INR E
   D  CD B2 00              CALL SPLIT
  B0  12                    STAX D
   1  C9                    RET
-------------------------------------------
00B2  1A          SPLIT     LDAX D
   3  E6 0F                 ANI 0F
   5  07                    RLC
   6  07                    RLC
   7  07                    RLC
   8  07                    RLC
   9  B0                    ORA B
   A  47                    MOV B,A
   B  C9                    RET
-------------------------------------------
00BC  11 F1 0B    CONVT     LXI D,0BF1
   F  3A F8 0B              LDA 0BF8
  C2  CD D5 00              CALL BYTE
   5  11 F4 0B    CNVAD     LXI D,0BF4
   8  3A F9 0B              LDA 0BF9
   B  CD D5 00              CALL BYTE
   E  3A FA 0B              LDA 0BFA
  D1  CD D5 00              CALL BYTE
   4  C9                    RET
-------------------------------------------
00D5  4F          BYTE      MOV C,A
   6  CD E2 00               CALL SSGCV
   9  79                    MOV A,C
   A  0F                    RRC
   B  0F                    RRC
   C  0F                    RRC
   D  0F                    RRC
   E  CD E2 00              CALL SSGCV
  E1  C9                    RET
-------------------------------------------
00E2  E6 0F       SSGCV     ANI 0F
   4  26 03                 MVI H,03
   6  C6 F0                 ADI F0
   8  6F                    MOV L,A
   9  7E                    MOV A,m
   A  12                    STAX D
   B  13                    INX D
   C  C9                    RET
-------------------------------------------
00ED  11 F4 0B    DECPT     LXI D,0BF4
  F0  3A FF 0B              LDA 0BFF
   3  E6 01                 ANI 01
   5  C2 01 01              JNZ DAT
   8  3A FF 0B              LDA 0BFF
   B  E6 02                 ANI 02
   D  C8             END1    RZ
   E  C3 03 01               JMP ADR
```

```
0101  1E F1        DAT       MVI E,F1
   3  1A           ADR       LDAX D
   4  F6 80                  ORI 80
   6  12                     STAX D
   7  C9                     RET
------------------------------------------
0108  CD 75 00     RELSH     CALL HKEY
   B  D0           END       RNC
   C  CD 19 01               CALL READ2
   F  C3 08 01               JMP RELSH
------------------------------------------
0112  2A F9 0B     READ      LHLD 0BF9
   5  7E                     MOV A,m
   6  32 F8 0B               STA 0BF8
0119  CD BC 00     READ2     CALL CONVT
   C  CD ED 00               CALL DECPT
   F  11 F8 0B     DISPL     LXI D,0BF8
  22  CD 4E 01               CALL NDIS
   5  D3 70                  OUT 70
   7  CD 4E 01               CALL NDIS
   A  D3 60                  OUT 60
   C  CD 4E 01               CALL NDIS
   F  D3 50                  OUT 50
  31  CD 4E 01     DIG4      CALL NDIS
   4  D3 40                  OUT 40
   6  CD 4E 01     DIG3      CALL NDIS
   9  D3 30                  OUT 30
   B  CD 4E 01               CALL NDIS
   E  D3 20                  OUT 20
  40  CD 4E 01               CALL NDIS
   3  D3 10                  OUT 10
   5  CD C7 03               CALL DELY1
   8  3A F8 0B               LDA 0BF8
   B  D3 00                  OUT 00
   D  C9                     RET
   E  CD C7 03     NDIS      CALL DELY1
  51  1B                     DCX DE
   2  1A                     LDAX DE
   3  C9                     RET
   4  C6 D1                  ADI D1
   6  00                     NOP
   7  C3 8B 01               JMP BRNCH+3
------------------------------------------
015A  CD 65 01     CMDKY     CALL CCKEY
   D  D0           END1      RNC
   E  CD 88 01               CALL BRNCH
  61  CD 93 01               CALL RELSC
   4  C9                     RET
```

```
0165  CD 76 01  CCKEY   CALL CKEY
   8  D0        END2    RNC
   9  47                MOV B,A
   A  CD C7 03          CALL DELY1
   D  CD 76 01          CALL CKEY
0170  B8                CMP B
   1  C2 70 00          JNZ CLRCY
   4  37        SETCY   STC
   5  C9                RET
-------------------------------------
0176  2E 00     CKEY    MVI L,00
   8  DB 3C             IN 3C
   A  00                NOP
   B  EE FF             XRI FF
   D  C8        END3    RZ
   E  0F        CNT     RRC
   F  DA 86 01          JC MOVE
  82  2C                INR L
   3  C3 7E 01          JMP CNT
   6  7D        MOVE    MOV A,L
   7  C9                RET
-------------------------------------
0188  07        BRNCH   RLC
   9  C6 E1             ADI E1
   B  26 03             MVI H,03
   D  6F                MOV L,A
   E  56        FETCH   MOV D,m
   F  2B                DCX H
  90  5E                MOV E,m
   1  EB                XCHG
   2  E9        JUMP    PCHL
-------------------------------------
0193  CD 76 01  RELSC   CALL CKEY
   6  D0        END1    RNC
   7  FE 07             CPI 07
   9  C8        END2    RZ
   A  CD 12 01          CALL READ
   D  C3 93 01          JMP RELSC
-------------------------------------
01A0  2A F9 0B  TNXT    LHLD 0BF9
   3  3A F8 0B          LDA 0BF8
   6  77                MOV m,A
   7  23                INX H
   8  22 F9 0B          SHLD 0BF9
   B  C9                RET
-------------------------------------
01AC  2A F9 0B  TBST    LHLD 0BF9
   F  2B                DCX H
  B0  22 F9 0B          SHLD 0BF9
   3  C9                RET
-------------------------------------
01B4  21 00 08  TADR    LXI H,0800
   7  22 F9 0B          SHLD 0BF9
   A  3A FF 0B  ADR2    LDA 0BFF
   D  E6 F0             ANI F0
   F  F6 02             ORI 02
  C1  32 FF 0B          STA 0BFF
   4  C9                RET
```

```
01C5  3A FF 0B   TDAT    LDA 0BFF
   8  E6 F0              ANI F0
   A  F6 01              ORI 01
   C  32 FF 0B           STA 0BFF
1CF   C9                 RET
-------------------------------------
01D0  CD 29 00   TRUN    CALL CLRDS
   3  D3 00              OUT 00
   5  2A F9 0B           LHLD 0BF9
   8  E5                 PUSH H
   9  21 EA 0B           LXI H,0BEA
   C  7E                 MOV A,m
   D  2C                 INR L
   E  46                 MOV B,m
   F  2C                 INR L
  E0  4E                 MOV C,m
   1  2C                 INR L
   2  56                 MOV D,m
   3  2C                 INR L
   4  5E                 MOV E,m
   5  2A E8 0B           LHLD 0BE8
   8  C9         GO      RET
-------------------------------------
01E9  3E 08      TREG    MVI A,08
   B  32 F9 0B           STA 0BF9
   E  CD 50 00   REGLP   CALL HEXKY
  F1  CD 01 02           CALL CNVRG
   4  CD 2C 02           CALL CMDK2
   7  C3 EE 01           JMP REGLP
   A  CD 1A 02   REDRG   CALL REGNR
   D  1A                 LDAX D
   E  32 F8 0B           STA 0BF8
 201  CD 1A 02   CNVRG   CALL REGNR
   4  0A                 LDAX B
   5  32 F4 0B           STA 0BF4
   8  3E 40              MVI A,40
   A  32 F3 0B           STA 0BF3
   D  CD 72 03           CALL DISP2
  10  CD ED 00           CALL DECPT
   3  11 F5 0B           LXI D,0BF5
   6  CD 31 01           CALL DIG4
   9  C9                 RET
-------------------------------------
021A  3A F9 0B   REGNR   LDA 0BF9
   D  E6 0F              ANI 0F
   F  47                 MOV B,A
  20  C6 40              ADI 40
   2  4F                 MOV C,A
   3  78                 MOV A,B
   4  C6 E0              ADI E0
   6  5F                 MOV E,A
   7  06 00              MVI B,00
   9  16 0B              MVI D,0B
   B  C9                 RET
```

```
022C  CD 65 01   CMDK2   CALL CCKEY
   F  D0         END1    RNC
  30  FE 05             CPI 05
   2  CC C5 01          CZ TDAT
   5  FE 03             CPI 03
   7  CC BA 01          CZ ADR2
   A  FE 04             CPI 04
   C  C2 42 02          JNZ CTE
   F  33                INX SP
  40  33                INX SP
   1  D7                RST 2
   2  FE 00    CTE     CPI 00
   4  CC 53 02          CZ TNXT2
   7  CD 76 01 RELC2    CALL CKEY
   A  FE 07             CPI 07
   C  C8       END2     RZ
   D  CD FA 01          CALL REDRG
  50  C3 47 02          JMP RELC2
0253  CD 1A 02 TNXT2    CALL REGNR
   6  3A F8 0B          LDA 0BF8
   9  12                STAX D
   A  21 F9 0B          LXI H,0BF9
   D  34                INR m
   E  7E                MOV A,m
   F  E6 0F             ANI 0F
  61  C0                RNZ
   2  36 08             MVI m,08
   4  C9                RET
0265  21 EA 0B SAVE      LXI H,0BEA
   8  77                MOV m,A
   9  2C                INR L
   A  70                MOV m,B
   B  2C                INR L
   C  71                MOV m,C
   D  2C                INR L
   E  72                MOV m,D
   F  2C                INR L
0270  73                MOV m,E
   1  22 E8 0B          SHLD 0BE8
   4  C9                RET
0275  F5       DSPSW     PUSH PSW
   6  F5                PUSH PSW
   7  11 F3 0B          LXI D,0BF3
   A  3E 77             MVI A,77
   C  12                STAX D
   D  1D                DCR E
   E  3E 40             MVI A,40
0280  12                STAX D
   1  1D                DCR E
   2  1D                DCR E
   3  F1                POP PSW
```

```
0284  CD D5 00              CALL BYTE
  7   E3                    XTHL
  8   7D                    MOV A,L
  9   32 F8 0B              STA 0BF8
  C   E3                    XTHL
  D   11 F4 0B              LXI D,0BF4
 90   CD 31 01    DPLP1     CALL DIG4
  3   CD A2 03              CALL NEXT
  6   D2 8D 02              JNC DPLP1
  9   CD A2 03    DPLP2     CALL NEXT
  C   DA 99 02              JC DPLP1
  F   F1                    POP PSW
 A0   C9                    RET
```

02A1 bis 033F — 158 (dezimal) Bytes frei für Eigenprogramme

```
0340  CD 61 00    NIBIN     CALL HHKEY
  3   D2 51 03              JNC CTE
  6   32 F8 0B              STA 0BF8
  9   11 F1 0B              LXI D,0BF1
  C   CD E2 00              CALL SSGCV
  F   D3 10                 OUT 10
 51   CD A2 03              CALL NEXT
  4   3A F0 0B              LDA 0BF0
  7   00 00                 NOP
  9   C9                    RET
035A  CD 64 03    INPUT     CALL KEYBD
  D   CD A2 03              CALL NEXT
 60   3A FE 0B              LDA 0BFE
  3   C9                    RET
0364  3E 01       KEYBD     MVI A,01
  6   32 FF 0B              STA 0BFF
  9   CD 61 00              CALL HHKEY
  C   D2 81 03              JNC END1
  F   CD 91 00              CALL SHIFT
 72   3A F8 0B    DISP2     LDA 0BF8
  5   11 F1 0B              LXI D,0BF1
  8   CD D5 00              CALL BYTE
  B   CD 75 00    RELS2     CALL HKEY
  E   DA 7B 03              JC 037B
 81   11 F3 0B    END1      LXI D,0BF3
  4   C3 3B 01              JMP 013B
  7   21 F3 0B    RELS3     LXI H,0BF3
  A   36 71                 MVI m,71
  C   2B                    DCX HL
  D   36 40                 MVI m,40
  F   CD 40 03              CALL NIBIN
 92   D2 99 03              JNC
  5   87                    ADD A
  6   C3 54 01              JMP 0154
```

```
0399   11 F4 0B              LXI D,0BF4
   C   CD 36 01              CALL DIG3
   F   C3 87 03              JMP RELS3
-------------------------------------------------
03A2   CD 65 01    NEXT      CALL CCKEY
   5   D0          END1      RNC
   6   FE 00                 CPI 00
   8   C0          END2      RNZ
   9   3A F8 0B              LDA 0BF8
   C   32 F0 0B              STA 0BF0
   F   37                    STC
  B0   C9                    RET
-------------------------------------------------
03B1   C5          ONSEC     PUSH B
   2   0E 0A                 MVI C,0A
   4   CD BD 03   DELYC      CALL DELAY
   7   0D                    DCR C
   8   C2 B4 03              JNZ DELYC
   B   C1                    POP B
   C   C9                    RET
-------------------------------------------------
03BD   06 64       DELAY     MVI B,64
   F   CD C7 03    DELYB     CALL DELY1
  C2   05                    DCR B
   3   C2 BF 03              JNZ DELYB
   6   C9                    RET
03C7   F5          DELY1     PUSH PSW
   8   3E 8B                 MVI A,8B
   A   3D          DELYA     DCR A
   B   C2 CA 03              JNZ DELYA
   E   F1                    POP PSW
   F   C9                    RET
-------------------------------------------------
03D0   00 04       FCTAB     (FN 0)
   2   87 04                 (FN 1)
   4   FF FF                 (FN 2)
   6   FF FF                 (FN 3)
   8   FF FF                 (FN 4)
   A   FF FF                 (FN 5)
   C   FF FF                 (FN 6)
   E   FF FF                 (FN 7)
03E0   A0 01       TROUT     (TNXT)
   2   AC 01                 (TBST)
   4   87 03                 (TFCT)
   6   E9 01                 (TREG)
   8   B4 01                 (TADR)
   A   C5 01                 (TDAT)
   C   D0 01                 (TRUN)
   E   36 00                 (SFTM)
-------------------------------------------------
03F0   3F 06 5B 4F SSGTB     "0", "1", "2", "3"
   4   66 6D 7D 07           "4", "5", "6", "7"
   8   7F 6F 77 7C           "8", "9", "A", "B"
   C   39 5E 79 71           "C", "D", "E", "F"
```

```
0400  CD 24 04   CASOT   CALL EXCHG
   3  3E C0      LEADR   MVI A,C0
   5  30                 SIM
   6  3E 0A              MVI A,0A
   8  CD B1 03   LELOP   CALL ONSEC
   B  3D                 DCR A
   C  C2 08 04           JNZ LELOP
   F  CD 34 04   OUTPT   CALL WORAU
  12  CD 6C 04           CALL CMPAR
   5  C2 0F 04           JNZ OUTPT
   8  CD 78 04           CALL CHSUM
   B  32 FF 0B           STA 0BFF
   E  CD 31 04           CALL WORTO
  21  D7 00 00           RST 2
0424  2A FD 0B   EXCHG   LHLD 0BFD
   7  55                 MOV D,L
   8  5C                 MOV E,H
   9  2A FB 0B           LHLD 0BFB
   C  7C                 MOV A,H
   D  65                 MOV H,L
   E  6F                 MOV L,A
   F  2B                 DCX H
  30  C9                 RET
0431  21 FE 0B   WORTO   LXI H,0BFE
   4  23                 INX H
   5  CD 56 04           CALL STABI
   8  7E                 MOV A,m
   9  06 08              MVI B,08
   B  A7       SAVE     ANA A
   C  F5                 PUSH PSW
   D  F2 46 04           JP NULL
  40  CD 67 04   EINS    CALL STOBI
   3  C3 49 04           JMP CTE1
   6  CD 5A 04   NULL    CALL LOBIT
   9  F1         CTE1    POP PSW
   A  05                 DCR B
   B  CA 52 04           JZ CTE2
   E  07                 RLC
   F  C3 3B 04           JMP SAVE
  52  CD 67 04   CTE2    CALL STOBI
   5  C9                 RET
0456  3E 01      STABI   MVI A,01
   8  D3 04              OUT 04
   A  3E 40      LOBIT   MVI A,40
   C  30         HI      SIM
   D  CD 72 04           CALL DELY2
  60  CD 72 04           CALL DELY2
   3  AF                 XRA A
   4  D3 04              OUT 04
   6  C9                 RET
   7  3E C0      STOBI   MVI A,C0
   9  C3 5C 04           JMP HI
```

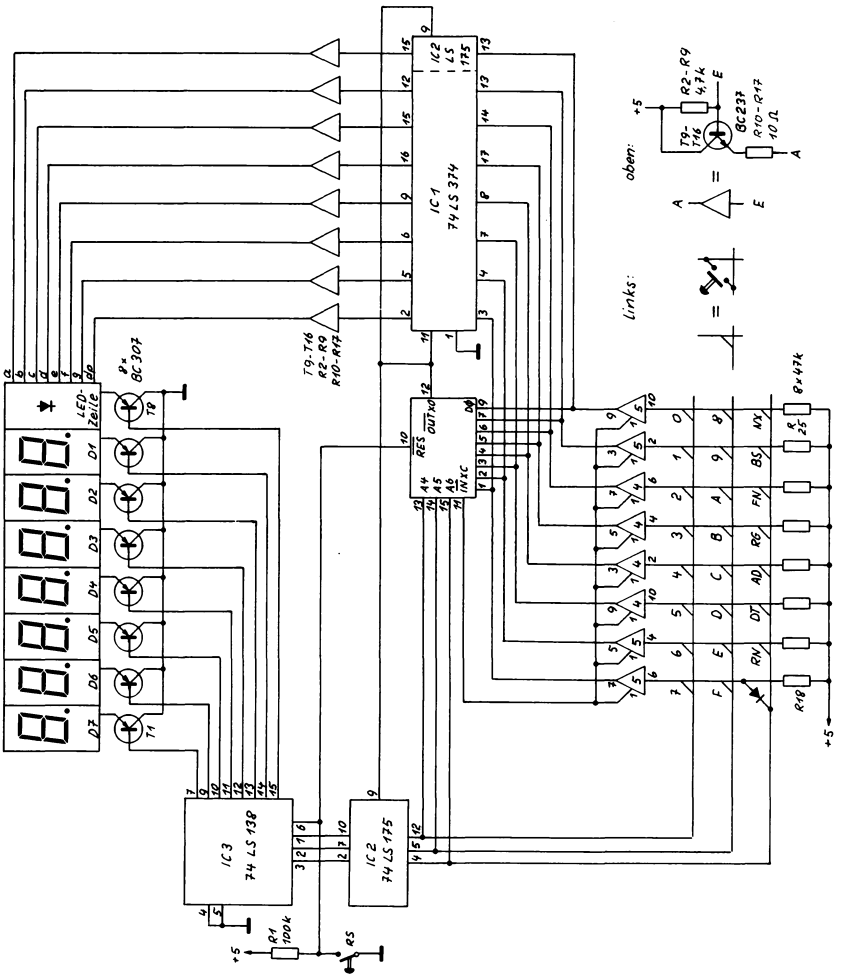
```
046C  7D          CMPAR   MOV A,L
   D  BB                  CMP E
   E  C0                  RNZ
   F  7C                  MOV A,H
  70  BA                  CMP D
   1  C9                  RET
0472  F5          DELY2   PUSH PSW
   3  3E DE               MVI A,DE
   5  C3 CA 03            JMP LOOP
0478  CD 24 04    CHSUM   CALL EXCHG
   B  AF                  XRA A
   C  23          CHLOP   INX H
   D  86                  ADD m
   E  47                  MOV B,A
   F  CD 6C 04            CALL CMPAR
  82  78                  MOV A,B
   3  C2 7C 04            JNZ CHLOP
   6  C9                  RET
-------------------------------------
0487  CD 24 04    CASIN   CALL EXCHG
   A  CD BA 04    INPUT   CALL WORIN
   D  CD 6C 04            CALL CMPAR
  90  C2 8A 04            JNZ INPUT
   3  CD B7 04            CALL WORTI
   6  CD 78 04            CALL CHSUM
   9  21 FF 0B            LXI H,0BFF
   C  BE                  CMP m
   D  C2 A3 04            JNZ ERROR
  A0  D7 00 00            RST 2
04A3  21 F3 0B    ERROR   LXI H,0BF3
   6  36 79               MVI m,79
   8  2B                  DCX H
   9  36 50               MVI m,50
   B  2B                  DCX H
   C  36 50               MVI m,50
   E  11 F4 0B            LXI D,0BF4
  B1  CD 36 01            CALL DIG3
   4  C3 A3 04            JMP ERROR
04B7  21 FE 0B    WORTI   LXI H,0BFE
   A  23          WORIN   INX H
   B  20          SYNC    RIM
   C  A7                  ANA A
   D  F2 BB 04            JP SYNC
  C0  3E 01               MVI A,01
   2  D3 04               OUT 04
   4  20          SYLOP   RIM
   5  A7                  ANA A
   6  FA C4 04            JM SYLOP
   9  CD 72 04            CALL DELY2
   C  AF                  XRA A
   D  D3 04               OUT 04
   F  01 00 08            LXI B,0800
```

```
04D2    CD 72 04    BYTE    CALL DELY2
   5    CD 72 04            CALL DELY2
   8    20                  RIM
   9    E6 80               ANI 80
   B    B1                  ORA C
   C    07                  RLC
   D    4F                  MOV C,A
   E    05                  DCR B
   F    C2 D2 04            JNZ BYTE
  E2    77                  MOV m,A
   3    D3 00               OUT 00
   5    C9                  RET

04E6
bis     frei für Eigenprogramme
07FF
```

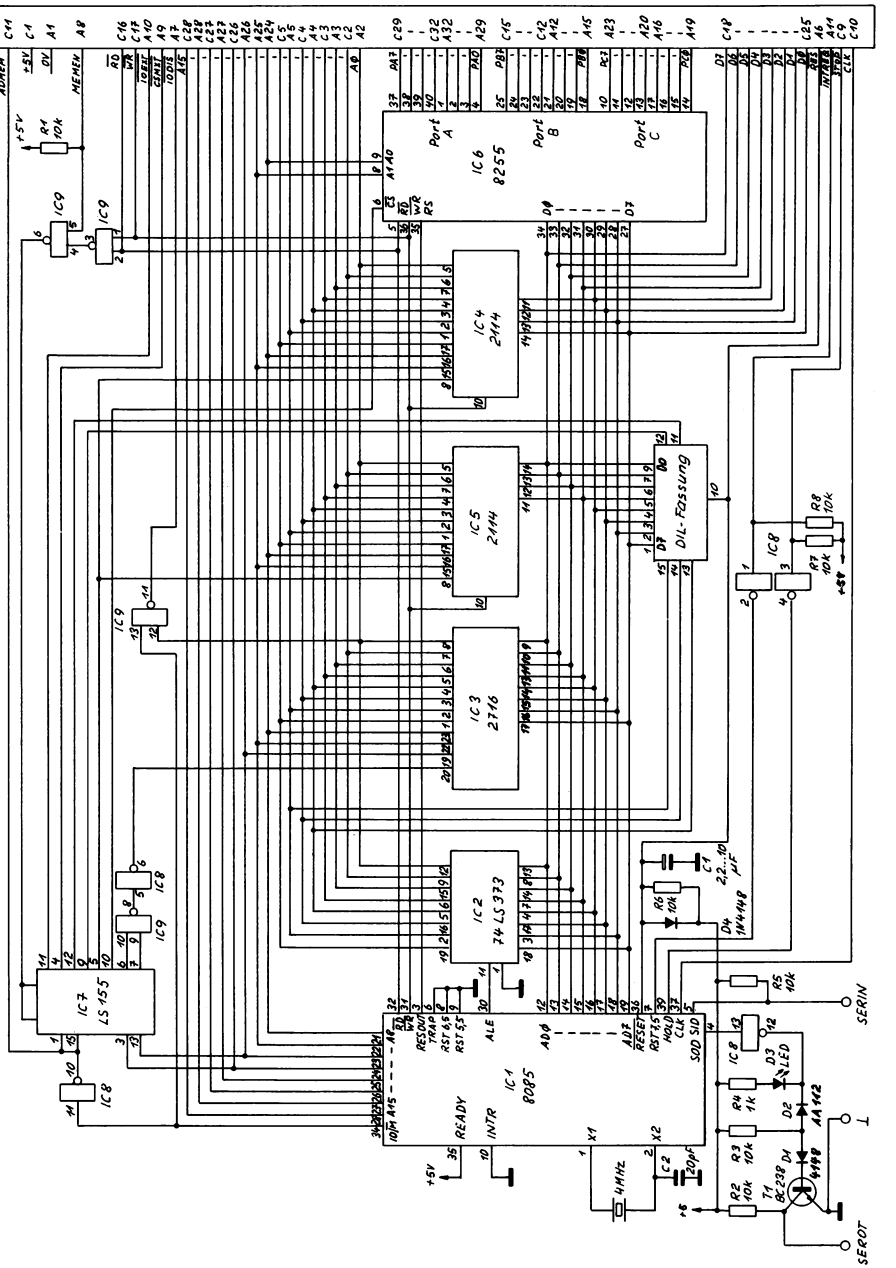
## Monitor-Unterprogramme

Der Monitor enthält zahlreiche Unterprogramme. Besonders diejenigen, die die Eingabe über die Tastatur und die Ausgabe über die Siebensegmentanzeige betreffen, sind auch für den Benutzer interessant. Die wichtigsten sind nachfolgend aufgeführt. Sie können alle mit CALL aufgerufen werden; die Einsprungadresse steht in der ersten Spalte.

| | | |
|---|---|---|
| 035A | INPUT | Byte von der Tastatur einlesen, darstellen und nach NXT in den Akku überschreiben. |
| 0340 | NIBIN | Nibble von der Tastatur einlesen; ansonsten wie INPUT. |
| 03A2 | NEXT | fragt NXT ab; gedrückt: CY=1 und (0BFE) ← (0BF8). |
| 011F | DISPL | überschreibt den Anzeigepuffer 0BF1...0BF7 in die Anzeige (8 ms). |
| 0029 | CLRDS | löscht den Anzeigepuffer 0BF1...0BF7. |
| 00BC | CONVT | Siebensegmentumsetzung: (0BF8) → 0BF1,2 (Datenfeld); (0BF9) → 0BF4,5 und (0BFA) → 0BF6,7 (Adressenfeld). |
| 00D5 | BYTE | Siebensegmentumsetzung des Akkuinhalts; Zieladresse in DE. |
| 00E2 | SSGCV | SSG-Umsetzung des unteren Akku-Nibbles; Zieladresse in DE. |
| 0061 | HHKEY | HEX-Tastatur abfragen; gedrückt: Tasten-Nr. in A und CY = 1. |
| 0165 | CCKEY | CMD-Tastatur abfragen; gedrückt: Tasten-Nr. in A und CY = 1. |
| 0108 | RELSH | Auf Loslassen einer HEX-Taste warten und Anzeige aktivieren. |
| 0193 | RELSC | Auf Loslassen einer CMD-Taste warten und Anzeige aktivieren. |
| 03B1 | ONSEC | Zeitverzögerung 1 s |
| 03BD | DELAY | Zeitverzögerung 100 ms |
| 03C7 | DELY1 | Zeitverzögerung 1 ms |

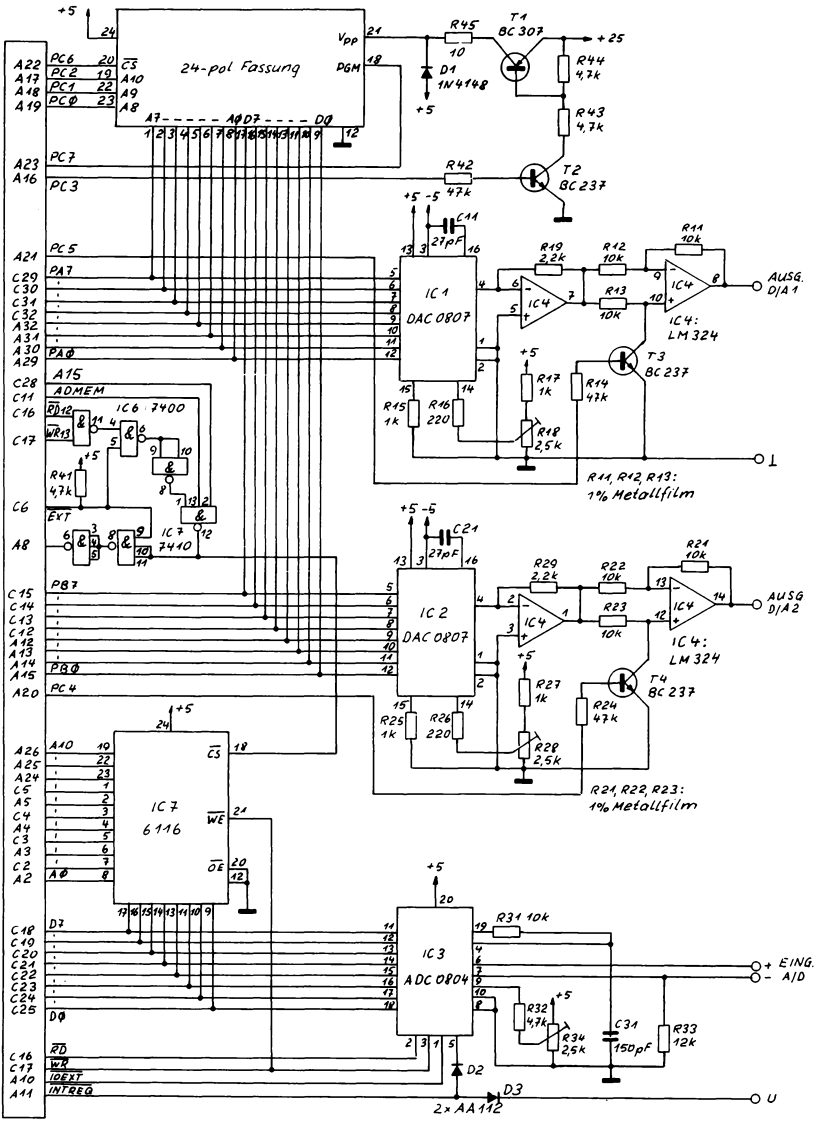



Schaltbild der Tastatur- und Anzeigeplatine

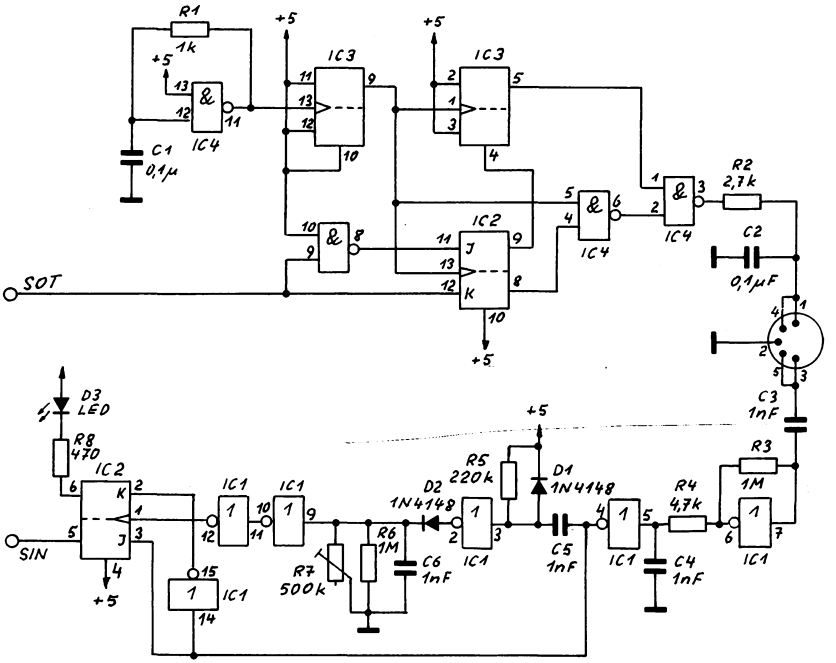



Schaltbild der Rechnerplatine

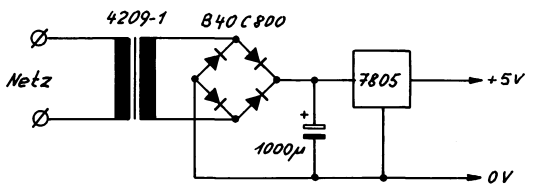
DIPL.-ING. H. WEIDNER, D-3300 BRAUNSCHWEIG, KRÖGERSTRASSE 69



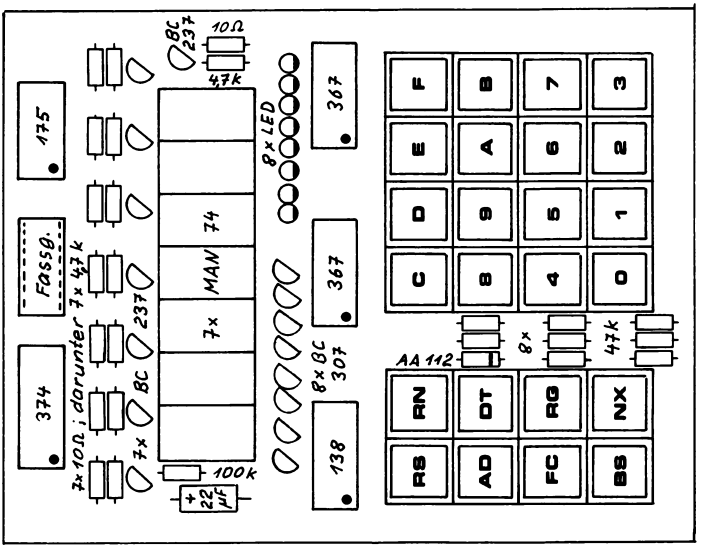

Schaltbild der Erweiterungsplatine



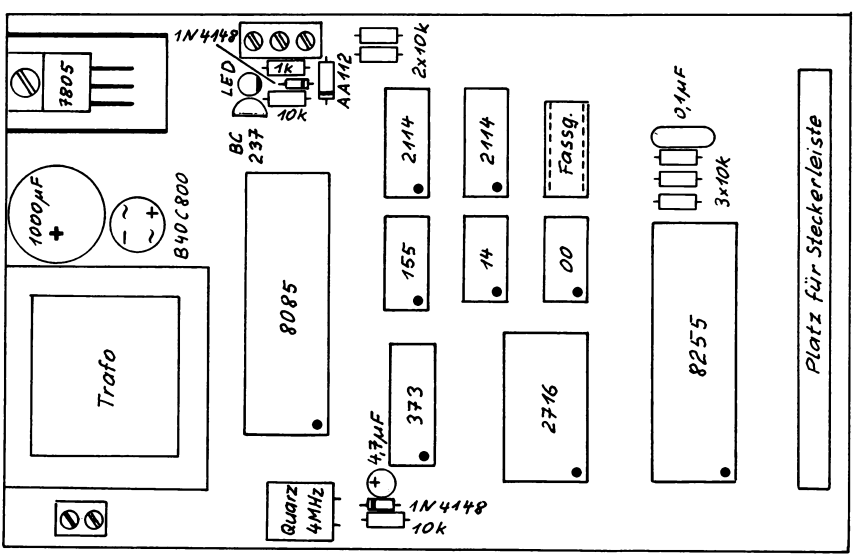

Schaltbild des Kassetten-Interface



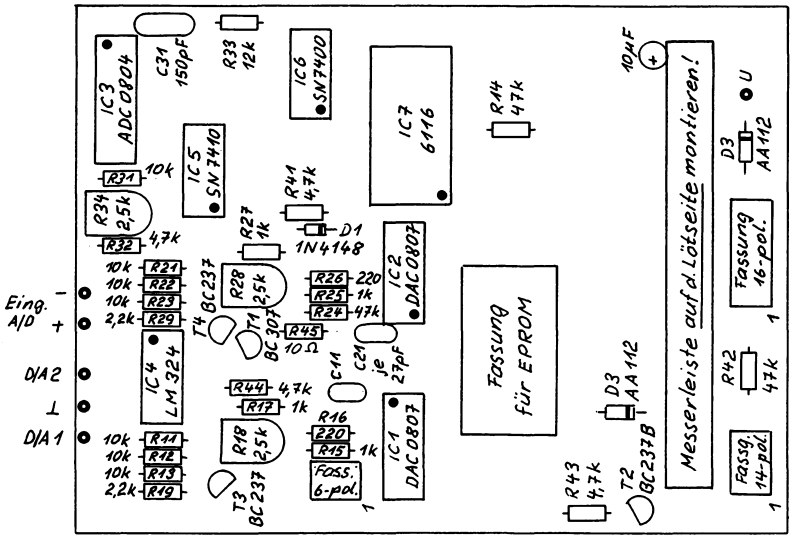

Schaltbild des Netzteils (Normalausführung)



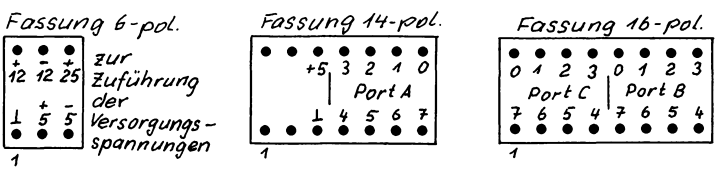

Bestückungsplan der Tastatur- und Anzeigeplatine



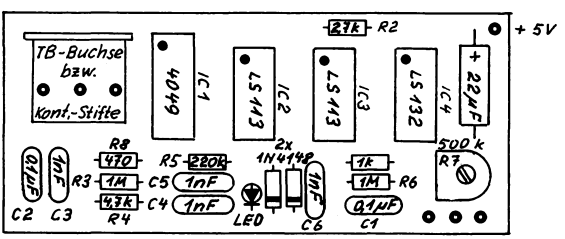

Bestückungsplan der Rechnerplatine, Normalausführung




Bestückungsplan der Erweiterungsplatine




Kontaktbelegung der DIL-Fassungen für Flachkabelstecker




Bestückungsplan der Kassetten-Interface-Platine

| | C | A |
|---|---|---|
| 32 | PA 4 | PA 3 |
| 31 | PA 5 | PA 2 |
| 30 | PA 6 | PA 1 |
| 29 | PA 7 | PA 0 |
| 28 | A 15 | A 14 |
| 27 | A 13 | A 12 |
| 26 | A 11 | A 10 |
| 25 | D 0 | A 9 |
| 24 | D 1 | A 8 |
| 23 | D 2 | PC 7 |
| 22 | D 3 | PC 6 |
| 21 | D 4 | PC 5 |
| 20 | D 5 | PC 4 |
| 19 | D 6 | PC 0 |
| 18 | D 7 | PC 1 |
| 17 | $\overline{WR}$ | PC 2 |
| 16 | $\overline{RD}$ | PC 3 |
| 15 | PB 7 | PB 0 |
| 14 | PB 6 | PB 1 |
| 13 | PB 5 | PB 2 |
| 12 | PB 4 | PB 3 |
| 11 | ADMEM | $\overline{INTREQ}$ |
| 10 | $\overline{CLK}$ | $\overline{IOXT}$ |
| 9 | $\overline{STOP}$ | $\overline{CSMXT}$ |
| 8 | | $\overline{MEMEN}$ |
| 7 | | $\overline{IODIS}$ |
| 6 | | $\overline{RESET}$ |
| 5 | A 7 | A 6 |
| 4 | A 5 | A 4 |
| 3 | A 3 | A 2 |
| 2 | A 1 | A 0 |
| 1 | +5 V | GND |

Belegung der Steckeranschlüsse

## Bedeutung der Steckeranschlüsse

| | | |
|---|---|---|
| A0...A15 | Adreßbus, Adresse 0 bis Adresse 15 | Ausg. |
| D0...D7 | Datenbus, Datenleitung 0 bis Datenleitung 7 | Eing./Ausg. |
| PA0...PA7 | Port A, Leitung 0 bis Leitung 7 | Eing./Ausg. |
| PB0...PB7 | Port B, Leitung 0 bis Leitung 7 | Eing./Ausg. |
| PC0...PC7 | Port C, Leitung 0 bis Leitung 7 | Eing./Ausg. |
| $\overline{WR}$ | Memory write | Ausg. |
| $\overline{RD}$ | Memory read | Ausg. |
| CLK | Takt (clock), 2 MHz | Ausg. |
| $\overline{RESET}$ | System-Reset, entspricht der RS-Taste | Eing. |
| $\overline{STOP}$ | versetzt den Prozessor in den HALT-Zustand. Die Adressen- und Datenleitungen sowie die Leitungen WR, RD und IO/M gehen in den hochohmigen Zustand. Nun ist (z.B. für einen anderen Prozessor) der Zugriff zum Adreß- und Datenbus frei. | Eing. |
| $\overline{INTREQ}$ | Unterbrechungsanforderung (interrupt request). Nach Beendigung eines gerade auszuführenden Befehls wird der Befehlszähler (PC) mit der Adresse 083C geladen. Der Prozessor fährt ab dort mit der Abarbeitung der Befehle fort. | Eing. |
| MEMEN | (memory enable) L-Pegel an diesem Eingang sperrt die Speicher auf der Rechnerplatine. Einer externen Dekodierung wird die Verwaltung eines separaten Speichers ermöglicht (wie z.B. auf der Erweiterungskarte). | Eing. |
| $\overline{CSMXT}$ | (chip select memory external) Ein Aktivierungssignal für den Speicherbereich 0C00 bis 0FFF. Einfache Erweiterungsmöglichkeit um 1 K. | Ausg. |
| ADMEM | (memory address) Dies ist das invertierte IO/$\overline{M}$-Signal. Ist es auf H, dann wird eine Adresse im Speicherbereich angesprochen; ist es auf L, dann wird eine Port-Adresse angesprochen. | Ausg. |
| $\overline{IODIS}$ | (IO disable) geht auf LOW, wenn die IO/$\overline{M}$-Leitung und die Adreßleitung A0 auf H sind. | Ausg. |
| $\overline{IOXT}$ | (IO external) Ein Aktivierungssignal für externe Peripherie, die als Port angesprochen werden soll. Mit diesem Signal wird z.B. der A/D-Wandler auf der Erweiterungskarte aktiviert. | Ausg. |

Der 8255 ist ein sehr vielseitiger Schnittstellen-Baustein für den Datenverkehr zwischen Mikroprozessorsystem und Umwelt. Er hat 3 E/A-Ports zu je 8 Bit, die in ihrer Funktion auf verschiedene Weise programmiert werden können. Welche Funktion ausgeführt werden soll, wird durch ein Steuerwort bestimmt. Als Ausgang programmiert, kann jeder Port-Anschluß 1 mA bei 1,5 V liefern.

Für den Datenverkehr von und zu einem Port bzw. Steuerwort dienen die Befehle IN und OUT. Sie bewirken, daß die System-Steuerleitung IO/M auf H-Pegel geht. Damit werden alle Speicherbausteine abgeschaltet, alle Port-Bausteine hingegen aktiviert. Die dem IN- bzw. OUT-Befehl folgende Adresse (8 Bit) erscheint zweimal auf dem Adreßbus, einmal auf den unteren 8 Bits und einmal auf den oberen 8 Bits. Direkt, also ohne Dekodierung, können zwei Bausteine 8255 an ein 8085-System angeschlossen werden. Mit vollständiger Dekodierung können bis zu 64 Bausteine angeschlossen werden.

In dem vorliegenden System gelten folgende Adressen:

| | |
|---|---|
| Port A | 04 |
| Port B | 05 |
| Port C | 06 |
| Steuerwort | 07 |

Der 8255 kennt drei Betriebsarten, die vom Programm bestimmt werden können:

| | |
|---|---|
| Betriebsart 0 | Standard-Ein/Ausgabe |
| Betriebsart 1 | getaktete Ein/Ausgabe |
| Betriebsart 2 | bidirektionaler Bus |

Wenn der Reset-Eingang auf H geht, werden alle Ports als Eingänge definiert und sind hochohmig. In diesem Zustand verbleibt der Baustein, auch wenn Reset wieder auf L geht. Während der Ausführung eines Programms kann jede mögliche Betriebsart gewählt werden, indem über einen OUT-Befehl das Steuerwort entsprechend geladen wird.

Die Betriebsarten für Port A und Port B können getrennt definiert werden, während Port C in zwei Hälften aufgeteilt wird entsprechend der Betriebsartbestimmung von Port A und B. Alle Ausgaberegister einschließlich der Zustands-Flipflops werden bei einem Wechsel der Betriebsart gelöscht. Die Betriebsarten können kombiniert werden, so daß eine allen E/A-Anforderungen genügende Konfiguration geschaffen werden kann.

Die nachfolgende Beschreibung der Programmierung des 8255 ist wie folgt unterteilt:

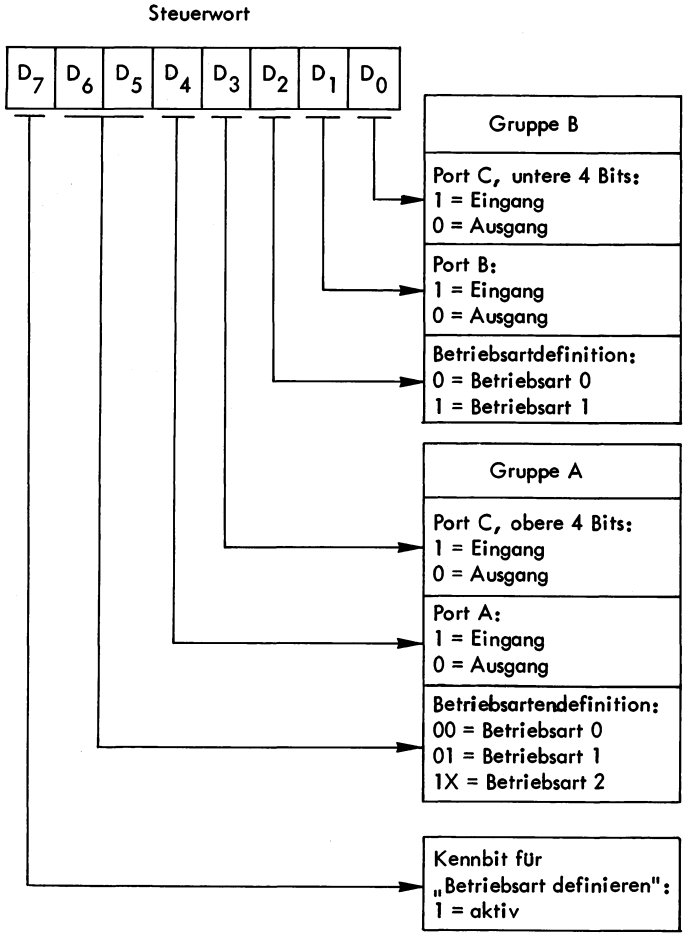
Definition der Betriebsart
Setzen und Löschen von Einzelbits im Port C
Beschreibung der Betriebsart
- Betriebsart 0
- Betriebsart 1
- Betriebsart 2

Die verwendeten Symbole und Bezeichnungen sind gebräuchlich. Für die Beschreibung des Zustandes einer logischen Variablen wird verwendet

HIGH (abgekürzt H) oder 1 für ein besetztes Bit
LOW (abgekürzt L) oder 0 für ein nicht besetztes Bit.

## Definition der Betriebsart

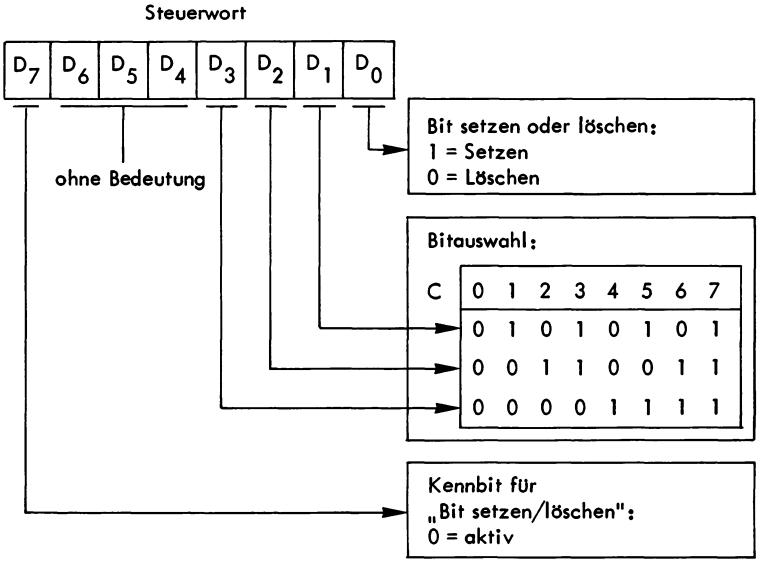
Das nachfolgend dargestellte Schema zeigt in übersichtlicher Form, wie das Steuerwort aussehen muß, damit die gewünschte Betriebsart definiert wird.




## Setzen und Löschen von Einzelbits im Port C

Jedes der acht Bits im Port C kann durch Anwendung eines OUT-Befehls gesetzt oder gelöscht werden. Das Schema auf der nächsten Seite zeigt übersichtlich, wie das Steuerwort aussehen muß, um ein bestimmtes Bit zu setzen oder zu löschen. Man beachte die Bedeutung von Bit 7 im Steuerwort:

Bit 7 = 1: Betriebsart definieren
Bit 7 = 0: Einzelbits setzen/löschen




Wenn der 8255 so programmiert wird, daß er in Betriebsart 1 oder 2 arbeitet, sind Steuersignale erforderlich, die als Interrupt-Anforderungseingabe für die CPU benutzt werden können. Die Unterbrechungs-Anforderungssignale, die von Port C generiert werden, können verriegelt oder freigegeben werden, indem das zugehörige INTE-Flipflop gesetzt oder gelöscht wird. Diese Eigenschaft gestattet es dem Programmierer, die Interruptmöglichkeit für ein bestimmtes E/A-Teil zu verhindern oder freizugeben, ohne ein anderes Teil in der Interrupt-Struktur zu beeinflussen.

Definition des INTE-Flipflops:

Bit gesetzt: INTE ist gesetzt - Interrupt ist möglich
Bit gelöscht: INTE ist gelöscht - Interrupt ist gesperrt

(INTE-Flipflop: Siehe Beschreibung der Betriebsarten 1 und 2.)

**Betriebsart 0 (Standard-Ein/Ausgabe)**

Diese Betriebsart ist durch folgende grundlegenden Eigenschaften gekennzeichnet:

Zwei 8-Bit-Ports und zwei 4-Bit-Ports.
Jeder Port kann Eingang oder Ausgang sein.
Ausgänge haben einen Auffangspeicher.
Eingänge haben keinen Auffangspeicher.
16 verschiedene E/A-Konfigurationen sind möglich.

Die Tabelle auf der nächsten Seite zeigt die möglichen Konfigurationen abhängig von den Bits $D_0$, $D_1$, $D_3$ und $D_4$ im Steuerwort. Bits $D_2$, $D_5$ und $D_6$ sind immer 0; Bit $D_7$ ist immer 1.

| | | | | Gruppe A | | Gruppe B | |
|---|---|---|---|---|---|---|---|
| $D_4$ | $D_3$ | $D_1$ | $D_0$ | Port A | Port C Bit 4 - 7 | Port B | Port C Bit 0 - 3 |
| 0 | 0 | 0 | 0 | Ausgang | Ausgang | Ausgang | Ausgang |
| 0 | 0 | 0 | 1 | Ausgang | Ausgang | Ausgang | Eingang |
| 0 | 0 | 1 | 0 | Ausgang | Ausgang | Eingang | Ausgang |
| 0 | 0 | 1 | 1 | Ausgang | Ausgang | Eingang | Eingang |
| 0 | 1 | 0 | 0 | Ausgang | Eingang | Ausgang | Ausgang |
| 0 | 1 | 0 | 1 | Ausgang | Eingang | Ausgang | Eingang |
| 0 | 1 | 1 | 0 | Ausgang | Eingang | Eingang | Ausgang |
| 0 | 1 | 1 | 1 | Ausgang | Eingang | Eingang | Eingang |
| 1 | 0 | 0 | 0 | Eingang | Ausgang | Ausgang | Ausgang |
| 1 | 0 | 0 | 1 | Eingang | Ausgang | Ausgang | Eingang |
| 1 | 0 | 1 | 0 | Eingang | Ausgang | Eingang | Ausgang |
| 1 | 0 | 1 | 1 | Eingang | Ausgang | Eingang | Eingang |
| 1 | 1 | 0 | 0 | Eingang | Eingang | Ausgang | Ausgang |
| 1 | 1 | 0 | 1 | Eingang | Eingang | Ausgang | Eingang |
| 1 | 1 | 1 | 0 | Eingang | Eingang | Eingang | Ausgang |
| 1 | 1 | 1 | 1 | Eingang | Eingang | Eingang | Eingang |

**Betriebsart 1 (getaktete Ein/Ausgabe)**

Diese Betriebsart schafft die Möglichkeit, Daten in Verbindung mit Takten oder „Handshake"-Signalen von oder zu einem Port zu transportieren. In Betriebsart 1 benutzen Port A und Port B die Anschlüsse von Port C, um diese Handshake-Signale zu erzeugen oder zu erkennen. Diese Betriebsart ist durch folgende grundlegenden Eigenschaften gekennzeichnet:

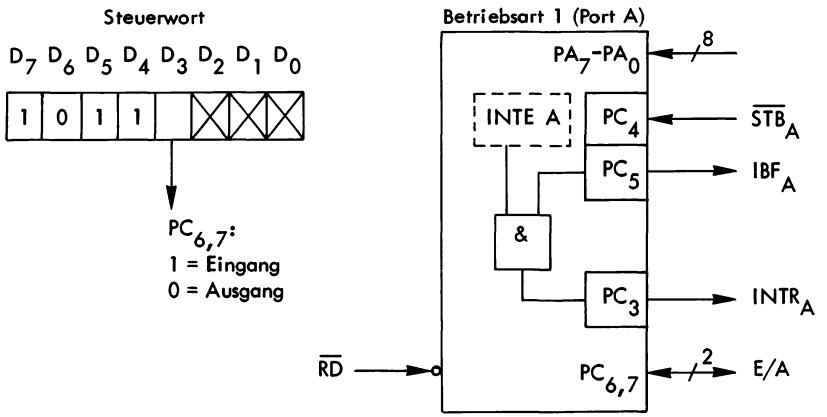
Zwei Gruppen (Gruppe A und Gruppe B)
Jede Gruppe umfaßt einen 8-Bit-Datenport und einen 4-Bit-Steuer/Datenport.
Der 8-Bit-Datenport kann entweder Eingang oder Ausgang sein.
In beiden Fällen gibt es ein Auffangregister.
Der 4-Bit-Port wird zur Steuerung und Zustandsanzeige des 8-Bit-Datenports benutzt.

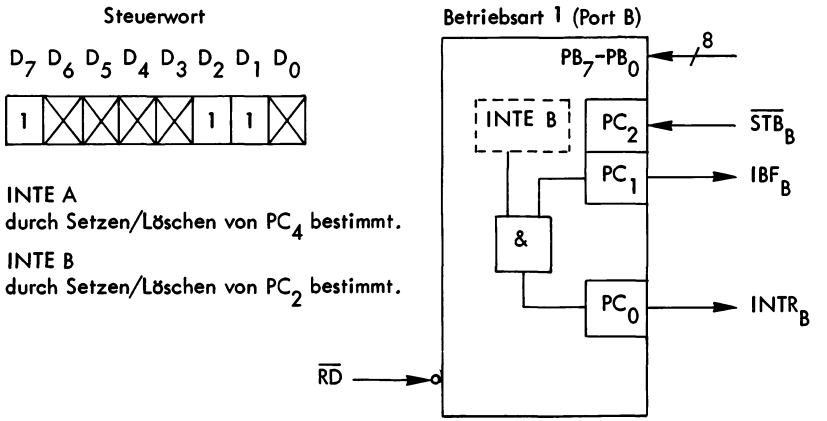
Definition der Eingangs-Steuersignale

$\overline{\text{STB}}$ (Strobe Input) L-Pegel an diesem Eingang lädt die Daten in den Eingangs-Auffangspeicher.

IBF (Input Buffer Full) H-Pegel an diesem Ausgang zeigt an, daß die Daten in den Eingangs-Auffangspeicher geladen wurden; es ist also ein Quittiersignal. IBF wird von der negativen Flanke des STB-Signals gesetzt und von der positiven Flanke des RD-Signals gelöscht.

INTR (Interrupt Request) Mit H-Pegel an diesem Ausgang kann die CPU unterbrochen werden, wenn eine Eingabeeinheit Bedienung verlangt. INTR wird von der positiven Flanke von STB gesetzt, wenn IBF und INTE den Wert H haben, und von der negativen Flanke von RD gelöscht. Auf diese Weise ist es einer Eingabeeinhait möglich, von der CPU Bedienung zu verlangen, indem sie einfach die Daten taktweise in den Port schreibt.




INTE A
durch Setzen/Löschen von $PC_4$ bestimmt.

INTE B
durch Setzen/Löschen von $PC_2$ bestimmt.

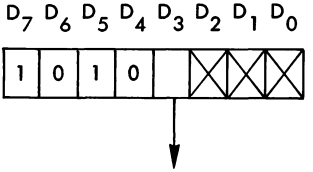
Definition der Ausgangs-Steuersignale

$\overline{OBF}$ (Output Buffer Full) Wenn der $\overline{OBF}$-Ausgang auf L geht, zeigt er damit an, daß die CPU Daten in den angegebenen Ausgangs-Port geschrieben hat. Das OBF-Flipflop wird von der positiven Flanke des WR-Eingangs gesetzt und von der negativen Flanke des $\overline{ACK}$-Signals gelöscht.

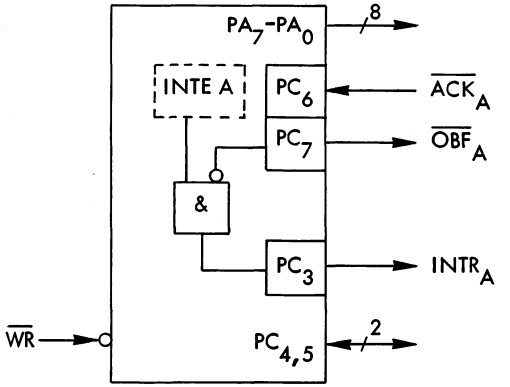
$\overline{ACK}$ (Acknowledge Input) L-Pegel an diesem Eingang zeigt dem 8255 an, daß die Daten von Port A oder B übernommen worden sind. Dies ist im Grunde eine Rückmeldung der Peripherieeinheit, daß die Daten, die von der CPU ausgegeben wurden, angekommen sind.

INTR (Interrupt Request) H-Pegel an diesem Ausgang kann dazu benutzt werden, um die CPU zu unterbrechen, wenn eine Ausgabeeinheit die von der CPU übermittelten Daten übernommen hat. INTR wird von der positiven Flanke von $\overline{ACK}$ gesetzt, wenn $\overline{OBF}$ und INTE auf H sind, und von der negativen Flanke von $\overline{WR}$ gelöscht.

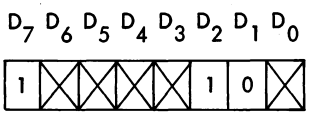
Steuerwort




Betriebsart 1 (Port A)



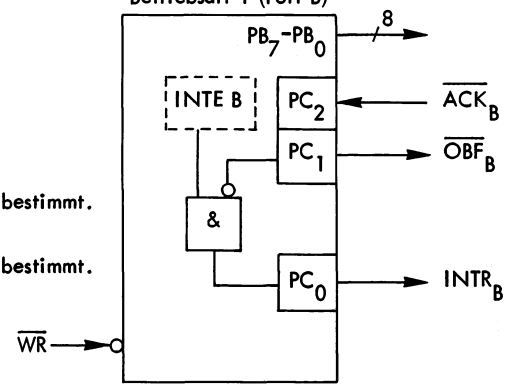

Steuerwort




INTE A
durch Setzen/Löschen von $PC_6$ bestimmt.

INTE B
durch Setzen/Löschen von $PC_2$ bestimmt.

Betriebsart 1 (Port B)




**Betriebsart 2 (getaktete bidirektionale Bus-Ein/Ausgabe)**

In dieser Betriebsart hat man die Möglichkeit, mit einer Peripherieeinheit über einen einzigen 8-Bit-Bus Daten auszutauschen. Ähnlich wie bei Betriebsart 1 stehen „Handshake"-Signale zur Verfügung, um den Datenfluß geordnet aufrecht zu erhalten. Interrupterzeugung sowie Sperr- und Freigabefunktionen stehen ebenfalls zur Verfügung. Die Betriebsart 2 ist durch folgende grundlegenden Eigenschaften gekennzeichnet:

Nur bei Gruppe A anzuwenden.
Ein 8-Bit bidirektionaler Bus-Port (Port A) und ein 5-Bit Steuerport (Port C).
Eingänge und Ausgänge haben Auffangregister.
Der 5-Bit Steuerport wird für die Steuerung und Zustandsanzeige von Port A benutzt.

Definition der Steuersignale für den bidirektionalen E/A-Bus

INTR (Interrupt Request) H-Pegel an diesem Ausgang kann dazu benutzt werden, um die CPU zum Zweck der Datenein - oder Datenausgabe zu unterbrechen.

Ausgabeoperationen:

$\overline{OBF}$ (Output Buffer Full) Wenn der $\overline{OBF}$-Ausgang auf L geht, zeigt er damit an, daß die CPU Daten in den Port A geschrieben hat.

$\overline{ACK}$ (Acknowledge) L-Pegel an diesem Eingang ermöglicht es dem tri-state Ausgangspuffer von Port A, die Daten zu senden; andernfalls ist der Ausgangspuffer im hochohmigen Zustand.

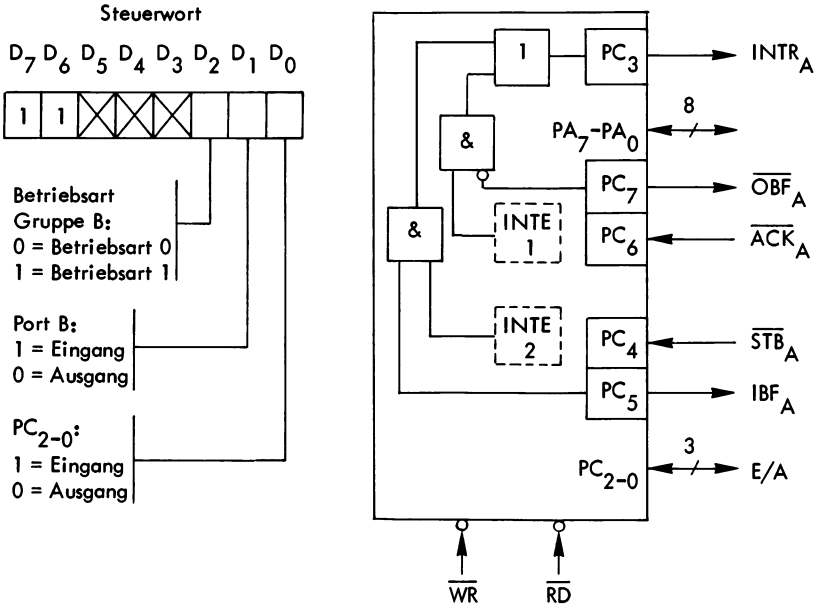
INTE 1 (das zu OBF gehörige INTE-Flipflop) wird durch Setzen oder Löschen von $PC_6$ gesteuert.

Eingabeoperationen:

$\overline{STB}$ (Strobe Input) L-Pegel an diesem Eingang lädt die Daten in den Eingangs-Zwischenspeicher.

IBF (Input Buffer Full) H-Pegel an diesem Ausgang zeigt an, daß Daten in den Ausgangs-Zwischenspeicher geladen wurden.

INTE 2 (das zu IBF gehörige INTE-Flipflop) wird durch Setzen oder Löschen von $PC_4$ gesteuert.




Es gibt einige Betriebsarten-Kombinationen, bei denen nicht alle Bits von Port C für Steuer- oder Anzeigezwecke benutzt werden. Die übrigbleibenden Bits können wie folgt benutzt werden:

Wenn sie als Eingänge programmiert sind:

Alle Eingangsleitungen sind über eine normale Port C-Leseoperation zugänglich.

Wenn sie als Ausgänge programmiert sind:

Bits $PC_7$ bis $PC_4$ müssen einzeln durch eine Setz- oder Löschoperation angesprochen werden.

Bits $PC_3$ bis $PC_0$ können über die Einzelbit-Operation angesprochen werden oder gemeinsam über einen Port C-Schreibbefehl.

Lesen des Zustandes von Port C

In Betriebsart 0 transportiert Port C Daten von oder zu einer Peripherieeinheit. Wenn der 8255 in Betriebsart 1 oder 2 arbeitet, erzeugt oder empfängt Port C „handshake"-Signale zu oder von der Peripherieeinheit. Das Auslesen von Port C ermöglicht es dem Programmierer, den augenblicklichen Zustand der Peripherieeinheit zu prüfen oder zu bestätigen.

In den folgenden Schemen ist Port C als Zustandswort dargestellt.