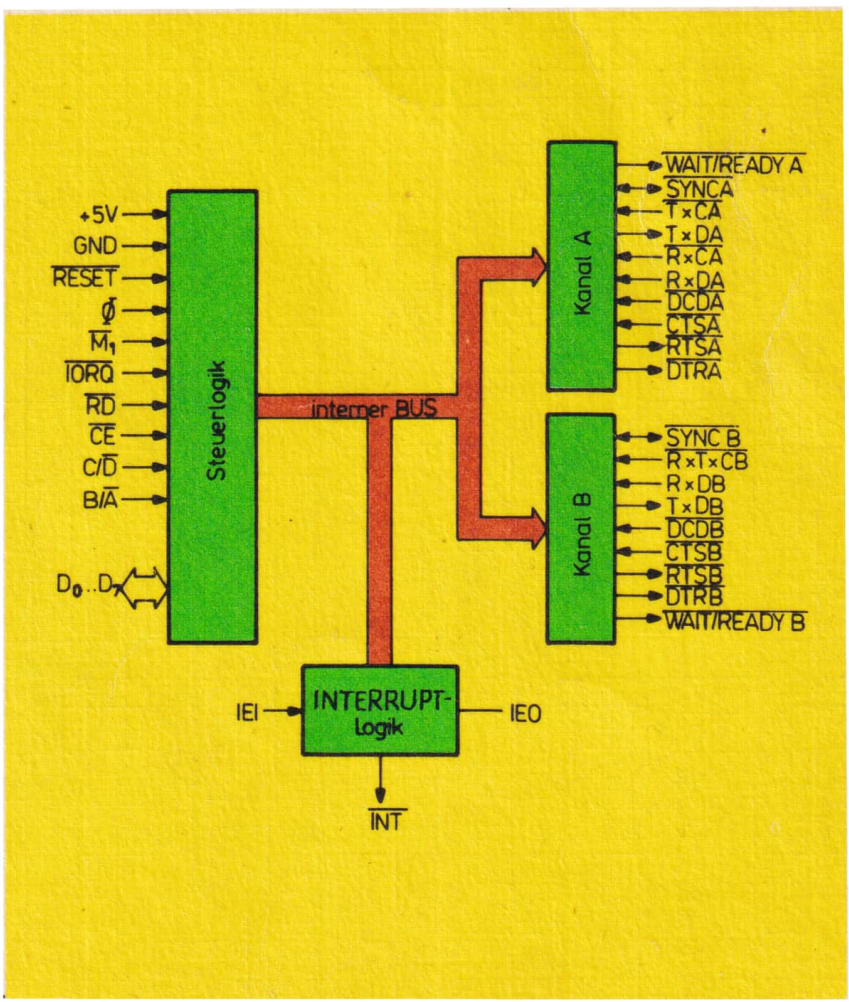
8-Bit-Mikrorechen-technik

Bäurich/
Barthold



Heinz Bäurich/Hans Barthold

8-Bit-Mikrorechentechnik

Prozessoren, Schaltkreise und ihre Programmierung

Militärverlag
der Deutschen Demokratischen
Republik

Die Autoren danken den Herren Jürgen Hamann und Rainer Erlekampf, die durch fachliche Beratung und konstruktive Hinweise die Vorbereitung dieser Veröffentlichung unterstützten.

Bäurich, H./Barthold, H.:
8-Bit-Mikrorechentechnik; Prozessoren, Schaltkreise und ihre Programmierung. – Berlin: Militärverlag der Deutschen Demokratischen Republik, 1988. – 160 S.: 138 Bilder

ISBN 3-327-00668-7

1. Auflage 1988

Lizenz-Nr. 5
Printed in the German Democratic Republic
Lichtsatz: INTERDRUCK Graphischer Großbetrieb Leipzig – III/18/97
Druck und buchbinderische Weiterverarbeitung:
Druckerei des Ministeriums für Nationale Verteidigung – (VEB) Berlin – 30360-8
Lektor: Rainer Erlekampf
Zeichnungen: Angelika Ulsamer
Typografie: Catrin Kliche
Umschlaggestaltung: Rosemarie Lebek
Redaktionsschluß: 2. März 1988
LSV 3539
Bestellnummer: 747 116 8
00900

Inhalt

1. Einleitung

Die Computertechnik, auf der Basis der 8-Bit-Mikroprozessoren, hat einen großen Anteil an der gesamten Rechentechnik erlangt. Es gibt Anwendungen für die unterschiedlichsten Gebiete. Ebenso groß ist das Spektrum der Geräte mit 8-Bit-Mikroprozessoren. Angefangen von Mikroprozessorbausätzen, Klein- und Lerncomputern über Büro- und Personalcomputer zu modularen Systemen für den Einsatz in der Prozeßautomatisierung, Experimentautomatisierung und Gerätesteuerung. Ähnlich umfangreich ist die dazu entstandene Software.
Die Anwendung solcher Computer geht von der einfachen Nutzung im Büro bis zu speziellen Einsatzfällen, in denen der Computer in Hard- und Software präpariert werden muß.
Für denjenigen, der seinen Computer etwas genauer kennenlernen möchte oder gar spezielle Funktionen verändern will, bietet diese Veröffentlichung eine Zusammenfassung der Grundlagen der 8-Bit-Mikrorechnersysteme.
Die Grundlagen der Rechentechnik konnten aus Umfangsgründen nicht behandelt werden. Interessierte Leser finden sie jedoch in dem 1988 erscheinenden Titel »Einführung in die 16-Bit-Mikrorechentechnik mit dem *K 1810 WM86*«.
International sind eine Reihe von Mikroprozessoren mit einer Datenbreite von 8 Bit entstanden. Begonnen hat diese Entwicklung mit dem Prozessor *i8008* von *Intel*. Die wichtigsten Prozessoren, die danach folgten, sind:

U808	DDR
8080	Intel
K580 JK80A	UdSSR
6800	Motorola
6808	Motorola
Z80	Zilog
U880	DDR

Die in der DDR am meisten vertretenen Prozessoren sind der *U808, U880* und der *8080*, wovon in dieser Veröffentlichung die Bausteine *U880* und *8080* beschrieben werden.

2. Der Mikroprozessor U 880

2.1. Registerstruktur

Der *U880* enthält einen internen 8-Bit-Bus, von dem aus alle Register erreichbar sind. Dieser Bus ermöglicht die Ein- und Ausgabe von Daten über den externen Datenbus D_0 bis D_7, der in beiden Richtungen betrieben werden kann.
Der externe Datenbus ist über Bustreiber mit dem internen 8-Bit-Bus verbunden. An den internen 8-Bit-Bus sind alle internen Register angeschlossen (Bild 2.1).

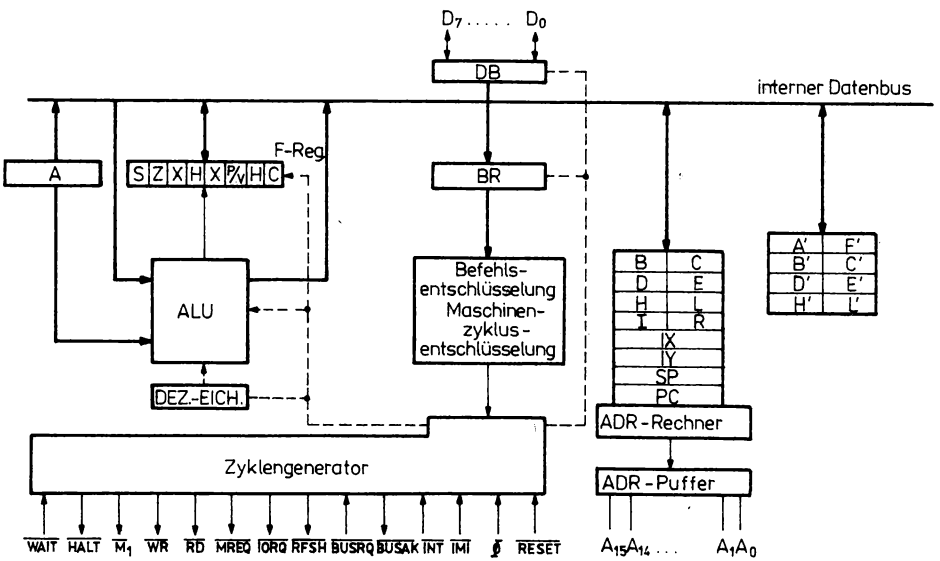


Bild 2.1
Registerstruktur des *U880*

An den internen 8-Bit-Bus sind angeschlossen:
- 2 Registersätze zur Zwischenspeicherung der Zahlen im Prozessor, die aus je 8 8-Bit-Registern bestehen. Der 1. Registersatz beinhaltet die Register A, F[1)], B, C, D, E, H, L, der 2. Registersatz die Register A', F'[1)], B', C', D', E', H', L'. Durch einen einfachen Austauschbefehl können die Inhalte der Registersätze komplett vertauscht werden. Dadurch ist es möglich, einen bestimmten Programmabschnitt einem der Registersätze zuzuordnen. Wechselt das Programm, dann können die dazugehörigen Registersätze umgetauscht werden.

[1)] F, F' sind Flagregister.

- Zweckregister I, R, IX, IY, SP, PC.
  Das Register I (8-Bit-INTERRUPT-Adreßregister) enthält den höherwertigen Teil einer Adresse, deren niederwertiger Teil bei einem INTERRUPT von dem entsprechenden Gerät gebildet wird. Die Adresse weist auf eine Speicherzelle, in der die Startadresse des INTERRUPT-Bedienprogramms steht.
  Das Register R (Speicherauffrischregister) enthält eine 7-Bit-Adresse, die in Verbindung mit dem Auffrischsignal RFSH auf den niederwertigen 7 Bit des Adreßbus ausgesendet wird. Diese Adresse wird während der Operationscodeentschlüsselung ausgesendet. Sie dient zum Auffrischen von dynamischen Speichern. Während jedes Operationscodeholzyklus erhöht sich der Inhalt des Registers R um 1.
  Die beiden Register IX und IY (Indexregister) können eine 16-Bit-Basisadresse enthalten. Während der Adressenrechnung wird aus dieser Basisadresse durch Addition einer Adreßzahl die eigentliche Operandenadresse ermittelt.
  Das Register SP (Stackpointer) enthält eine 16-Bit-Adresse, die die Speicherzelle an der Spitze eines Kellerspeichers adressiert. Der Kellerspeicher ist als »last in – first out-Speicher« organisiert (das zuletzt eingeschriebene Wort wird zuerst gelesen).
  Das Register PC (Programm-Counter oder Befehlszähler) enthält eine 16-Bit-Adresse, die angibt, aus welcher Speicherzelle der laufende Befehl geholt wird.
- In dem Befehlsregister BR wird der Operationscode des laufenden Befehls gespeichert. Hier kommt es zur Decodierung des Befehls und zur Bildung der Steuersignale für dessen Abarbeitung. Die Steuersignale bestehen aus den Befehls- und den Zeitsignalen. Die Befehlssignale werden durch die Befehlsentschlüsselung und die dazugehörigen Zeitsignale durch die Zeitsteuerung gebildet. Die Zeitsteuerung besteht aus dem Zyklengenerator, der durch den externen Takt Φ und die Befehlssignale gesteuert wird. Im Zyklengenerator werden auch die Signale zur Steuerung der externen Bausteine gebildet sowie die von den externen Bausteinen kommenden Signale abgetastet.
- Das Flagregister enthält 6 Flip-Flop, die in Abhängigkeit von den einzelnen Befehlen und vom Ergebnis der Befehle gesetzt oder rückgesetzt werden. Die einzelnen Flags haben folgende Bedeutung:
  C: Carry-Flag
  C ist gleich 1, wenn bei der Addition ein Übertrag in die 8. Stelle auftritt oder wenn bei der Subtraktion ein Borgen von der 8. Stelle notwendig wird.
  N: Subtraktions-Flag
  N ist gleich 1, wenn die ausgeführte Operation eine Subtraktion war.
  P/V: Parity-Überlauf-Flag (Überlauf = Overflow)
  P/V ist gleich 1, bei logischen Operationen, wenn die Anzahl der Einsen im Ergebnis geradzahlig ist, bei Rechenoperationen, wenn ein Überlauf auftritt (Ergebnis größer als die größte darstellbare Zahl).[2)]
  H: Half-Carry-Flag
  H ist gleich 1, wenn es bei der Addition zu einem Übertrag in die 4. Stelle kommt oder wenn bei der Subtraktion ein Borgen von der 4. Stelle notwendig wird.
  Z: Zero-Flag
  Z ist gleich 1, wenn im Ergebnis alle Bitstellen 0 sind.

[2)] Der Prozessor *U880* arbeitet mit einem 8-Bit-Zahlwort im Zweierkomplement. Der Stellenwert 2^7 entspricht dem Vorzeichen. Die größte positive Zahl ist $2^7 - 1$, die negative Zahl mit dem größten Betrag -2^7. Der vom Prozessor erfaßte Zahlenbereich umfaßt $-2^7 \leqq Z \leqq 2^7 - 1$.

S: Sign-Flag
S ist gleich 1, wenn im Ergebnis das Vorzeichen 1 (negativ) ist.

Beispiele zur Flagbelegung

```
                    1  ↖                 Übertrag in 4. Stelle
  120 =   0 1 1 1       1 0 0 0          ergibt 1 → H
+ 121 =   0 1 1 1       1 0 0 1
-------------------------------------
  241 = 0] 1 1 1 1       0 0 0 1
        ↙   ↘
0 → C      1 → P/V      wegen Überlauf
                                          kein Übertrag in 4. Stelle
                                          ergibt 0 → H
  -  5 =   1 1 1 1       1 0 1 1
  - 16 =   1 1 1 1       0 0 0 0
-------------------------------------
  - 21 = 1] 1 1 1 0      1 0 1 1
        ↙
1 → C      aber 0 → P/V
```

2.2. Befehlsaufbau

2.2.1. Befehlsstruktur

Ein Befehl besteht aus Operationsteil und Adreßteil. Zur Darstellung eines Befehls werden 1 bis 4 Byte benötigt. Davon kann der Operationscode 1 bis 3 Byte[3)] und der Adreßteil 1 bis 2 Byte lang sein. Bei den meisten Befehlen ist der Operationscode 1 Byte lang. Bei 2 Byte langen Operationscodes gibt das 1. Byte die Befehlsgruppe an. Durch das 2. Byte und 3. Byte werden spezielle Befehle innerhalb der Gruppe gekennzeichnet. Bild 2.2 zeigt die im Prozessor *U880* möglichen Befehlsstrukturen.

2.2.2. Operanden

Zu einem Befehl gehören 1 oder 2 Operanden. Diese Operanden können in Registern, im Speicher oder im Befehl stehen (Direktoperand). Für im Speicher stehende Operanden gibt es eine Adresse, die über die Adreßrechnung ermittelt wird.
Beim *U880* gibt es folgende Möglichkeiten für die Operanden:

Registeroperand
Der Operand steht in einem im Befehl angegebenen Register.

[3)] Bei einigen Befehlen mit Indexrechnung ist der Operationscode 3 Byte und der Adreßteil 1 Byte lang.

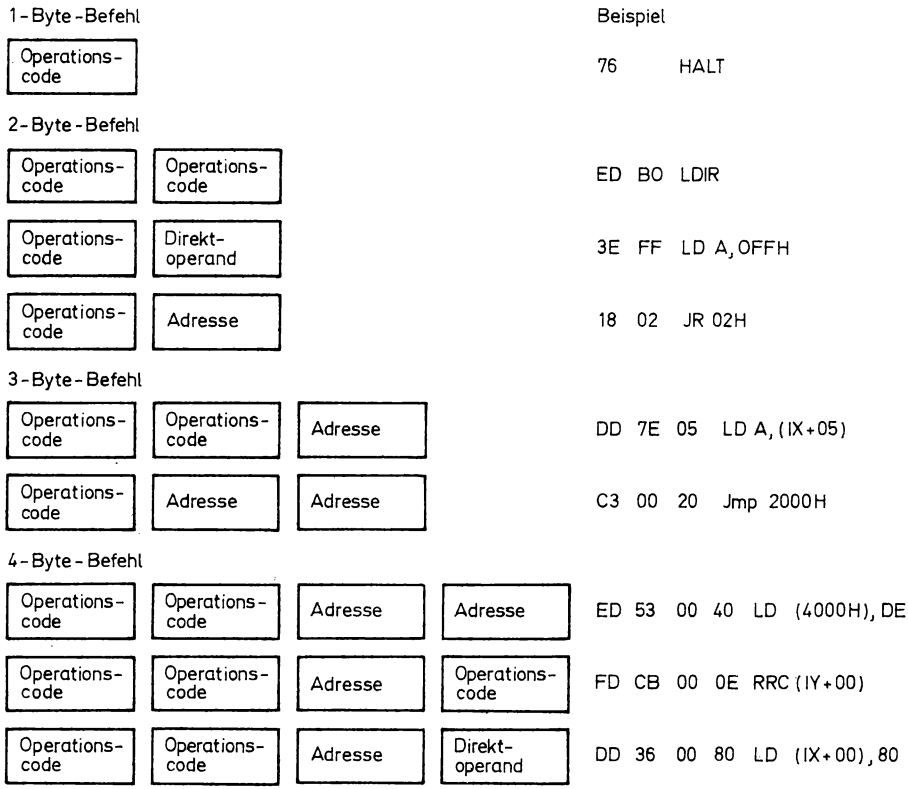


Bild 2.2
Befehlsstrukturen im *U880*
Anmerkung: Unter »Adresse« versteht man sowohl implizite Adreßangaben als auch Adreßmodifikationen (Adreßrechnungsbestandteile, die zum PC oder einem anderen Register addiert werden).
»Operanden« sind Werte, die im Akkumulator verarbeitet werden oder in ein Register/Speicherzelle geladen werden.

Direktoperand
Der zum Befehl gehörende Operand steht im Anschluß an den Operationscode:
Zelle 1 Operationscode,
Zelle 2 NWT-Operand (niederwertiger Teil des Operanden),
Zelle 3 HWT-Operand (höherwertiger Teil des Operanden).

Adressierter Operand
Im Befehl steht die Adresse der Speicherzelle, in der der Operand steht:
Zelle 1 Operationscode,
Zelle 2 NWT-ADR (niederwertiger Teil der Adresse),
Zelle 3 HWT-ADR (höherwertiger Teil der Adresse).

Relative Adressierung
Im Befehl steht eine positive oder negative Zahl N.
Der Operand steht um N Zellen nach oder vor dem Befehl.
Zelle 1 Operationscode
Zelle 2 N (positive oder negative Zahl im Zweierkomplement),
Zelle 3 nächster Befehl;
ADR als Operanden = ADR Zelle 3 + N.

Indirekte Adressierung
Die Adresse ADR des Operanden steht in einem speziellen Register:
ADR = ⟨Register⟩
Als Register treten die Registerpaare BC, DE, HL sowie die Register SP, IX und IY auf.

Indexierung
Die Adresse ADR des Operanden ergibt sich aus der im Befehl angegebenen Zahl N plus dem Inhalt eines Indexregisters.
Zelle 1 Operationscode
Zelle 2 N (positive oder negative Zahl im Zweierkomplement)
ADR = N + ⟨Indexregister⟩

2.3. Befehlsabarbeitung

2.3.1. Befehlszyklus

Für die Abarbeitung eines Befehls sind folgende Schritte erforderlich:
- Befehl holen,
- Befehl entschlüsseln,
- Operanden holen,
- Befehl ausführen und
- Adresse für nächsten Befehl ermitteln.

Die einzelnen Arbeitsgänge werden in Maschinenzyklen ausgeführt. Die Art des Maschinenzyklus wird durch die Befehlssignale, die aus der Befehlsentschlüsselung hervorgehen oder von außen als Signale des Steuerbus an den Prozessor gelangen, gebildet. In jedem Maschinenzyklus entstehen durch Hinzufügen von Zeitsignalen zu den Befehlssignalen interne Steuersignale, die die Abarbeitung in Form von Registertransporten steuern.

STS	=	BSI	·	ZSI
Steuersignal		Befehlssignal		Zeitsignal

Gleichzeitig werden zur Steuerung des Datentransfers mit den angeschlossenen Bausteinen äußere Steuersignale gebildet ($\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$, $\overline{MREQ}$, $\overline{M_1}$, $\overline{HALT}$, $\overline{BUSAK}$). Die Abarbeitung eines Befehls setzt sich aus mehreren Maschinenzyklen zusammen.
In einem Maschinenzyklus erfolgt außer der Abarbeitung interner Prozessorfunktionen der Zugriff zum Mikrorechnerbus. Entsprechend der Art dieses Zugriffs gibt es unterschiedliche Typen von Maschinenzyklen.
Es gibt Maschinenzyklen für folgende Funktionen:

- Operationscode holen,
- Speicher lesen,
- Speicher schreiben,
- Eingabe,
- Ausgabe,
- INTERRUPT,
- DMA-Funktion und
- Ausführung einer HALT-Operation.

Ein Maschinenzyklus unterteilt sich in 3 bis 6 Takte. Während der Befehlsabarbeitung werden mehrere Maschinenzyklen der unterschiedlichen Typen durchlaufen.
Bild 2.3 zeigt die Maschinenzyklen eines Befehls, in dem ein Byte aus dem Speicher geholt wird.

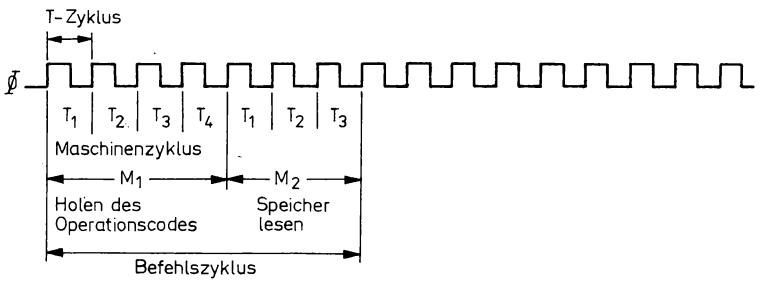


Bild 2.3
Maschinenzyklen für einen Befehl zum Lesen eines Bytes aus dem Speicher

2.3.2. Taktdiagramme der einzelnen Maschinenzyklen

Operationscode-Holzyklus (Bild 2.4)
Am Anfang des Zyklus enthält der Befehlszähler die Adresse des Operationscodes. Mit der Rückflanke von T_1 wird das Signal $\overline{MREQ}$ aktiv, gleichzeitig das Signal $\overline{RD}$. $\overline{MREQ}$ bedeutet eine Anforderung zum Speicher; $\overline{RD}$ sagt aus, daß eine Lese-Operation ablaufen soll. Das Einlesen der Daten geschieht mit der Vorderflanke von Φ während T_3. In den Zyklen T_3 und T_4 wird eine Auffrischadresse für dynamische Speicher an den Adreßbus gelegt. Die Auffrischadresse liegt an den Bitstellen A_0 bis A_6, während die übrigen Bit 0 sind. Zu dem Zeitpunkt, in dem Auffrischadresse am Adreßbus liegt, ist das Signal $\overline{RFSH}$ aktiv. Ist zum Zeitpunkt der Rückflanke von Φ im Zustand von T_2 das $\overline{WAIT}$-Signal aktiv, so wird nach T_2 ein Wartezustand eingeschoben. Dieses Einschieben von Wartezuständen wiederholt sich so lange, bis das $\overline{WAIT}$-Signal inaktiv wird.

Speicher-Lese-Zyklus (Bild 2.5)
Mit Beginn des Speicher-Lesezyklus (angesteuert durch die Vorderflanke von Φ im Zustand von T_1) wird die Speicheradresse auf den Adreßbus gelegt. Mit der Rückflanke von Φ (Zustand T_1) aktivieren sich die Signale $\overline{MREQ}$ und $\overline{RD}$. Das Signal $\overline{MREQ}$ kann zur Ansteuerung des betreffenden Speichers genommen werden, während $\overline{RD}$ den Speicher auf Lesen umschaltet. Zum Zeitpunkt der Rückflanke von Φ in T_2 tastet der Prozessor das $\overline{WAIT}$-Signal ab und fügt nach T_2 bei aktivem $\overline{WAIT}$-Signal einen Wartezyklus T_{WAIT} ein. Während

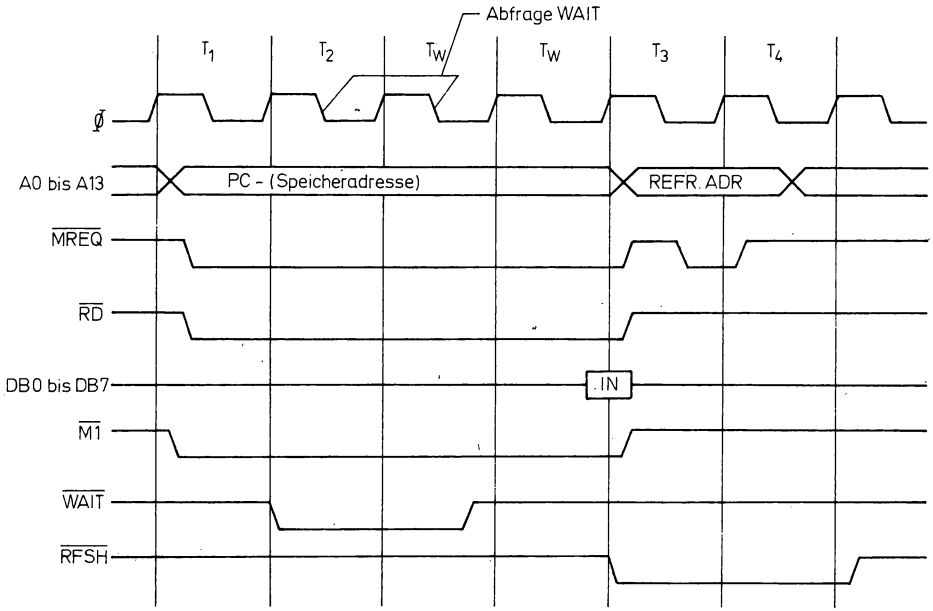


Bild 2.4
Operationscode-Holzyklus (M1-Zyklus) mit 2 Wartetakten.
Merkmal: Die Signale M1, MREQ und RD sind aktiv

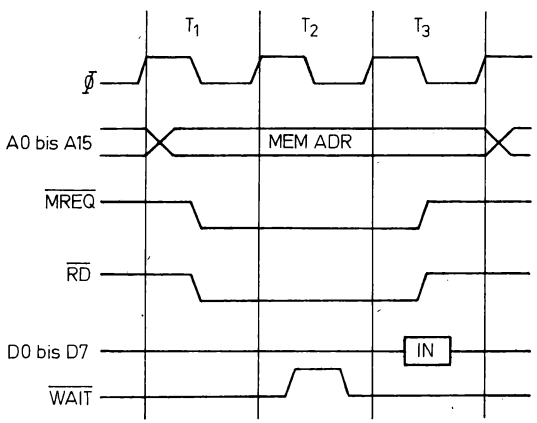


Bild 2.5
Speicher-Lesezyklus
Merkmal: Die Signale MREQ und RD sind aktiv

T_{WAIT} bleiben die Adresse am Adreßbus und die Daten am Datenbus erhalten. Bei der nächsten Rückflanke von Φ wird die Abfrage von $\overline{WAIT}$ wiederholt und eventuell ein weiterer Wartezustand eingeschoben. Ist das $\overline{WAIT}$-Signal nicht mehr aktiv, dann folgt der Zustand T_3. Während des Taktes Φ im Zustand T_3 werden die Daten vom Datenbus in den Prozessor übernommen, mit der Rückflanke von Φ in T_3 werden die Signale $\overline{MREQ}$ und $\overline{RD}$ wieder abgeschaltet.

Speicher-Schreib-Zyklus (Bild 2.6)
Beim Speicher-Schreib-Zyklus wird die Adresse genau wie zum Speicher-Lese-Zyklus mit der Vorderflanke von Φ in T_1 auf den Adreßbus gelegt. Mit der Rückflanke von Φ in T_1 werden die Daten an den Datenbus gelegt. Mit der Rückflanke von Φ in T_2 wird das Signal $\overline{WR}$ aktiv und gleichzeitig das $\overline{WAIT}$-Signal abgefragt. $\overline{WR}$ kann zum Umschalten des Speichers auf Schreiben benutzt werden. Während das Übernehmen der Daten in den Speicher mit Φ in T_3 erfolgen kann, wird mit der Rückflanke von Φ in T_3 $\overline{MREQ}$ und $\overline{RD}$ wieder abgeschaltet.

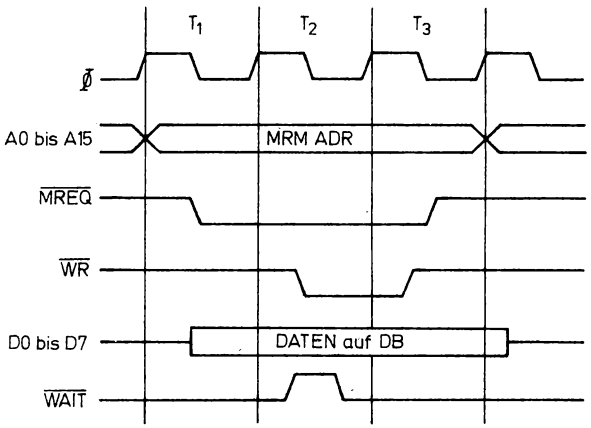


Bild 2.6
Speicher-Schreibzyklus
Merkmal: Die Signale MREQ und WR sind aktiv

Ein- und Ausgabe-Zyklus (Bilder 2.7 und 2.8)
Beim Ein- und Ausgabe-Zyklus wird nach T_2 automatisch ein Wartezyklus eingefügt, um dem Ein- und Ausgabebaustein zu ermöglichen, eine Adreßentschlüsselung durchzuführen und im Notfall das $\overline{WAIT}$-Signal zu setzen. Der Ablauf des Zyklus ähnelt dem des Speicher-Lese- oder -Schreib-Zyklus- Zum Zeitpunkt der Vorderflanke von Φ in T_2 wird das Signal $\overline{IORQ}$ aktiv. Gleichzeitig aktiviert sich entweder $\overline{RD}$ oder $\overline{WR}$, je nachdem, ob es sich um eine Eingabe oder um eine Ausgabe handelt. Bei der Ausgabe erscheinen die Daten auf dem Datenbus bereits während T_1, so daß zum Zeitpunkt Φ in T_3 die Daten abgenommen werden können.

INTERRUPT-Zyklus
Bild 2.9 zeigt das Zeitdiagramm für den maskierten INTERRUPT-Zyklus. Das Signal INT wird im letzten Zustand eines Befehls abgetastet. Ist es aktiv, dann beginnt mit T_1 ein INTERRUPT-Zyklus. Mit der Vorderflanke von Φ in T_1 gelangt die Adresse aus dem Befehlszähler an den Adreßbus. Gleichzeitig wird M_1 eingeschaltet. In jedem INTERRUPT-Zyklus werden automatisch 2 WAIT-Zustände eingeschoben, damit die INTERRUPT-Logik genügend Zeit zur Entschlüsselung der Adresse und zur Bereitstellung des INTERRUPT-Vektors hat. Mit der Rückflanke von Φ im ersten Wartezustand wird zusätzlich das Signal $\overline{IORQ}$ aktiv. Das gleichzeitige Vorhandensein von $\overline{IORQ}$ und $\overline{M_1}$ besagt, daß der INTERRUPT angenommen worden ist. Nach der Rückflanke von Φ des letzten Wartezustands wird vom Pro-

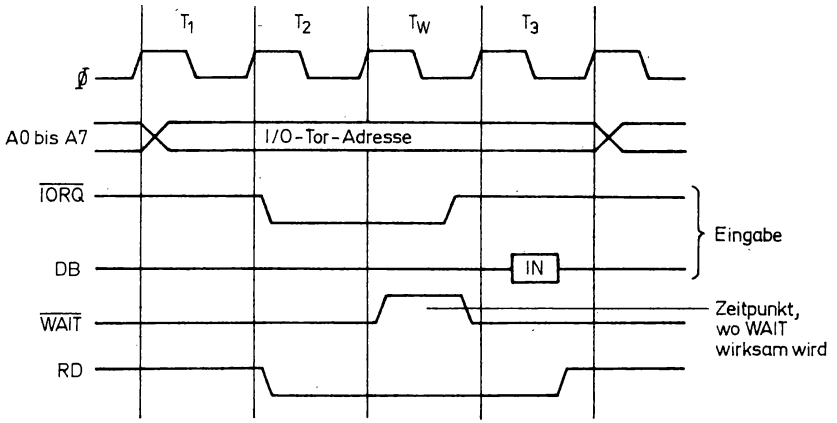


Bild 2.7
Eingabezyklus
Merkmal: Die Signale IORQ und RD sind aktiv. Ein Wartetakt wird automatisch eingefügt

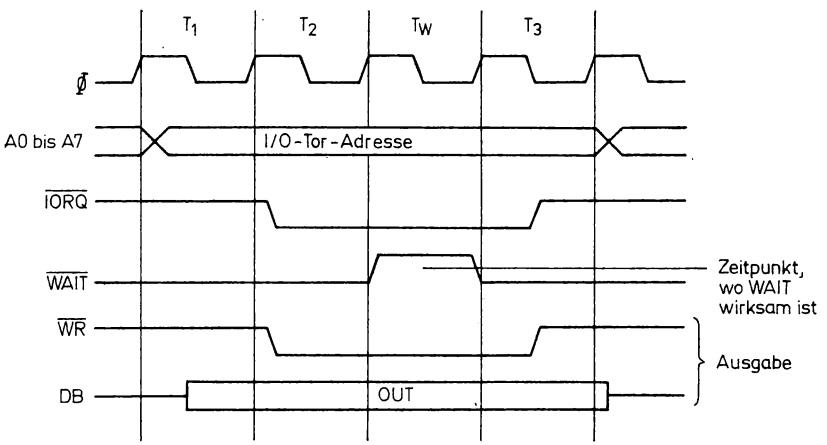


Bild 2.8
Ausgabezyklus
Merkmal: Die Signale IORQ und WR sind aktiv. Ein Wartetakt wird automatisch eingefügt

zessor der Datenbus abgetastet und als INTERRUPT-Vektor übernommen.
Während T_3 und T_4 kommt es wie beim Zyklus M_1 zur Ausgabe einer Auffrischadresse mit den dazugehörigen Signalen $\overline{MREQ}$ und $\overline{RFSH}$.
In Abhängigkeit vom INTERRUPT-MODE (0, 1, 2) wird der INTERRUPT-Vektor unterschiedlich interpretiert.

Maskierter INTERRUPT
MODE 0 Der INTERRUPT-Vektor wird als Befehlscode interpretiert.

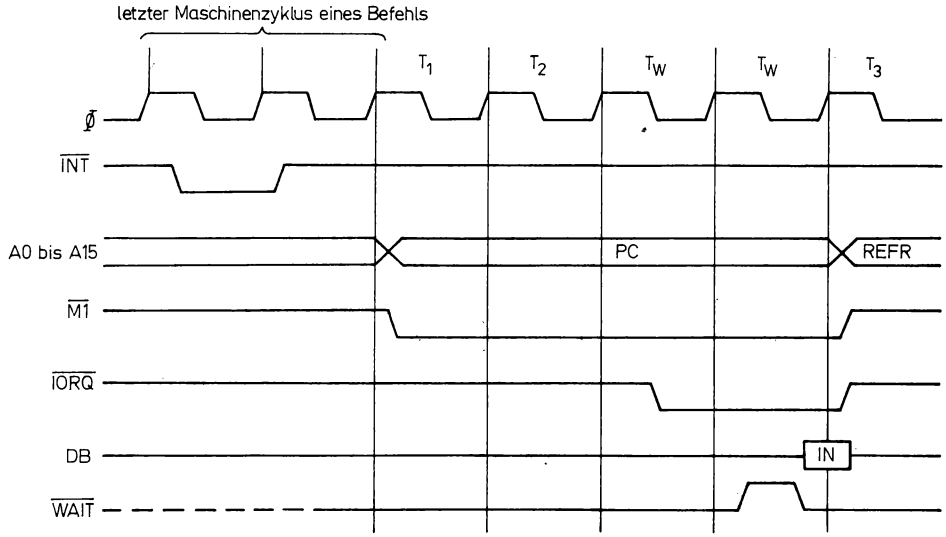


Bild 2.9
INTERRUPT-Zyklus (maskierbarer INTERRUPT)
Merkmal: Die Signale IORQ und M1 sind aktiv und ermöglichen die Eingabe des INTERRUPT-Vektors

MODE 1 Der INTERRUPT-Vektor bleibt unberücksichtigt. Es wird der Befehl CALL 38H gebildet und ausgeführt.

MODE 2 Der INTERRUPT-Vektor wird in Verbindung mit dem I-Register als Adresse interpretiert, die angibt, in welcher Zelle sich die Ansprungadresse des Bedienungsprogramms befindet. Es wird der Befehl CALL ⟨I-Register, INTERRUPT-Vektor⟩ ausgeführt.

Beim nichtmaskierten INTERRUPT wird ebenfalls ein Maschinenzyklus gebildet (Bild 2.10), in dem der Befehl CALL 66H im Befehlsregister aufgestellt wird.

Haltezyklus (Bild 2.11)
Nach der Entschlüsselung eines HALT-Befehls führt der Prozessor Leerbefehle (NOP) aus, und zwar so lange, bis ein INTERRUPT erscheint (entweder ein nichtmaskierter oder ein maskierter INTERRUPT, wenn dieser erlaubt ist). Die INTERRUPT-Eingänge werden mit der Vorderflanke von Φ in T_4 abgetastet. Ist zu diesem Zeitpunkt ein INTERRUPT-Eingang aktiv, dann setzt sich mit dem nächsten Takt die Befehlsabarbeitung fort. Es wird ein Sprung an die Stelle ausgeführt, die der entsprechenden INTERRUPT-Behandlung entspricht.

DMA-Zyklus (Bild 2.12)
Mit der Vorderflanke von Φ jedes letzten Taktes eines Maschinenzyklus wird das Signal $\overline{\text{BUSRQ}}$ abgetastet. Ist es zu diesem Zeitpunkt aktiv, so werden mit Beginn des nächsten T_1 der Adreßbus, der Datenbus und die Steuersignale $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{IORQ}}$ und $\overline{\text{RFSH}}$ in den hochohmigen Zustand gesetzt. Gleichzeitig aktiviert sich das Signal $\overline{\text{BUSAK}}$, als Zeichen dafür, daß der hochohmige Zustand erreicht ist. Nun wird in jedem Zustand mit der Vorderflanke von Φ das Signal $\overline{\text{BUSRQ}}$ abgetastet. Ist es nicht mehr aktiv, so wird im näch-

sten Takt der hochohmige Zustand beendet, und es beginnt ein neuer Maschinenzyklus. Während $\overline{BUSAK}$ aktiv ist, kann kein INTERRUPT auftreten. Der REFRESH ist unterbrochen.

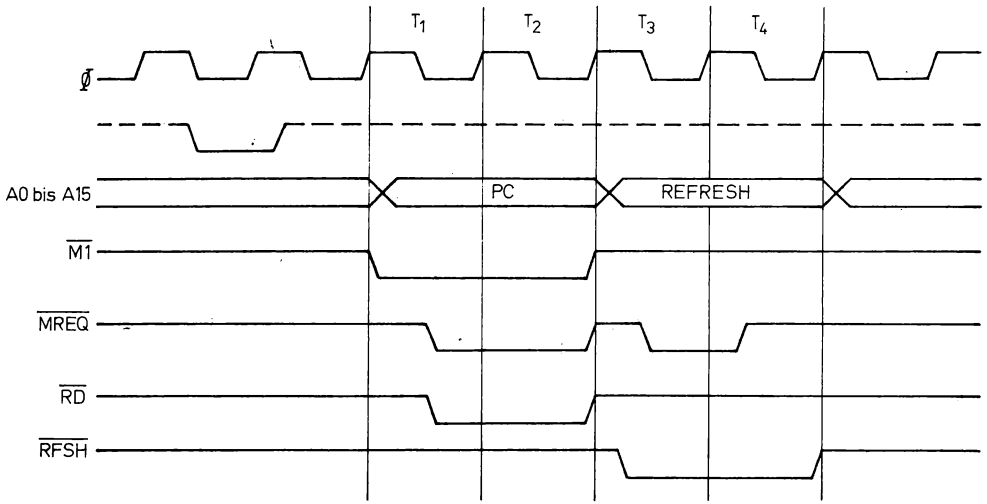


Bild 2.10
INTERRUPT-Zyklus (nichtmaskierbarer INTERRUPT)
Merkmal: Wie M1-Zyklus ohne Einlesen des Operationscode

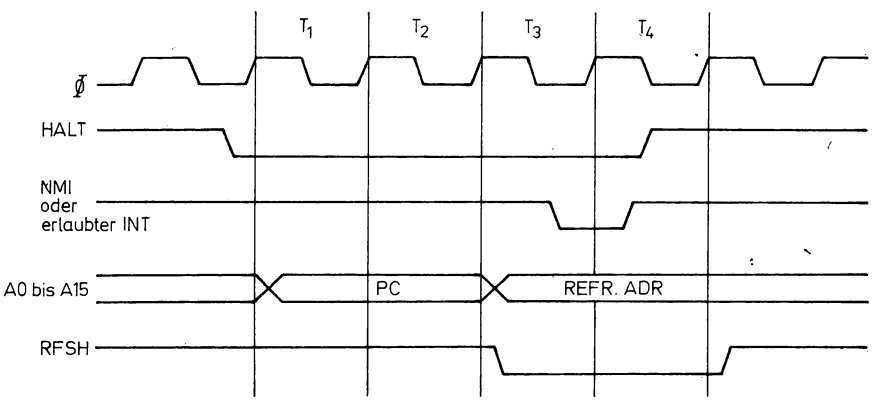


Bild 2.11
Haltezyklus
Merkmal: Das Signal HALT ist aktiv. Es werden REFRESH-Zyklen ausgeführt

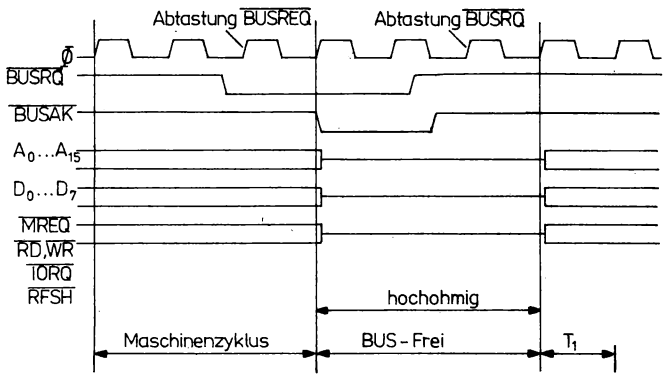


Bild 2.12
DMA-Zyklus
Merkmal: Das Signal BUSAK ist aktiv

2.4. Befehlsliste

2.4.1. Verwendete Abkürzungen

r – 8-Bit-Register des Registersatzes, A, B, C, D, E, H, L;
s – 8-Bit-Quellregister oder ein Speicherplatz oder eine 8-Bit-Zahl n
d – 8-Bit-Bestimmungsregister oder Speicherplatz
n – 8-Bit-Zahl
nn – 16-Bit-Zahl
dd – 16-Bit-Bestimmungsregister
ss – 16-Bit-Quellregister
s_b – Bit in einem speziellen 8-Bit-Register, b ist die Bit-Nr. (Zählung von rechts nach links, beginnend von Null).
Index L – der niederwertige Teil eines 16-Bit-Registers;
Index H – der höherwertige Teil eines 16-Bit-Registers.
– Steht ein Registername allein, z. B. A, so heißt das:
Inhalt von Register A.
– Steht ⟨HL⟩; so bedeutet das:
Inhalt der Speicherzelle, deren Adresse in HL steht.
– Steht ⟨nn⟩; so heißt das:
Inhalt der Speicherzelle, deren Adresse nn ist.
Bedeutung der Symbole für die Flagstellung (Bedeutung des Merkbits):
↕ Das Flag wird in Abhängigkeit vom Ergebnis der Operation beeinflußt.
· Das Flag bleibt unbeeinflußt.
0 Das Flag wird durch die Operation rückgesetzt.
1 Das Flag wird durch die Operation gesetzt.
V Das Flag wird in Abhängigkeit vom Überlauf des Ergebnisses beeinflußt.
P Das Flag wird in Abhängigkeit von der Parität des Ergebnisses beeinflußt.
X Das Flag ist beliebig.

2.4.2. Transportbefehle

Einzelworttransfer

	C	S	Z	P/V	H	N
LD r, s s → r		·		·		

Der Inhalt des Registers s oder eine Zahl n wird in ein Register r gebracht.
s kann sein: – Register A, B, C, D, E, H, L.
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird;
r kann sein: – Register A, B, C, D, E, H, L.

Beispiel
LD C, ⟨IX + 19H⟩
Der Inhalt von IX sei 25AFH. Durch den obigen Befehl wird der Inhalt der Zelle 25AFH + 19H = 25C8H nach Register C gebracht.

	C	S	Z	P/V	H	N
LD d, s s → d		·				

Der Inhalt des Registers s oder eine Zahl n wird nach der Zelle d gebracht.
s kann sein: – Register A, B, C, D, E, H, L oder eine Zahl n;
d kann sein: – ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird;
– Register A, B, C, D, E, H, L.

Beispiel
LD ⟨HL⟩, 28H
Der Inhalt von HL sei 25A5H. Durch obigen Befehl wird die Zahl 28H nach der Zelle 25A5H gebracht.

	C	S	Z	P/V	H	N	
LD A, s s → A		↕	↕	IFF	O	O	bei s = I, R
							bei s nicht I, R

Der Inhalt einer Zelle s wird nach Register A gebracht.
s kann sein: – ⟨BC⟩, ⟨DE⟩, d. h. eine Speicherzelle, deren Adresse in BC oder DE steht;
– (nn), d. h. eine Speicherzelle, deren Adresse nn ist;
– Register I oder R.

Beispiel
LD A, ⟨DE⟩
Der Inhalt von DE sei 8A25H. Durch obigen Befehl wird der Inhalt von Zelle 8A25H nach Register A gebracht.

	C	S	Z	P/V	H	N
LD d, A A → d						

Der Inhalt des Registers A wird nach der Zelle d gebracht.
d kann sein: – ⟨BC⟩, ⟨DE⟩, d. h. eine Speicherzelle, deren Adresse in BC oder DE steht;

- (nn), d. h. eine Speicherzelle, deren Adresse nn ist,
- Register I oder R.

Beispiel
LD ⟨25H⟩, A
Der Inhalt des Registers A wird nach der Zelle 25H gebracht.

Doppelworttransfer

C S Z P/V H N

LD dd, nn nn → dd

Der Direktoperand nn wird in das Doppelregister dd gebracht.
dd kann sein: BC, DE, HL, SP, IX, IY.

Beispiel
LD HL, 28DEH
Die Zahl 28DEH wird nach Register HL gebracht.

C S Z P/V H N

LD dd, ⟨nn⟩ ⟨nn⟩ → dd

Der Inhalt von Zelle nn und nn + 1 wird in das Doppelregister dd gebracht.
dd kann sein: BC, DE, HL, SP, IX, IY.

Beispiel
LD IX, ⟨8AH⟩
Der Inhalt von Zelle 8AH und 8BH wird in das Indexregister IX gebracht.

C S Z P/V H N

LD ⟨nn⟩, ss ss → ⟨nn⟩

Der Inhalt des Doppelregisters ss wird in die Speicherzelle nn und nn + 1 gebracht.
ss kann sein: BC, DE, HL, SP, IX, IY.

Beispiel
LD ⟨20H⟩, SP
Der Inhalt des SP wird nach Zelle 20H gebracht.

C S Z P/V H N

LD SP, ss ss → SP

Der Inhalt des Doppelregisters ss wird in den Stackpointer SP gebracht.
ss kann sein: HL, IX, IY.

Beispiel
LD SP, IX
Der Inhalt des Indexregisters IX wird in den Stackpointer SP gebracht.

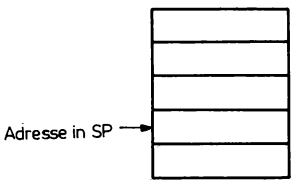
C S Z P/V H N

PUSH ss ss_H, ss_L → ⟨SP-1⟩, ⟨SP-2⟩
SP-2 → SP

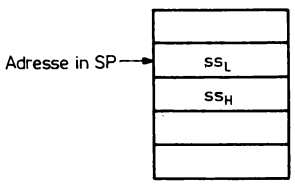
Der Inhalt des Doppelregisters ss wird in den Kellerspeicher gebracht. Der höherwertige Teil

ss_H kommt in die Zelle SP-1. Der niederwertige Teil in die Zelle SP-2. Nach Ausführung des Befehls ist der Inhalt des Stackpointers SP um 2 erniedrigt.

Kellerspeicher
vor Ausführung
von PUSH ss



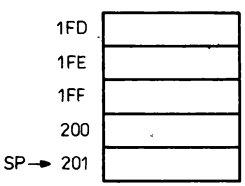
Kellerspeicher
nach Ausführung
von PUSH ss



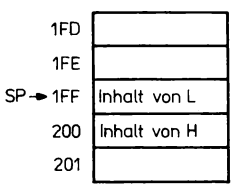
ss kann sein: AF, BC, DE, HL, IX, IY.

Beispiel
Der Inhalt von SP ist 201H

Kellerspeicher
vor PUSH HL



Kellerspeicher
nach PUSH HL

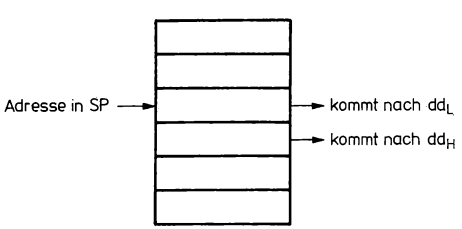


C S Z P/V H N

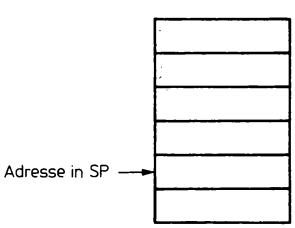
POP dd $\langle SP+1\rangle, \langle SP\rangle \rightarrow dd_H, dd_L$
$SP+2 \rightarrow SP$

Aus dem Kellerspeicher werden 2 Byte in das Doppelregister dd gebracht. Das Byte aus Zelle SP kommt in den niederwertigen Teil von dd, das Byte aus Zelle SP + 1 in den höherwertigen Teil von dd. Nach Ausführung des Befehls ist der Inhalt des Stackpointers SP um 2 erhöht.

Kellerspeicher
vor Ausführung
von POP dd



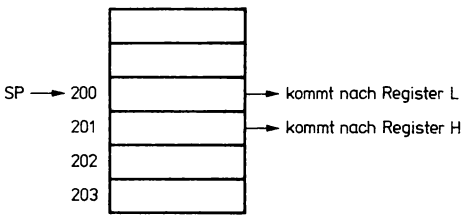
Kellerspeicher
nach Ausführung
von POP dd



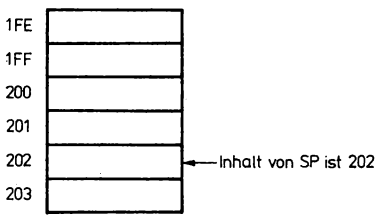
dd kann sein: AF, BC, DE, HL, IX, IY.

Beispiel
POP HL
Der Inhalt von SP sei 200H

Kellerspeicher vor POP HL



Kellerspeicher nach POP HL



Doppelworttransfer-Umtauschbefehle

C S Z P/V H N

EX DE, HL DE ↔ HL

Der Inhalt des Registerpaares DE wird mit dem Inhalt des Registerpaares HL vertauscht.

C S Z P/V H N

EX AF, AF′ AF ↔ A′F′

Die Inhalte der Register A und F werden mit den Inhalten der Register A′ und F′ vertauscht, und zwar A mit A′ und F mit F′.

C S Z P/V H N

EXX BC ↔ B′C′
DE ↔ D′E′
HL ↔ H′L′

Es werden die Inhalte der Register B mit B′, C mit C′, D mit D′, E mit E′, H mit H′ und L mit L′ vertauscht.

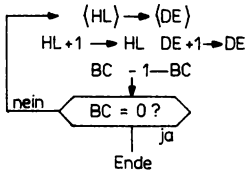
C S Z P/V H N

EX ⟨SP⟩, ss $ss_H, ss_L \leftrightarrow \langle SP+1\rangle, \langle SP\rangle$

Der Inhalt des Doppelregisters ss wird mit dem Inhalt von 2 Zellen des Kellerspeichers vertauscht. Es wird dabei der niederwertige Teil ss_L des Doppelregisters mit dem Inhalt der Zelle, deren Adresse in SP steht, und der höherwertige Teil ss_H mit der nächsten Zelle (Adresse SP + 1) vertauscht. Am Ende steht in SP der gleiche Wert wie vorher.
ss kann sein: HL, IX, IY.

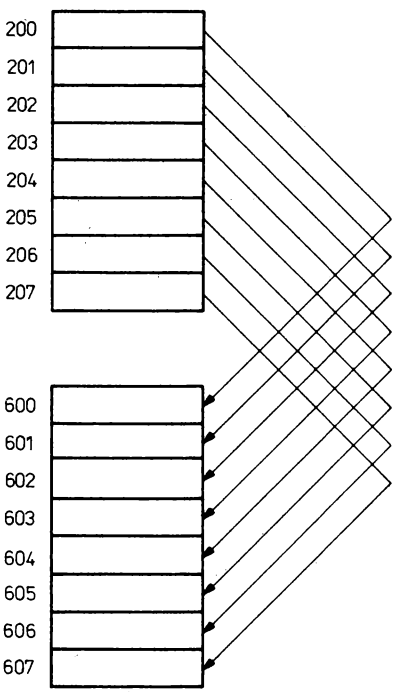
Blocktransfer

	C	S	Z	P/V	H	N
LDIR				0	0	0



Es wird der Inhalt eines Speicherbereiches, dessen Anfangsadresse in HL und dessen Blocklänge (Anzahl der Zellen des Speicherbereiches) in BC steht, in einem Speicherbereich mit der Anfangsadresse, die in DE steht, gespeichert.

Beispiel
Der Inhalt von HL sei 200, der von DE 600 und der von BC 8. Durch LDIR wird der Inhalt der Zellen 200 bis 207 in den Zellen 600 bis 607 gespeichert.



	C	S	Z	P/V	H	N
LDI ⟨HL⟩ → ⟨DE⟩				↕	0	0

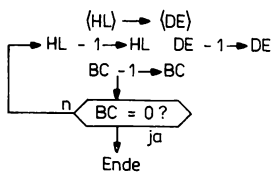
HL + 1 → HL DE + 1 → DE
BC − 1 → BC

Dieser Befehl dient zur Umspeicherung eines Speicherbereiches, dessen Anfangsadresse in

HL und dessen Blocklänge in BC steht, in einen Speicherbereich, dessen Anfangsadresse in DE steht. Durch eine einmalige Abarbeitung des Befehls wird der Inhalt der Zelle, deren Adresse in HL steht, in die Speicherzelle gebracht, deren Adresse in DE steht. Anschließend werden die Adressen in HL und DE um 1 erhöht und der Inhalt von BC um 1 erniedrigt. Ist BC − 1 = 0, so wird das Flag P/V = 0, sonst wird P/V = 1 gesetzt. Durch mehrmaliges Anwenden dieses Befehls läßt sich der Inhalt eines Speicherbereiches in einen anderen Speicherbereich umspeichern. Dabei kann entweder bei BC = 0 oder bei einer anderen Bedingung abgebrochen werden.

Beispiel
Der Inhalt von HL sei 300, der von DE 500 und der von BC sei 12.
Bei der 1. Befehlsabarbeitung von LDI kommt der Inhalt von Zelle 300 nach Zelle 500. Bei der 2. Abarbeitung von LDI wird der Inhalt von Zelle 301 nach Zelle 501 gebracht usw. Nach 12 Durchläufen (Inhalt von BC) ist der Inhalt von BC = 0, was als Endbedingung genommen werden kann.

	C	S	Z	P/V	H	N
LDDR				0	0	0



Es wird der Inhalt eines Speicherbereiches, dessen Endadresse in HL und dessen Blocklänge in BC steht, in einen Speicherbereich, dessen Endadresse in DE steht, gebracht.

Beispiel
Der Inhalt von HL sei 200H, der von DE 600H und der von BC 8H. Durch LDDR gelangt der Inhalt der Zellen 1F9-200H in die Zellen 5F9-600H.

	C	S	Z	P/V	H	N
LDD $\langle HL\rangle \rightarrow \langle DE\rangle$				↕	0	0

$HL - 1 \rightarrow HL, \quad DE - 1 \rightarrow DE$
$BC - 1 \rightarrow BC$

Dieser Befehl dient zur Umspeicherung eines Speicherbereiches, dessen Endadresse in HL und dessen Blocklänge in BC steht, in einen Speicherbereich, dessen Endadresse in DE steht.
Durch eine einmalige Abarbeitung des Befehls wird der Inhalt der Zelle, deren Adresse in HL steht, in die Speicherzelle gebracht, deren Adresse in DE steht. Anschließend werden die Adressen in HL und DE um 1 erniedrigt und der Inhalt von BC um 1 erniedrigt. Ist BC − 1 = 0, so wird das Flag P/V = 0, sonst wird P/V = 1 gesetzt. Durch mehrmaliges Anwenden dieses Befehls läßt sich der Inhalt eines Speicherbereiches in einen anderen Speicherbereich umspeichern. Dabei kann entweder bei BC = 0 oder bei einer anderen Bedingung abgebrochen werden.

2.4.3. Rechen- und logische Operationen mit einem Operanden

Akkumulator- und C-Bit-Befehle

		C	S	Z	P/V	H	N
CPL	$\overline{A} \rightarrow A$					1	1

Der Inhalt des Registers A wird bitweise negiert.

Beispiel
Der Inhalt von Register A sei 11001110.
Nach Ausführung des Befehls CPL ist der Inhalt des Registers A 00110001.

		C	S	Z	P/V	H	N
NEG	$\overline{A} + 1 \rightarrow A$ oder $0 - A \rightarrow A$	↕	↕	↕	V	↕	1

Vom Inhalt des Registers A wird das Zweierkomplement gebildet.

Beispiel
Der Inhalt von Register A sei 11001110.
Nach Ausführung des Befehls NEG ist der Inhalt des Registers A 00110010.

		C	S	Z	P/V	H	N
CCF	$\overline{C} \rightarrow C$	↕				↕	0

Der Inhalt des C-Bit wird negiert.
Im H-Bit wird der vorherige Wert des C-Bit gespeichert.

		C	S	Z	P/V	H	N
SCF	$1 \rightarrow C$	1				0	0

Der Inhalt des C-Bit wird 1 gesetzt.

		C	S	Z	P/V	H	N
DAA		↕	↕	↕	P	↕	·

Der Befehl DAA dient im Zusammenhang mit der Addition und der Subtraktion von Dualzahlen zur Berechnung von Summen oder Differenzen zweier im BCD-Code dargestellter Zahlen.

Beispiel
Im Register A stehe die Zahl 28 im BCD-Code, d. h., der Inhalt von Register 00101000. Im Register B stehe die Zahl 17 im BCD-Code, d. h., B = 00010111. Nach der dualen Addition der Inhalte von Register A und B steht im Register A 00111111.
Das ist jedoch nicht die BCD-Code-Darstellung von 28 + 17 = 45. Der Befehl DAA verändert A = 00111111 in den Wert A = 01000101 (= 45 in BCD-Darstellung).

Einzelwortbefehle

		C	S	Z	P/V	H	N
INC	d $\quad d + 1 \rightarrow d$		↕	↕	V	↕	0

Der Inhalt der Zelle oder des Registers d wird um 1 erhöht.
d kann sein: – Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;

– ⟨IX + d⟩, ⟨IY + d⟩, d. h. Inhalt einer Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

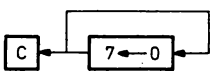
Beispiel
Der Inhalt vom Indexregister IY sei 1A5H.
Durch den Befehl INC ⟨IY + 17H⟩ wird der Inhalt von Zelle 1A5H + 17H = 1BCH um 1 erhöht.

			C	S	Z	P/V	H	N
DEC	d	d − 1 → d		↕	↕	V	↕	1

Der Inhalt der Zelle oder des Registers d wird um 1 erniedrigt,
d kann sein:
– Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

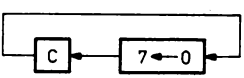
Beispiel
Der Inhalt des Doppelregisters HL sei 800.
Durch den Befehl DEC ⟨HL⟩ wird der Inhalt der Zelle 800 um 1 erniedrigt.

Verschiebebefehle



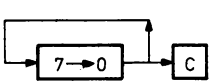
		C	S	Z	P/V	H	N
RLC	s	↕	↕	↕	P	0	0
RLC	A[4]	↕		·		0	0

Der Inhalt des Registers oder der Zelle s wird um eine Stelle nach links verschoben. Das aus Bit 7 (Zählung von rechts nach links) heraustretende Bit wird in das C-Bit und Bit 0 eingetragen.
s kann sein:
– Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.



		C	S	Z	P/V	H	N
RL	s	↕	↕	↕	P	0	0
RL	A[5]	↕	·	·		0	0

Der Inhalt des Registers s oder der Zelle s wird zusammen mit dem C-Bit um eine Stelle nach links verschoben. Das aus Bit 7 kommende Bit wird in das C-Bit und das C-Bit in Bit 0 eingetragen.
s kann sein:
– Register A, B, C, D, E, H, L;
– ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
– ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

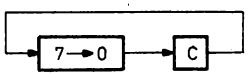


		C	S	Z	P/V	H	N
RRC	s	↕	↕	↕	P	0	0
RRC	A[5]	↕	·	·	·	0	0

Der Inhalt des Registers s oder der Zelle s wird um eine Stelle nach rechts verschoben. Das aus Bit 0 heraustretende Bit wird in das C-Bit und in Bit 7 eingetragen.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

RR s
RR A[6]



C	S	Z	P/V	H	N
↕	↕	↕	P	0	0
↕	·			0	0

Der Inhalt des Registers s oder der Zelle s wird zusammen mit dem C-Bit um eine Stelle nach rechts verschoben. Bit 0 kommt ins C-Bit und das C-Bit nach Bit 7.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

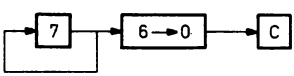
SLA s

C	S	Z	P/V	H	N
↕	↕	↕	P	0	0

Dieser Befehl bewirkt eine arithmetische Linksverschiebung des Registers oder der Speicherzelle s. Das aus Bit 7 heraustretende Bit wird in das Register C-Bit eingetragen.
In Bit 0 wird eine 0 eingetragen. Der Befehl entspricht der Multiplikation des Registerinhalts mit 2.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

SRA s

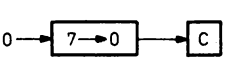


C	S	Z	P/V	H	N
↕	↕	↕	P	0	0

Dieser Befehl bewirkt eine arithmetische Rechtsverschiebung des Registers oder der Speicherzelle s. Bit 7 bleibt erhalten. In Bit 6 wird Bit 7 eingetragen, Bit 0 kommt ins C-Bit.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

SRL s



C	S	Z	P/V	H	N
↕	↕	↕	P	0	0

Dieser Befehl bewirkt eine Rechtsverschiebung des Registers oder der Zelle s. Dabei wird in Stelle 7 eine 0 und in das C-Bit Bit 0 eingetragen.

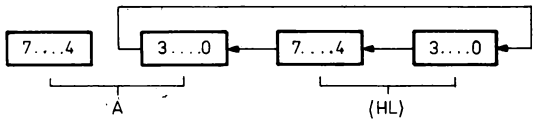
s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

[4] RLCA führt dieselbe Funktion wie RLC A aus, ist jedoch ein 1-Byte-Befehl.

[5] RLA und RRCA führen dieselbe Verschiebung wie RL A und RRC A aus, sind aber 1-Byte-Befehle.

[6] RRA führt dieselbe Verschiebung wie RR A aus, ist aber ein 1-Byte-Befehl.

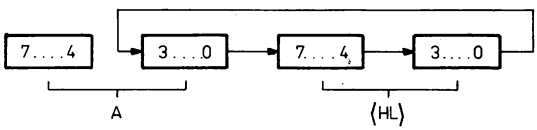
RLD



C	S	Z	P/V	H	N
	↕	↕	P	0	0

Dieser Befehl bewirkt eine Linksverschiebung um eine Tetrade (4 Bit) des Registers A und einer Speicherzelle, deren Adresse in HL steht.
Dabei werden eingetragen:
- die niederwertige Tetrade von A in die niederwertige Tetrade der Speicherzelle,
- die niederwertige Tetrade der Speicherzelle in die höherwertige Tetrade der Speicherzelle und
- die höherwertige Tetrade der Speicherzelle in die niederwertige Tetrade des Registers A.

RRD



C	S	Z	P/V	H	N
	↕	↕	P	0	0

Dieser Befehl bewirkt eine Rechtsverschiebung um eine Tetrade (4 Bit) des Registers A und einer Speicherzelle, deren Adresse in HL steht.
Dabei werden eingetragen:
- die niederwertige Tetrade von A in die höherwertige Tetrade der Speicherzelle,
- die höherwertige Tetrade der Speicherzelle in die niederwertige Tetrade der Speicherzelle und die
- niederwertige Tetrade der Speicherzelle in die niederwertige Tetrade des Registers A.

Beispiel (zu den Verschiebebefehlen)
Im Register D stehe die Bit-Folge 01110011 und im C-Bit eine 1.
Befehl: Information im D-Register und C-Bit nach dem Befehl:

Befehl	C-Bit	Register D
RLC D	0	11100110
RL D	0	11100111
RRC D	1	10111001
RR D	1	10111001
SLA D	0	11100110
SRA D	1	00111001
SRL D	1	00111001

Bit-Befehle

	C	S	Z	P/V	H	N
BIT b, s $\overline{s_b} \rightarrow Z$	·	X	↕	X	1	0

Das Bit b des Registers oder der Speicherzelle s wird in negierter Form in das Z-Flag gebracht, b ist eine Zahl zwischen 0 und 7.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

Bit-Zählung

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

Beispiel

BIT 3, ⟨HL⟩

Bit 3 der Speicherzelle, deren Adresse in HL steht, wird in negierter Form in das Z-Flag gebracht.
- In HL steht die Adresse 80H.
- In der Zelle 80H steht 11101110.
- Durch Bit 3, ⟨HL⟩ wird eine 0 in das Z-Flag gebracht.

	C	S	Z	P/V	H	N
SET b, s $1 \rightarrow s_b$					·	

Das Bit b des Registers oder der Speicherzelle s wird 1 gesetzt, b ist eine Zahl zwischen 0 und 7.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

	C	S	Z	P/V	H	N
RES b, s $0 \rightarrow s_b$						

Das Bit b des Registers oder der Speicherzelle s wird 0 gesetzt.

s kann sein:
- Register A, B, C, D, E, H, L;
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

Doppelwortbefehle

	C	S	Z	P/V	H	N
INC dd dd + 1 → dd						

Der Inhalt des Doppelregisters dd wird um 1 erhöht.
dd kann sein: BC, DE, HL, SP, IX, IY

	C	S	Z	P/V	H	N
DEC dd dd − 1 → dd						

Der Inhalt des Doppelregisters dd wird um 1 erniedrigt.
dd kann sein: BC, DE, HL, SP, IX, IY.

2.4.4. Rechen- und logische Operationen mit 2 Operanden

1-Wort-Befehle

			C	S	Z	P/V	H	N
ADD	A,s	A + s → A	↕	↕	↕	V	↕	0
ADC	A,s	A + s + C → A						

Der Inhalt des Akkumulators und der Inhalt des Registers oder der Zelle s bzw. C-Bit werden addiert, und das Ergebnis wird ins Register A gebracht.

s kann sein:
- Direktoperand n (8-Bit-Zahl);
- ⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht;
- ⟨IX + d⟩, ⟨IY + d⟩, d. h. eine durch Indexierung ermittelte Speicherzelle;
- Register A, B, C, D, E, H, L.

			C	S	Z	P/V	H	N
SUB	A,s	A − s → A	↕	↕	↕	V	↕	1
SBC	A,s	A − s − C → A						

Der Inhalt der Zelle oder des Registers s bzw. C-Bit wird vom Register A subtrahiert. Das Ergebnis kommt in das Register A.

s kann sein: siehe 1-Wort-Befehl ADD s.

Die logischen Operationen AND s (A ∧ s → A), OR s (A ∨ s → A), XOR s (A ⊕ s → A) werden analog abgearbeitet. Das P/V-Bit wird als P gewertet.

		C	S	Z	P/V	H	N
CP	s	↕	↕	↕	V	↕	1

Der Inhalt des Registers A wird mit dem Inhalt des Registers oder der Zelle s verglichen.

Doppelwortbefehle

ADD HL,ss HL ss HL

		C	S	Z	P/V	H	N
ADC	HL, ss HL + ss + C → HL	↕	↕	↕	V	↕	0

Der Inhalt des Doppelregisters HL wird mit dem Inhalt des Doppelregisters ss bzw. C-Bit addiert. Zum Ergebnis wird das C-Bit in die niedrigste Stelle addiert. Das letzte Ergebnis kommt nach HL. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.

ss kann sein: BC, DE, HL, SP.

		C	S	Z	P/V	H	N
SBC	HL, ss HL − ss − C → HL	↕	↕	↕	V	↕	1

Der Inhalt des Doppelregisters ss wird vom Doppelregister HL subtrahiert. Anschließend wird davon das C-Bit in der letzten Stelle subtrahiert. Das Ergebnis kommt nach HL, Bit H wird vom Übertrag von Bit 12 gesetzt.

ss kann sein: BC, DE, HL, SP.

		C	S	Z	P/V	H	N
ADD	IX, ss IX + ss → IX	↕	·	·	↕	0	

Der Inhalt des Doppelregisters ss wird zum Inhalt des Indexregisters IX addiert. Das Ergebnis kommt nach IX. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.

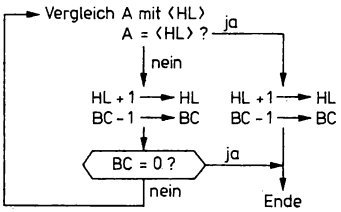
ss kann sein: BC, DE, IX, SP.

	C	S	Z	P/V	H	N
ADD IY, ss IY + ss → IY	↕		·		↕	0

Der Inhalt des Doppelregisters ss wird zum Inhalt des Indexregisters IY addiert. Das Ergebnis kommt nach IY. Bit H wird mit dem Übertrag aus Bit 11 gesetzt.
ss kann sein: BC, DE, IY, SP.

2.4.5. Rechen- und logische Operationen mit mehreren Operanden

CPIR



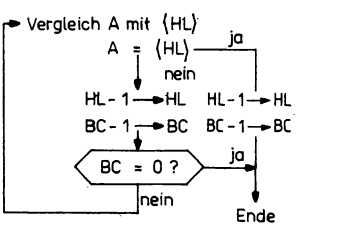
C	S	Z	P/V[7]	H	N
	↕	↕	↕	↕	1

Der Inhalt eines Speicherbereiches, dessen Anfangsadresse in HL und dessen Länge in BC steht, wird nach einer Information im Register A durchsucht. Befindet sich die Information in einer Zelle des Speicherbereiches, so ist der Befehl an dieser Stelle beendet. Befindet sich die Information nicht in dem gesuchten Speicherbereich, so wird der Befehl nach der letzten Zelle des Bereiches beendet. Wird die Information im Speicherbereich gefunden, so wird trotzdem HL um 1 erhöht und BC um 1 erniedrigt. Im Flagregister steht das Ergebnis der letzten Vergleichsoperation. Das Bit P/V ist 1, wenn BC − 1 ≠ 0, sonst 0.

	C	S	Z	P/V[8]	H	N
CPI Vergleich A mit ⟨HL⟩ HL + 1 → HL BC − 1 → BC		↕	↕	↕	↕	1

Der Inhalt des Speicherplatzes, dessen Adresse in HL steht, wird mit dem Inhalt des A-Registers verglichen und das Flagregister gesetzt. Das P/V-Flag ist 1, wenn BC − 1 ≠ 0, sonst 0. Anschließend wird der Inhalt von HL um 1 erhöht und der Inhalt von BC um 1 erniedrigt. Mit Hilfe des Befehls CPI läßt sich ein Speicherbereich auf eine Bit-Folge, die im A-Register steht, durchsuchen, wobei die Abbruchbedingung frei wählbar ist.

CPDR



C	S	Z	P/V[8]	H	N
	↕	↕	↕	↕	1

Der Inhalt eines Speicherbereiches, dessen Endadresse in HL und dessen Länge in BC steht, wird nach einer Information, die im Register A steht, durchsucht. Befindet sich die Informa-

[7] Wenn BC zum Befehlsende 0 ist, dann wird P/V = 0, ist BC ≠ 0, so wird P/V = 1 gesetzt.

[8] Wenn BC nach Befehlsausführung 0 ist, dann wird P/V = 0; ist BC ≠ 0, so wird P/V = 1 gesetzt.

tion in einer Zelle des Speicherbereichs, so wird der Befehl an dieser Stelle beendet. Steht die Information nicht in dem gesuchten Speicherbereich, so endet der Befehl nach der ersten Zelle des Bereiches. Wird die Information im Speicherbereich gefunden, so werden trotzdem HL und BC um 1 erniedrigt.
Im Flagregister steht das Ergebnis der letzten Vergleichsoperation. Das Bit P/V ist 1, wenn BC − 1 ≠ 0, sonst 0.

Beispiel
Die Zellen 200H bis 205H sollen nach der Bit-Folge 00000111 durchsucht werden. In den Zellen 200H bis 205H stehen:
200H 11111111
201H 00000000
202H 01000000
203H 00000111
204H 00000000
205H 10000001

Dazu bringt man die obige Bit-Folge in das Register A, die Adresse 205H in das Register HL und die Anzahl 6 in das Registerpaar BC. Nach Ausführung des Befehls CPDR steht in HL 202H, und das Z-Bit ist gesetzt.
Würde sich in Zelle 203H die Bit-Folge 01010101 befinden, so stünde nach Abarbeitung des Befehls CPDR in HL 1FFH, und das Z-Bit würde rückgesetzt sein.

		C	S	Z	P/V[8]	H	N
CPD	Vergleich A mit ⟨HL⟩		↕	↕	↕	↕	1
	HL − 1 → HL						
	BC − 1 → BC						

Der Inhalt des Speicherplatzes, dessen Adresse in HL steht, wird mit dem Inhalt des A-Registers verglichen und das Flagregister gesetzt. Das P/V-Flag ist 1, wenn BC − 1 ≠ 0, sonst 0. Anschließend werden der Inhalt von HL und der von BC um 1 erniedrigt. Mit Hilfe des Befehls CPD läßt sich ein Speicherbereich auf eine Bit-Folge, die im A-Register steht, durchsuchen, wobei die Abbruchbedingung frei wählbar ist.

2.4.6. Sprungbefehle

			C	S	Z	P/V	H	N
JP	nn	nn → PC						

Die Adresse nn wird in den Befehlszähler PC gebracht. Der nächste abzuarbeitende Befehl ist damit der Befehl aus Zelle nn. Man sagt, der Rechner führt einen Sprung in die Zelle nn aus.

			C	S	Z	P/V	H	N
JP	cc, nn	Wenn cc = true, nn → PC						

Wenn die Bedingung cc erfüllt ist, dann wird die Adresse nn in den Befehlszähler PC gebracht.

cc kann sein: – NZ Z-Bit rückgesetzt,
– Z Z-Bit gesetzt,
– NC C-Bit rückgesetzt,
– C C-Bit gesetzt,
– PO Paritätsbit auf ungerade gesetzt,
– PE Paritätsbit auf gerade gesetzt,
– P Vorzeichenbit auf positiv gesetzt,
– M Vorzeichenbit auf negativ gesetzt.

C S Z P/V H N

JR e PC + e → PC

Der Befehlszähler PC wird um die Zahl e verändert. e ist eine 8-Bit-Zahl im Zweierkomplement, d. h., e kann auch negativ sein. Ausgangspunkt für PC + e → PC ist die Adresse des Operationscodes des Befehls. Der Rechner führt einen Sprung um e Zellen aus. Die Zahl e steht als e − 2 in der Zelle nach dem Operationscode.

Beispiel
Von Zelle 100 soll ein Sprung nach 105 erfolgen.
Befehl: JR 5

```
┌ 100  18     ← Operationscode des Befehls JR
│ 101   3     ← e − 2
│ 102  − −
│ 103  − − −
│ 104  − − −
└→105  − − −  ← PC-Stand nach dem Befehl.
```

C S Z P/V H N

JR kk, e Wenn kk = true, PC + e → PC
wenn kk = not true, Leerbefehl

Wenn die Bedingung kk erfüllt ist, dann führt der Prozessor einen Sprung um e Zellen aus.
kk kann sein: – NZ Z-Bit rückgesetzt,
– Z Z-Bit gesetzt,
– NC C-Bit rückgesetzt,
– C C-Bit gesetzt.

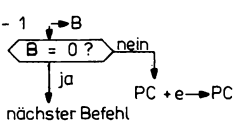
C S Z P/V H N

JP ⟨ss⟩ ss → PC

Der Inhalt des Doppelregisters ss wird nach PC gebracht. Der Rechner führt einen Sprung in die Zelle aus, deren Adresse in ss steht.
ss kann sein: HL, IX, IY.

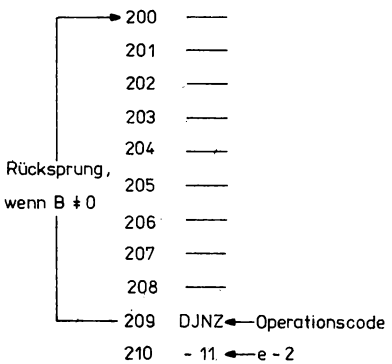
C S Z P/V H N

DJNZ e



Solange B − 1 ≠ 0 ist, führt der Rechner einen Sprung um e Zellen aus. e ist eine 8-Bit-Zahl im Zweierkomplement, d. h., e kann auch negativ sein. Ausgangspunkt für PC + e → PC ist die Adresse des Operationscodes des Befehls. Die Zahl e steht als e − 2 in der Zelle nach dem Operationscode.

Beispiel
Eine Programmschleife von Zelle 200 bis 209 soll 10mal durchlaufen werden. Dazu bringt man eine 10 in das Register B. Der Befehl, der diese 10fache Programmschleife realisiert, lautet DJNZ − 11.
Aufbau der Programmschleife:



2.4.7. Unterprogrammbefehle

Unterprogrammrufe

C S Z P/V H N

CALL nn $PC_H, PC_L \rightarrow \langle SP-1\rangle, \langle SP-2\rangle$
$SP-2 \rightarrow SP$
$nn \rightarrow PC$

Der CALL-Befehl realisiert einen Sprung in ein Unterprogramm, dessen Startadresse nn ist. Dabei wird der aktuelle Befehlszählerstand (Adresse des Operationscodes des nächsten Befehls) im STACK gespeichert. Der höherwertige Teil von PC kommt in die Zelle, deren Adresse sich aus den um 1 erniedrigten Inhalt des STACKPOINTER SP ergibt. Der niederwertige Teil von PC gelangt in die Zelle, deren Adresse sich aus dem um 2 erniedrigten Inhalt des STACKPOINTER ergibt. Der STACKPOINTER ist am Ende des Befehls um 2 erniedrigt.

C S Z P/V H N

CALL cc, nn wenn cc = true
$PC_H, PC_L \rightarrow \langle SP-1\rangle, \langle SP-2\rangle$
$SP-2 \rightarrow SP$
$nn \rightarrow PC$

Wenn die vorgegebene Bedingung cc erfüllt ist, so wird der Befehl CALL nn ausgeführt. Ist die Bedingung nicht erfüllt, dann hat der Befehl die Funktion eines Leerbefehls.
cc kann sein: – NZ Z-Bit rückgesetzt,

- Z Z-Bit gesetzt,
- NC C-Bit rückgesetzt,
- C C-Bit gesetzt,
- PO Paritätsbit auf ungerade gesetzt,
- PE Paritätsbit auf gerade gesetzt,
- P Vorzeichenbit auf positiv gesetzt,
- M Vorzeichenbit auf negativ gesetzt.

C S Z P/V H N

RST z $PC_H, PC_L \rightarrow \langle SP-1\rangle, \langle SP-2\rangle$
$SP-2 \rightarrow SP$
$z \rightarrow PC$

z ist eine der Hexadezimaladressen 0H, 8H, 10H, 18H, 20H, 28H, 30H, 38H.
Der Befehl RST z (RESTART) ist ein CALL-Befehl mit der Adresse z. Er wird in Verbindung mit dem INTERRUPT in MODE 0 verwendet. Hier muß während des INTERRUPT-Zyklus über den Datenbus ein 1-Byte-Befehl in den Prozessor gegeben werden. Der RST-Befehl stellt einen 1-Byte-CALL-Befehl dar, während der vollständige CALL-Befehl 3 Byte lang ist.

Rückkehrbefehle

C S Z P/V H N

RET $\langle SP+1\rangle, \langle SP\rangle \rightarrow PC_H, PC_L$
$SP+2 \rightarrow SP$

Der RET-Befehl realisiert den Rücksprung aus einem Unterprogramm. Durch ihn wird die zuletzt in den STACK gespeicherte Adresse in den Befehlszähler PC gebracht. Der Inhalt der Zelle, dessen Adresse im STACKPOINTER steht, kommt in den niederwertigen Teil von PC, der Inhalt der Zelle, dessen Adresse sich aus dem Inhalt des STACKPOINTER plus 1 ergibt, in den höherwertigen Teil von PC. Der STACKPOINTER ist am Ende des Befehls um 2 erhöht.

C S Z P/V H N

RET cc wenn cc = true;
$\langle SP+1\rangle, \langle SP\rangle \rightarrow PC_H, PC_L$
$SP+2 \rightarrow SP$

Wenn die vorgegebene Bedingung cc erfüllt ist, dann wird der Befehl RET ausgeführt. Ist die Bedingung nicht erfüllt, so hat der Befehl die Funktion eines Leerbefehls.
cc kann sein:
- NZ Z-Bit zurückgesetzt,
- Z Z-Bit zurückgesetzt,
- NC C-Bit zurückgesetzt,
- C C-Bit gesetzt,
- PO Paritätsbit auf ungerade gesetzt,
- PE Paritätsbit auf gerade gesetzt,
- P Vorzeichenbit auf positiv gesetzt,
- M Vorzeichenbit auf negativ gesetzt,

C S Z P/V H N

RETI $\langle SP + 1\rangle, \langle SP\rangle \rightarrow PC_H, PC_L$
$SP + 2 \rightarrow SP$
IFF2 $\rightarrow$ IFF1

Der Befehl RETI dient als Rücksprungbefehl beim maskierten INTERRUPT.

C S Z P/V H N

RETN $\langle SP + 1\rangle, \langle SP\rangle \rightarrow PC_H, PC_L$
$SP + 2 \rightarrow SP$
IFF2 $\rightarrow$ IFF1

Der Befehl RETN dient als Rücksprungbefehl beim nichtmaskierten INTERRUPT.

2.4.8. Ein- und Ausgabebefehle

Einzelwortein- und -ausgabe

C S Z P/V H N

IN A, $\langle n\rangle$ $\langle n\rangle \rightarrow A$

Innerhalb eines Eingabezyklus wird das auf dem Datenbus vorhandene Byte in das Register A gebracht. Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den alten Inhalt des Registers A und der niederwertige Teil die Zahl n, die als Adresse für das periphere Gerät dient.

C S Z P/V H N

IN r, $\langle C\rangle$ $\langle C\rangle \rightarrow r$ ↕ ↕ P 0 0

Innerhalb eines Eingabezyklus wird das auf dem Datenbus vorhandene Byte in das Register r gebracht. Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des Registers B und der niederwertige Teil den Inhalt des Registers C, der als Adresse für das periphere Gerät dient.
r kann sein: Register A, B, C, D, E, F, H, L.

C S Z P/V H N

OUT $\langle n\rangle$, A $A \rightarrow \langle n\rangle$

Innerhalb eines Ausgabezyklus wird der Inhalt des Registers A auf den Datenbus gebracht. Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den alten Inhalt des Registers A und der niederwertige Teil die Zahl n, die als Adresse für das periphere Gerät dient.

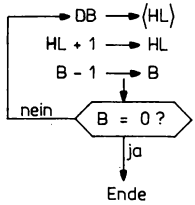
C S Z P/V H N

OUT $\langle C\rangle$, r $r \rightarrow \langle C\rangle$

Innerhalb eines Ausgabezyklus wird der Inhalt des Registers r auf den Datenbus gebracht. Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des Registers B und der niederwertige Teil den Inhalt des Registers C, der als Adresse für das periphere Gerät dient.
r kann sein: Register A, B, C, D, E, H, L.

Blocktransfer

INIR



C	S	Z	P/V	H	N
	X	1	X	X	1

Adreßbus:

A_{15} bis A_8	A_7 bis A_0
Inhalt des B-Registers	Inhalt des C-Registers

Der Prozessor führt eine Reihe Eingabebefehle durch, deren Anzahl n im Register B steht. Während jedes Eingabezyklus wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Nach jedem Eingabezyklus erhöht sich der Inhalt von HL um 1, und der Inhalt von B wird um 1 erniedrigt. Insgesamt liest der Prozessor n Bytes ein, die in einen Speicherbereich gebracht werden, dessen Anfangsadresse in HL steht. Während eines Eingabezyklus enthält der niederwertige Teil des Adreßbus den Inhalt des C-Registers, der als periphere Adresse dient.

INI DB $\rightarrow$ (HL)
HL + 1 $\rightarrow$ HL
B − 1 $\rightarrow$ B

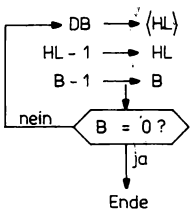
C	S	Z	P/V	H	N
	X	↕	X	X	1

Adreßbus:

A_{15} bis A_8	A_7 bis A_0
Inhalt des B-Registers	Inhalt des C-Registers

Der Prozessor führt einen Eingabebefehl aus. Dabei wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Anschließend wird der Inhalt von HL um 1 erhöht und der Inhalt von B um 1 erniedrigt. Das Z-Bit wird 0, wenn $B - 1 \neq 0$ wird, und 1, wenn $B - 1 = 0$ wird.
Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

INDR



C	S	Z	P/V	H	N
	X	1	X	X	1

Adreßbus:

A_{15} bis A_8	A_7 bis A_0
Inhalt des B-Registers	Inhalt des C-Registers

Der Prozessor führt eine Reihe Eingabebefehle durch, deren Anzahl n im Register B steht. Während jedes Eingabezyklus wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Nach jedem Eingabezyklus erniedrigt sich der Inhalt von HL und B um 1. Insgesamt liest der Prozessor n Bytes ein, die in einen Speicherbereich gebracht werden, dessen Endadresse in HL steht.
Während eines Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-

Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

	C	S	Z	P/V	H	N
		X	↕	X	X	1

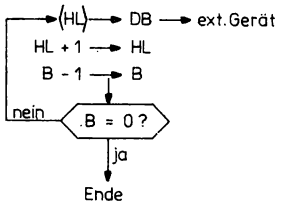
IND DB $\rightarrow \langle HL \rangle$
HL − 1 $\rightarrow$ HL
B − 1 $\rightarrow$ B

Adreßbus: A_{15} bis A_8 (Inhalt des B-Registers) A_7 bis A_0 (Inhalt des C-Registers)

Der Prozessor führt einen Eingabebefehl aus. Dabei wird der Datenbus DB abgetastet und sein Inhalt in eine Speicherzelle gebracht, deren Adresse in HL steht. Anschließend erniedrigt sich der Inhalt von HL um 1. Das Z-Bit wird 0, wenn B − 1 ≠ 0, und 1, wenn B − 1 = 0 ist.
Während des Eingabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

	C	S	Z	P/V	H	N
		X	1	X	X	1

OTIR



Adreßbus: A_{15} bis A_8 (Inhalt des B-Registers) A_7 bis A_0 (Inhalt des C-Registers)

Der Prozessor führt eine Reihe Ausgabebefehle durch, deren Anzahl n im Register B steht. Während jedes Ausgabezyklus wird der Inhalt der Speicherzelle, deren Adresse in HL steht, auf den Datenbus DB gebracht. Nach jedem Ausgabezyklus erhöht sich der Inhalt von HL um 1, und der Inhalt von B wird um 1 erniedrigt. Insgesamt gibt der Prozessor n Bytes aus, die in einem Speicherbereich stehen, dessen Anfangsadresse in HL steht.
Während eines Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

	C	S	Z	P/V	H	N
		X	↕	X	X	1

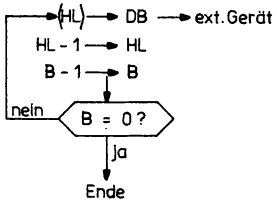
OUTI $\langle HL \rangle \rightarrow$ DB $\rightarrow$ ext. Gerät
HL + 1 $\rightarrow$ HL
B − 1 $\rightarrow$ B

Adreßbus: A_{15} bis A_8 (Inhalt des B-Registers) A_7 bis A_0 (Inhalt des C-Registers)

Der Prozessor führt einen Ausgabebefehl aus. Dabei wird der Inhalt der Speicherzelle, deren Adresse in HL steht, auf den Datenbus DB gebracht. Anschließend erhöht sich der Inhalt von HL um 1, und der Inhalt von B wird um 1 erniedrigt. Das Z-Bit wird 0, wenn B − 1 ≠ 0

ist, und 1, wenn B − 1 = 0 ist. Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

OTDR



C	S	Z	P/V	H	N
	X	1	X	X	1

Adreßbus:
A_{15} bis A_8 — Inhalt des B-Registers
A_7 bis A_0 — Inhalt des C-Registers

Der Prozessor führt eine Reihe Ausgabebefehle durch, deren Anzahl n im Register B steht. Während jedes Ausgabezyklus wird der Inhalt der Speicherzelle, deren Adresse in HL steht, auf den Datenbus gebracht. Nach jedem Ausgabezyklus erniedrigt sich der Inhalt von HL und B um 1. Insgesamt gibt der Prozessor n Bytes aus, die in einem Speicherbereich stehen, dessen Endadresse in HL steht.
Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

OUTD $\langle HL\rangle \rightarrow DB \rightarrow$ ext. Gerät
$HL - 1 \rightarrow HL$
$B - 1 \rightarrow B$

C	S	Z	P/V	H	N
	X	↕	X	X	1

Adreßbus: A_{15} bis A_8 — Inhalt des B-Registers
A_7 bis A_0 — Inhalt des C-Registers

Der Prozessor führt einen Ausgabebefehl aus. Dabei wird der Inhalt der Speicherzelle, deren Adresse in HL steht, auf den Datenbus DB gebracht. Anschließend erniedrigt sich der Inhalt von HL und B um 1. Das Z-Bit wird 0, wenn $B - 1 \neq 0$ ist, und 1, wenn $B - 1 = 0$ ist.
Während des Ausgabezyklus enthält der höherwertige Teil des Adreßbus den Inhalt des B-Registers und der niederwertige Teil den Inhalt des C-Registers, der als periphere Adresse dient.

2.4.9. Steuerbefehle

HALT
Der Befehl HALT bringt den Prozessor in den STOP-Zustand. Während dieses Zustandes führt der Prozessor NOP-Befehle aus, wobei er über den Adreßbus die Auffrischadresse für dynamische Speicher aussendet. Den HALT-Zustand kann der Baustein durch INTERRUPT oder RESET verlassen.

NOP
Während NOP führt der Prozessor einen Leerzyklus aus.

DI
0 → IFF1, 0 → IFF2
Durch den Befehl DI wird der maskierte INTERRUPT-Eingang gesperrt. Die beiden INTERRUPT-Flip-Flop IFF1 und IFF2 werden rückgesetzt.

EI
1 → IFF1, 1 → IFF2
Durch den Befehl EI wird der maskierte INTERRUPT-Eingang geöffnet. Die beiden INTERRUPT-Flip-Flop IFF1 und IFF2 werden gesetzt.

IM0
IM0 setzt den Prozessor in den INTERRUPT-MODE 0.

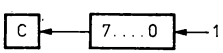
IM1
IM1 setzt den Prozessor in den INTERRUPT-MODE 1.

IM2
IM2 setzt den Prozessor in den INTERRUPT-MODE 2.

2.4.10. Befehlssatzerweiterung

Außer den bis jetzt genannten offiziellen Befehlen, die in den Assemblersprachen implementiert sind, gibt es noch eine Reihe zusätzlicher Befehle, die in den Assemblersprachen kaum vorkommen, jedoch als Maschinenbefehle abgearbeitet werden.
Diese Befehle beinhalten

1. Den Ladebefehl LD d, s
   wobei s eines der Register A, B, C, D, E, H, L, IXL, IXH, IYL, IYH oder ein Direktoperand von 8 Bit sein kann und d eines der genannten Register (IXL = Low-Teil von IX, IXH = High-Teil von IX, IYL = Low-Teil von IY, IYH = High-Teil von IY).
2. Die Befehle ADD, ADC, SUB, SBC, AND, OR, XOR, CP, INC, DEC mit IXL, IXH, IYL und IYH.
3. Den Befehl SLI s
   Dieser Befehl bewirkt eine arithmetische Linksverschiebung. Das aus Bit 7 heraustretende Bit wird in das C-Bit eingetragen. In Bit 0 wird jedoch eine 1 eingetragen.



s kann sein:
Register A, B, C, D, E, H, L
⟨HL⟩, d. h. eine Speicherzelle, deren Adresse in HL steht.
⟨IX + d⟩; ⟨IY + d⟩, d. h. eine Speicherzelle, deren Adresse durch Indexrechnung ermittelt wird.

2.5. INTERRUPT

2.5.1. INTERRUPT-Prinzip

Der Prozessor *U880* hat die INTERRUPT-Eingänge NMI (nicht maskierbar) und INT (maskierbar).
Maskierbar heißt, daß das Eingangstor durch das Programm geöffnet und gesperrt werden kann. Das Öffnen bewirkt ein spezieller Befehl EI. Das Sperren übernimmt der Befehl DI.
Zur Steuerung des INTERRUPT-Eingangstors INT gibt es im Prozessor die Flip-Flop IFF1 und IFF2.

- Mit IFF1 läßt sich das INT-Tor unmittelbar steuern. Bei eingeschaltetem IFF1 ist das Tor geöffnet; im ausgeschalteten Zustand ist das Tor gesperrt.
- IFF2 ist Zwischenspeicher für IFF1, wenn ein Signal über den NMI-Eingang kommt.

Nach Erscheinen des Signals am Eingang NMI oder am geöffneten Eingang INT wird der gerade laufende Befehl noch abgearbeitet. Anschließend durchläuft der Prozessor einen INTERRUPT-Zyklus.
Im INTERRUPT-Zyklus wird der Übergang zu einem Bedienprogramm für das betreffende INTERRUPT-Signal vorbereitet. Anschließend erfolgt der Übergang ins Bedienprogramm.

2.5.2. Ablauf

Jedes von außen kommende INT-Signal schaltet das IFF1, nachdem es das laufende Programm unterbrochen hat, zunächst aus. Damit erreicht man, daß das durch das INT-Signal aufgerufene Programm nicht wieder unterbrochen werden kann. Das IFF1 wird auch ausgeschaltet, wenn ein Unterbrechungssignal über den NMI-Eingang kommt. In diesem Fall wird aber vorher der Zustand des IFF1 auf das IFF2 übertragen.
Ist das zu NMI gehörige Programm abgearbeitet, wird durch den Rückkehrbefehl für den nichtmaskierten INTERRUPT RETN der Inhalt von IFF2 wieder nach IFF1 gebracht.
Den allgemeinen Ablauf zeigt Bild 2.13.
Der Übergang in das zum INTERRUPT gehörende Arbeitsprogramm ist bei den Eingängen NMI und INT unterschiedlich.

- Bei dem nichtmaskierbaren INTERRUPT (NMI) geschieht das durch den Befehl CALL 66H.
  Dieser Befehl wird im Prozessor automatisch gebildet und abgearbeitet.
- Bei dem maskierbaren INTERRUPT (INT) gibt es 3 Möglichkeiten des Übergangs in das dazugehörige Arbeitsprogramm. Die 3 Möglichkeiten werden mit MODE 0, MODE 1 und MODE 2 bezeichnet. Diese MODE lassen sich vorher durch die Befehle IM0, IM1 und IM2 einstellen.
- Im MODE 0 wird ein während des INTERRUPT-Zyklus eingelesenes 8-Bit-Wort als Befehl interpretiert und sofort abgearbeitet. Dafür verwendet man meistens den Befehl RST z. Er ist ein CALL-Befehl zu einer festen Adresse.
- Im Mode 1 wird ähnlich wie beim nichtmaskierbaren INTERRUPT ein Befehl CALL 38H gebildet und ausgeführt.
- Im MODE 2 wird aus dem I-Register als höherwertigem Teil und aus einem eingelesenen

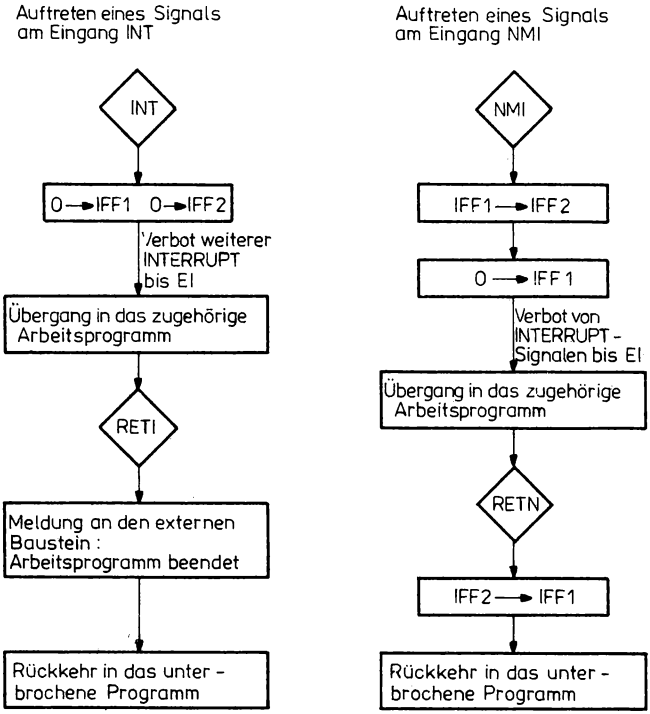


Bild 2.13
Ablauf bei Auftreten eines INTERRUPT-Signals. Mit dem RETURN-Befehl kann man aus dem zugehörigen Arbeitsprogramm in das unterbrochene Programm zurückkehren

8-Bit-Wort als niederwertigem Teil eine Adresse ADR gebildet. Das niederwertigste Bit des eingelesenen Bytes muß 0 sein. Das eingelesene 8-Bit-Wort nennt man INTERRUPT-Vektor.

ADR = | I-Register | INTERRUPT-Vektor |

Diese Adresse zeigt auf eine Zelle, deren Inhalt als Adresse eines CALL-Befehls verwendet wird. In MODE 2 wird also bei einem auftretenden INT-Signal der Befehl
CALL ⟨I-Register, INTERRUPT-Vektor⟩
gebildet und ausgeführt.
Das INTERRUPT-Bedienprogramm wird also im MODE 2 über eine INTERRUPT-Tabelle, in der die Adressen der Bedienprogramme stehen, gestartet (Bild 2.14).

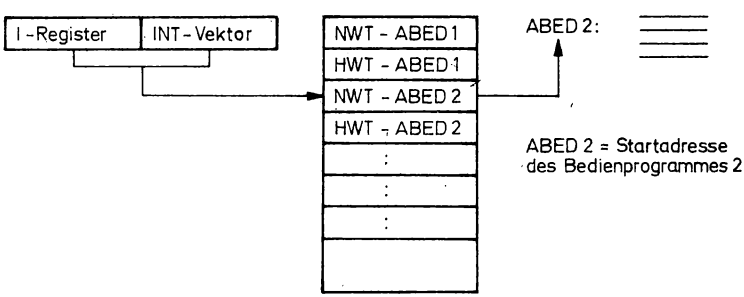


Bild 2.14
Aufbau der INTERRUPT-Tabelle für MODE 2

2.5.3. Handhabung der INTERRUPT-Eingänge

Der nichtmaskierte INTERRUPT wird für sehr wichtige Ereignisse verwendet. Er hat die höchste Priorität und unterbricht in jedem Fall das gerade laufende Programm. Zur Bedienung dieses INTERRUPT muß man ab Zelle 66H ein dazugehöriges Arbeitsprogramm speichern.
Der maskierte INTERRUPT-Eingang dient zum Aufbau umfangreicher Unterbrechungsschaltungen. An diese lassen sich Sammelschaltungen für eine große Anzahl Unterbrechungsquellen anschließen. Die Unterbrechungsquellen können eine Mehrebenenstruktur haben und nach Prioritäten gestaffelt sein.

2.5.4. Setzen und Löschen der INTERRUPT-Flip-Flop IFF1 und IFF2

IFF1 und IFF2 können durch Befehle, durch INTERRUPT oder durch RESET geändert werden. Tabelle 2.1. verdeutlicht die Möglichkeiten zum Ändern der INTERRUPT-Flip-Flop.

Tabelle 2.1.
Wirkung der Befehle und Signale auf IFF1 und IFF2

	IFF1	IFF2	
RESET	0 → IFF1	0 → IFF2	INTERRUPT-
DI	0 → IFF1	0 → IFF2	MODE 0
EI	1 → IFF1	1 → IFF2	
LD A, I	bleibt	bleibt	IFF2 → Parity-Flag
LD A, R	bleibt	bleibt	IFF2 → Parity-Flag
INT	0 → IFF1	0 → IFF2	
NMI	0 → IFF1	IFF1 → IFF2	
RETN	IFF2 → IFF1	bleibt	
RETI	IFF2 → IFF1	bleibt	

2.6. Starten des Prozessors

Die Programmabarbeitung des Prozessors läßt sich über die INTERRUPT-Möglichkeiten oder über RESET starten. Dabei gibt es folgende Startvarianten:

1. Ein Startimpuls kommt über die NMI-Leitung. In diesem Fall muß ab Zelle 66H das Startprogramm stehen.
2. Der Startimpuls kommt über die INT-Leitung. Dazu müssen der INT-Eingang vorher mit EI freigemacht, der notwendige INTERRUPT MODE eingestellt und das INTERRUPT-Wort auf dem Datenbus bereitgestellt werden.
3. Nach RESET beginnt die Abarbeitung im Prozessor mit Zelle 0. In Zelle 0 kann damit der 1. Befehl des Startprogramms stehen.

2.7. Anschlüsse des Bausteins (Bild 2.15)

A_0 bis A_{15} Adreß-Bus	Adreßbus, Tristate-Ausgang
D_0 bis D_7 Data-Bus	Datenbus, Tristate-Ein- und -Ausgang
$\overline{M1}$	Der laufende Zyklus ist ein Operationscodehol-Zyklus Ausgangssignal, aktiv »Low«
$\overline{MREQ}$ Memory Request	Am Adreßbus ist eine Speicheradresse vorhanden. Tristate-Ausgang, aktiv »Low«
$\overline{IORQ}$ Input-Output Request	Der niederwertige Teil des Adreßbus enthält eine Ein- oder Ausgabeadresse. Ein gleichzeitiges Erscheinen von $\overline{M_1}$ und $\overline{IORQ}$ kennzeichnet einen INTERRUPT-Zyklus Tristate-Ausgang, aktiv »Low«
$\overline{WR}$ write	Der Datenbus des Prozessors enthält Daten zur Ausgabe. Tristate-Ausgabe, aktiv »Low«
$\overline{RD}$ read	Vom Prozessor werden Daten über den Datenbus eingelesen, Tristate-Ausgang, aktiv »Low«
$\overline{RFSH}$ Refresh	Die 7 niederwertigen Bit des Adreßbus enthalten eine Auffrischadresse für angeschlossene dynamische Speicher. Sie kann in Verbindung mit Memory Request zum Auffrischen dieses Speichers verwendet werden. Ausgabesignal, aktiv »Low«
$\overline{HALT}$ Halt state	Der Prozessor hat einen HALT-Befehl ausgeführt und befindet sich im HALT-Status. Während des HALT-Status führt der Prozessor NOP-Operationen aus, während dieser NOP-Operationen finden Auffrischzyklen statt. Ausgabesignal, aktiv »Low«

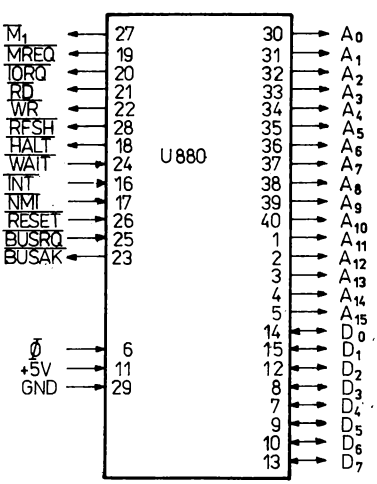
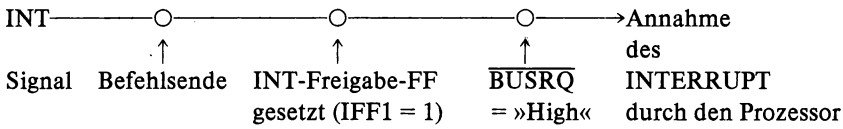


Bild 2.15
Anschlußbelegung des *U880*

$\overline{\text{WAIT}}$ Wait — Das WAIT-Signal veranlaßt den Prozessor, nach dem nächsten T2-Zustand in den Wartezustand zu gehen; solange WAIT aktiv ist, führt der Prozessor Wartezyklen durch.
Eingabesignal, aktiv »Low«

$\overline{\text{INT}}$ INTERRUPT Request — Anforderungssignal zu einer Programmunterbrechung im Prozessor. Es wird vom Prozessor am Ende eines Befehlszyklus angenommen, wenn das Flip-Flop IFF1 gesetzt (INTERRUPT-Erlaube-FF) und das Signal $\overline{\text{BUSRQ}}$ nicht aktiv ist.
Der Ablauf des INTERRUPT-Vorgangs ist im Abschnitt 4.1.6. beschrieben.
Eingabesignal, aktiv »Low«



$\overline{\text{NMI}}$ Non-Maskable INTERRUPT — Das Signal NMI verhält sich ähnlich wie das Signal INT. Im Unterschied zu INT ist es jedoch nicht maskierbar und hat eine höhere Priorität. Es wird am Ende eines Befehls angenommen, wenn das Signal $\overline{\text{BUSRQ}}$ nicht aktiv ist.
Eingabe mit negativer Flanke getriggert.

$\overline{\text{RESET}}$ — Dieses Signal setzt den Prozessor in den Grundzustand. Der Grundzustand ist gekennzeichnet durch:
$0 \rightarrow PC$ $0 \rightarrow I$
$0 \rightarrow IFF1$; $0 \rightarrow R$
$0 \rightarrow IFF2$; INTERRUPT MODE 0
Während das Signal RESET anliegt, sind Adreß- und Datenbus im hochohmigen Zustand und die Steuersignale inaktiv. Nach RESET beginnt die Abarbeitung mit Zelle 0.
Eingabesignal, aktiv »Low«

$\overline{\text{BUSRQ}}$ — Dieses Signal bringt den Adreßbus, den Datenbus und die Steuersignale in den hochohmigen Zustand. Es wird am Ende jedes Maschinenzyklus abgefragt.
Eingabesignal, aktiv »Low«

$\overline{\text{BUSAK}}$ Bus Acknowledge — Datenbus, Adreßbus und Steuerbus befinden sich im hochohmigen Zustand. Die CPU arbeitet nicht. Es erfolgt kein REFRESH.
Ausgabesignal, aktiv »Low«

Φ Steuertakt — Der Steuertakt kann über einen 330-Ω-Widerstand nach 5 V mit dem Ausgang eines TTL-Schaltkreises verbunden sein (Bild 2.16).

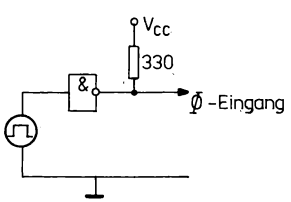


Bild 2.16
Takteingang des *U880*

3. Der Mikroprozessorbaustein *8080*

Der Mikroprozessor *8080*, ein stark verbreiteter Baustein, wird in sehr vielen Mikrorechner-Konfigurationen verwendet. Alle in ihm vorhandenen Möglichkeiten sind auch im Mikroprozessor *U880* enthalten.

3.1. Registerstruktur

Bild 3.1 zeigt die Registerstruktur des Bausteins *8080*. Die Register A, B, C, D, E, H, L, SP, PC, BR haben die gleiche Funktion wie beim Prozessor *U880*. Das gleiche gilt für die Flags, C, S, Z, P, H. Das P-Flag wird jedoch nur in Abhängigkeit von der Parität des Ergebnisses gesetzt.

3.2. Zeitverhalten

Der Mikroprozessor *8080* wird durch 2 gegeneinander versetzte Impulsfolgen Φ_1 und Φ_2 gesteuert. Die Abarbeitung eines Befehls erfolgt in mehreren Maschinenzyklen. Ein Maschi-

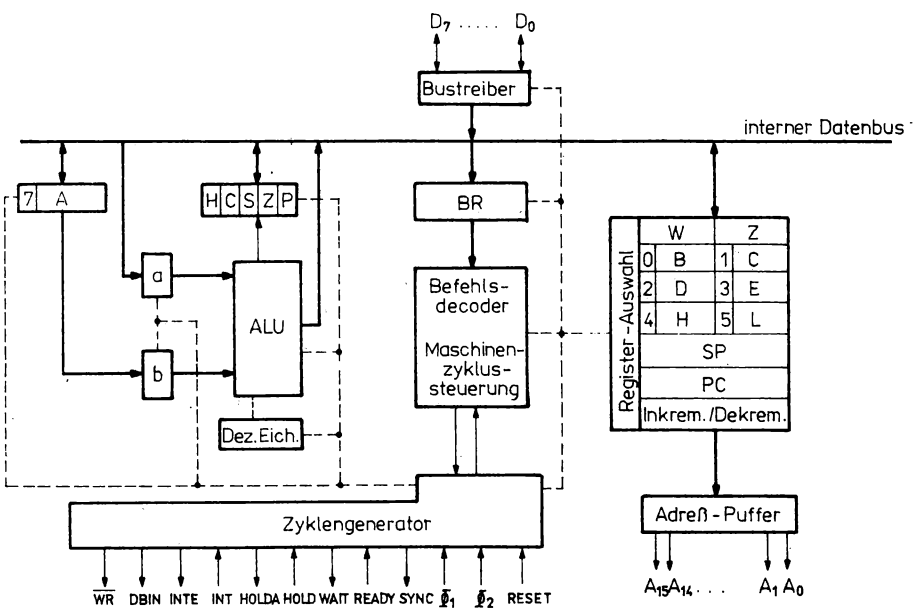


Bild 3.1
Registerstruktur des *8080*

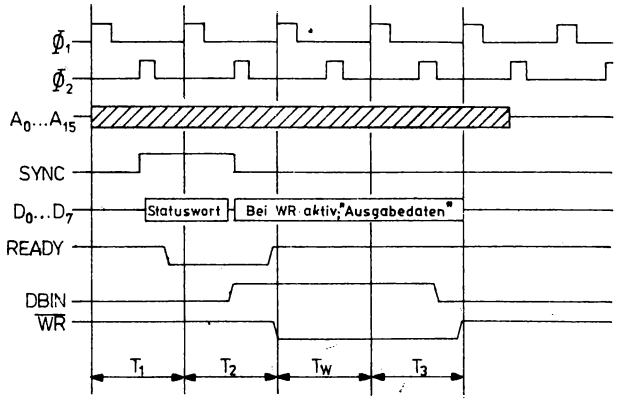


Bild 3.2
Taktdiagramm eines Maschinenzyklus des *8080*;

T_1	Ausgabe der Adresse auf den Adreßbus
T_1/T_2	Ausgabe des Statusworts auf den Datenbus
T_2	Abtasten des Signals »READY«
T_3	Ausgabe oder Eingabe über den Datenbus
T_4/T_5	Ausführung des Befehls

nenzyklus ist wiederum in 3 bis 5 Zeitzustände (T_1 bis T_5) unterteilt, wobei ein Zeitzustand die Länge von einer Periode Φ_1 hat.
Die wichtigsten Funktionen in den einzelnen Zeitzuständen innerhalb eines Maschinenzyklus sind in Bild 3.2 dargestellt.
Nach T_2 können beliebig viele Wartezustände T_W eingefügt werden. Zu Beginn jedes Maschinenzyklus sendet der Prozessor über den Datenbus ein Statuswort aus. Das Signal SYNC zeigt an, daß das Statuswort auf dem Datenbus D_0 bis D_7 vorhanden ist. Ist zum Zeitpunkt T_2/Φ_2 READY = »Low«, so kommt nach T_2 ein Wartezustand T_W. Ein weiterer Wartezustand folgt, wenn zum Zeitpunkt T_W/Φ_2 READY noch »Low« ist. Bei DBIN = »High« (aktiv) wird zum Zeitpunkt T_3/Φ_1 die Information vom Datenbus in den Prozessor geholt. Ist $\overline{WR}$ = »Low« (aktiv), so enthält der Datenbus eine vom Prozessor ausgegebene Information (DBIN und $\overline{WR}$ sind nie gleichzeitig aktiv). Zur Ausführung eines Befehls werden mehrere Maschinenzyklen durchlaufen. Im 1. Maschinenzyklus M1 wird der Befehl geholt (Befehlsholzyklus), entschlüsselt und, wenn keine Daten von außerhalb des Prozessors benötigt werden, während T_4 und T_5 ausgeführt. Benötigt man Daten von außerhalb eines Prozessors, so folgen weitere Maschinenzyklen (Bild 3.3).

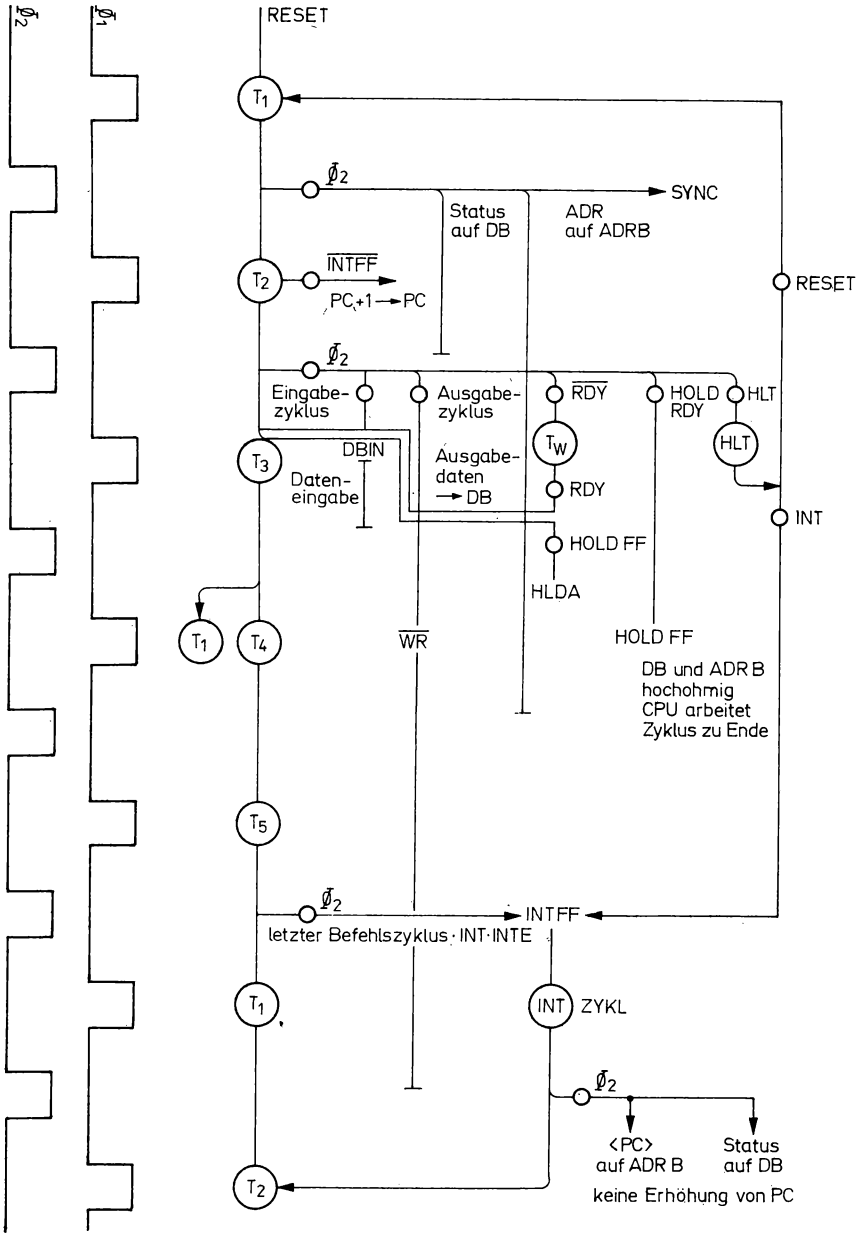


Bild 3.3
Ablauf eines Maschinenzyklus im *8080* (DB – Datenbus, ADR – Adresse; ADRB – Adreßbus; INTFF – INTERRUPT-Freigabe Flip-Flop; INTZYKL – INTERRUPT-Zyklus; TW – Wartezyklus; HOLDFF – Busfreigabe Flip-Flop; SYNC, WR, HLDA, DBIN, INT, INTE, INTA – Bussignale)

Beispiel
Eingabe eines Bytes nach dem A-Register.

Befehl: IN ADR Codierung im Speicher: 11011011 / ADR-Byte

Maschinenzyklus	Zustand	Funktion
M1	T_1	Ausgabe PC
		Ausgabe Status
	T_2	PC + 1 → PC
		Abfrage »READY«
	T_3	Befehl → BR
	T_4	interne Entschlüsselung
M2	T_1	Ausgabe PC
		Ausgabe Status
	T_2	PC + 1 → PC
		Abfrage »READY«
	T_3	ADR → Register Z und W
M3	T_1	Ausgabe Register Z und W (ist Geräteadresse)
	T_2	Abfrage READY
	T_3	Eingabe Byte vom DB → Register A

- Jeder Zustand wird durch Φ_1 eingeleitet.
- Das Signal SYNC wird während T_1 ausgegeben. Φ_2 bewirkt das Einschalten, und der nächste Φ_2 schaltet aus.
- Die Eingabedaten müssen an D_0 bis D_7 zwischen Φ_1/T_3 und Φ_2/T_3 anliegen.
- Die Adressen werden mit Φ_2/T_1 eingeschaltet und mit Φ_2 nach T_3 ausgeschaltet.

Wenn vor Φ_2/T_2 READY = 0 wird, geht die CPU in Wartezyklen. READY muß vor der Φ_2-Rückflanke wieder aktiv werden, wenn keine weiteren Wartezyklen folgen sollen.

3.3. Anschlußsignale (Bild 3.4 und Bild 3.5)

D_0 bis D_7	Bidirektionaler Datenbus, wird für die Ein- und Ausgabe von Datensignalen verwendet.
A_0 bis A_{15}	Adreßbus
SYNC	Ausgabesignal (aktiv »High«) sagt aus, daß der Datenbus das Statuswort enthält.
DBIN	Ausgabesignal (aktiv »High«) sagt aus, daß zum Zustand T_3 eine Eingabe vom Datenbus erfolgt.
READY	Eingabesignal (aktiv »High«) Ist zum Zeitpunkt T_2/Φ_2 oder T_W/Φ_2 READY = »High«, so folgt der Zustand T_3, bei READY = »Low« folgt ein Wartezustand T_W.
WAIT	Dieses Ausgabesignal (aktiv »High«) sagt aus, daß sich der Prozessor im Wartezustand T_W befindet.
$\overline{WR}$	Das Ausgabesignal (aktiv »Low«) sagt aus, daß sich auf dem Datenbus ein ausgegebenes Byte befindet.

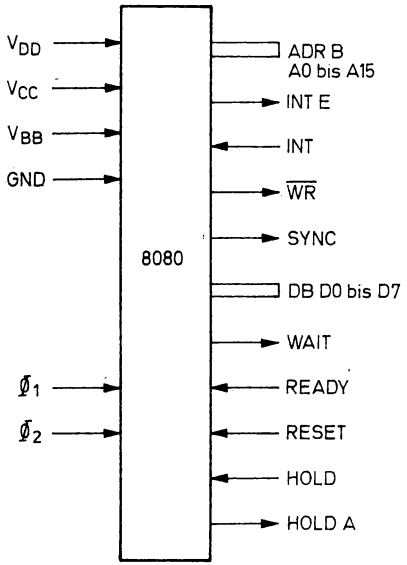


Bild 3.4
Anschlußsignale des *8080* ($V_{DD} = 12\,V \pm 5\,\%$, $V_{CC} = 5\,V \pm 5\,\%$, $V_{BB} = 5\,V \pm 5\,\%$, $V_{SS} = 0\,V$, $I_{DDmax} = 70\,mA$, $I_{CCmax} = 80\,mA$, $I_{BBmax} = 1\,mA$)

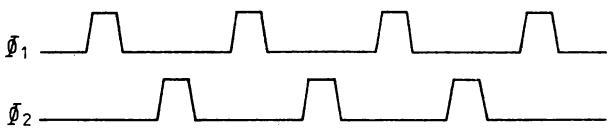


Bild 3.5
Grundtaktsignale Φ1 und Φ2 des *8080*

HOLD — Eingabesignal (aktiv »High«)
DMA-Anforderungssignal. Durch HOLD geht der Daten- und Adreßbus in den hochohmigen Zustand (HOLD-Zustand). Das Signal HOLD wird angenommen:
- im Zustand T_2 oder T_W;
- im HALT-Zustand.

Der HOLD-Zustand bleibt so lange bestehen, wie HOLD = »High« ist. Nachdem HOLD = »Low« wird, beginnt ein neuer Maschinenzyklus mit T_1.

HLDA — Ausgabesignal (aktiv »High«) sagt aus, daß sich der Prozessor im HOLD-Zustand befindet, d. h. Daten- und Adreßbus hochohmig sind.

INTE — Ausgabesignal (aktiv »High«) sagt aus, daß der INT-Eingang frei ist. Es wird rückgesetzt durch den Befehl DI oder bei Annahme eines INT-Signals.

INT — Eingabesignal (aktiv »High«)
Programmunterbrechungsanforderung
INT wird angenommen:
- am Ende eines Befehls;
- im HALT-Zustand;
- bei INTE = »High«.

Die Programmunterbrechung entspricht dem INTERRUPT-MODE 0 des Bausteins *U880*. Bei Annahme der Programmunterbrechung wird ein INT-

Zyklus durchlaufen. Im INT-Zyklus wird
- im Statuswort das Signal INTA ausgesandt;
- im Zustand T_3 ein 1-Byte-Befehl vom Datenbus eingelesen und sofort ausgeführt. Dazu verwendet man in der Regel den Befehl RST.

RESET Eingabesignal (aktiv »High«)
Durch RESET wird der Baustein *8080* in den Grundzustand versetzt. Der Grundzustand ist gekennzeichnet durch:
- PC gelöscht;
- BR gelöscht;
- INTE = »Low«;
- HLTA = »Low«.

Die Register A, B, C, D, E, H, L bleiben erhalten.

Statusinformationen
Zusätzlich zu den Steuersignalen, die direkt vom Prozessor *8080* ausgehen, gibt der Baustein *8080* am Anfang jedes Maschinenzyklus gleichzeitig mit dem Signal SYNC ein Statuswort über den Datenbus aus. Bild 3.6 zeigt den Aufbau des Statuswortes.

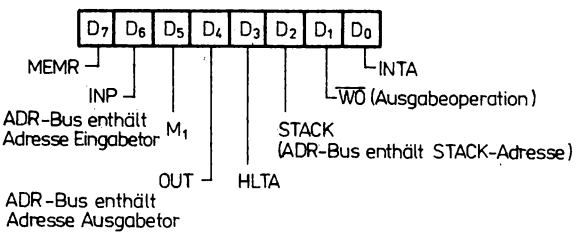


Bild 3.6
Statuswort des *8080*

3.4. Anschluß von Schaltkreisen

Mit Hilfe der Status- und Prozessor-Ein- und -Ausgabesignale werden die am Prozessor angeschlossenen Bausteine gesteuert. Bild 3.7 zeigt die Steuerung des Bausteins *8080* und die Bildung von Adreßbus, Datenbus und Steuerbus zur Realisierung von Mikrorechneraufbauten.
Darin bedeuten (s. Taktdiagramm Bild 3.8):
A_0 bis A_{15} Adreßbus
D_0 bis D_7 Datenbus
DBIN, WR *8080*-Signale

INA	DBIN · INTA	Eingabesignal für INTERRUPT-Befehl
IOR	= DBIN · INP	Eingabesignal für Eingabekanal
IOW	= WR · OUT	Ausgabesignal für Ausgangskanal
MER	= DBIN · MEMR	Speicherlesesignal
STR	$= \Phi_{1A}$ · SYNC	Statusübernahmesignal
		Φ_{1A} wird aus Φ_2 gebildet (etwas verzögertes Φ_2)

3.5. Befehlsschlüssel

3.5.1. Übersicht

Ein *8080* Befehl besteht aus einem Byte für den Operationscode und 0 bis 2 Byte für die Operanden.
Alle Befehle des *8080* sind auch im *U880* enthalten. Die Befehle des *8080* sind eine Teilmenge der Befehle des *U880.*
Die Befehlsmnemonik (Befehlsschreibweise in der Assemblersprache) ist jedoch anders.

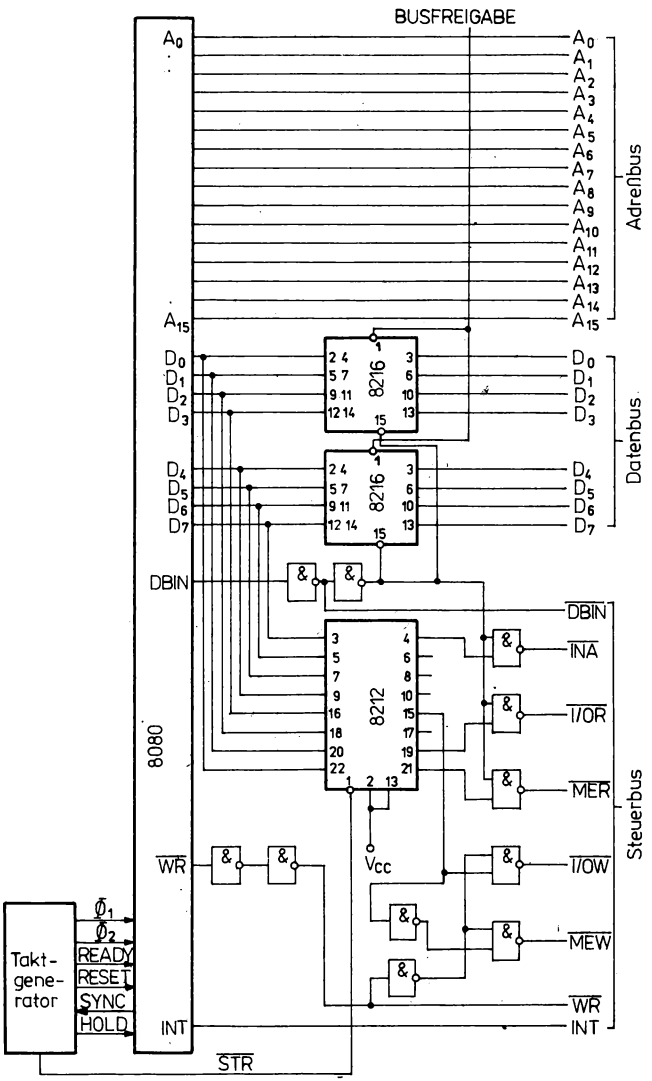


Bild 3.7
Bildung des Mikrorechnerbus mit dem *8080*

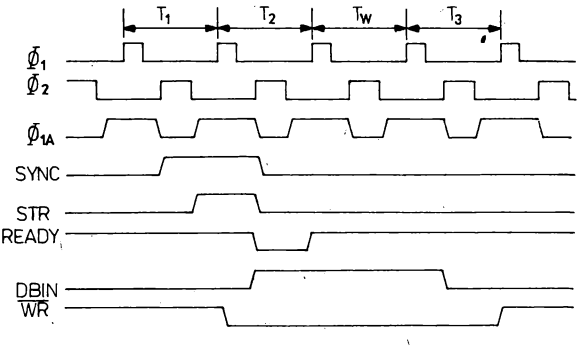


Bild 3.8
Taktdiagramm zur Bildung des Mikrorechnerbus mit dem *8080*

3.5.2. Befehlsliste

3.5.2.1. Transportbefehle

8-Bit-Transfer

C S Z P H

MOV d, s s → d

Der Inhalt von s kommt nach d.
s und d kann sein: Register A, B, C, D, E, H, L, M.
M ist eine Speicherzelle mit Adresse in HL.
Ausnahme: s und d gleichzeitig M

C S Z P H

MVI d, n n → d

Die Zahl n kommt nach d.
d kann sein: Register A, B, C, D, E, H, L, M.
M ist eine Speicherzelle mit Adresse in HL.

C S Z P H

STAX (RP) A → (RP)

Der Inhalt von Register A wird in die Speicherzelle gespeichert, deren Adresse im Registerpaar RP steht.
RP kann sein: BC oder DE

C S Z P H

LDAX (RP) (RP) → A

Der Inhalt der Speicherzelle, deren Adresse im Registerpaar RP steht, kommt nach Register A.
RP kann sein: BC oder DE

C S Z P H

STA nn A → (nn)

Der Inhalt von Register A kommt nach Speicherzelle nn.

C S Z P H

LDA nn (nn) → A

Der Inhalt der Zelle nn kommt nach Register A.

16-Bit-Transfer

C S Z P H

LXI RP, nn nn → RP

Die Zahl nn kommt nach Registerpaar RP.
RP kann sein: BC, DE, HL, SP

C S Z P H

LHLD nn (nn) → HL

Der Inhalt von Zelle nn und nn + 1 kommt nach HL.

C S Z P H

XCHG DE ↔ HL

Der Inhalt von DE wird mit dem Inhalt von HL vertauscht.

C S Z P H

PUSH dd dd → STACK

Der Inhalt des Registerpaares dd kommt in den STACK.
dd kann sein: BC, DE, HL, PSW
Bei PSW ist Register A der höherwertige Teil und PSW der niederwertige Teil.

C S Z P H

POP ss STACK → ss . . .
(außer PSW)

Registerpaar ss wird vom STACK gefüllt.
ss kann sein: BC, DE, HL, PSW
Bei PSW ist Register A der höherwertige Teil und PSW der niederwertige Teil.

C S Z P H

SPHL HL → SP

Der Inhalt von HL kommt nach SP.

3.5.2.2. Rechen- und logische Operationen mit einem Operanden

8-Bit-Befehle

CY S Z P H

CMA $\overline{A} \rightarrow A$

Der Inhalt von Register A wird bitweise negiert.

CMC $\overline{CY} \rightarrow CY$

CY	S	Z	P	H
↕	·			

Das Carry-Flag wird negiert.

STC $1 \rightarrow CY$

CY	S	Z	P	H
↕				

Das Carry-Flag wird gesetzt.

DAA BCD (A) $\rightarrow$ A

CY	S	Z	P	H
↕	↕	↕	↕	↕

Eine durch duale Addition von 2 BCD-Zahlen im Register A entstandene Zahl wird in die richtige BCD-Form gebracht. Eine Addition von 2 8-Bit BCD-Zahlen besteht daher aus einer dualen Addition und anschließendem DAA.

INR d $d + 1 \rightarrow d$

CY	S	Z	P	H
	↕	↕	↕	↕

Der Inhalt des Registers oder der Zahl d wird um 1 erhöht.
d kann sein: A, B, C, D, E, H, L, M
M ist eine Speicherzelle mit Adresse in HL.

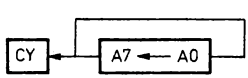
DCR d $d - 1 \rightarrow d$

CY	S	Z	P	H
	↕	↕	↕	↕

Der Inhalt des Registers oder der Speicherzelle d wird um 1 erniedrigt.
d kann sein: A, B, C, D, E, H, L, M
M ist eine Speicherzelle mit Adresse in HL.

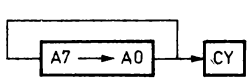
Verschiebebefehle

Alle Verschiebebefehle werden nur mit Register A ausgeführt. Bei den Flags wird dabei nur das CY-Flag geändert.

RLC 

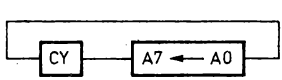
CY	S	Z	P	H
↕				

Der Inhalt des Registers A wird um eine Stelle nach links verschoben. Bit 7 wird ins CY und in Bit 0 eingetragen.

RRC 

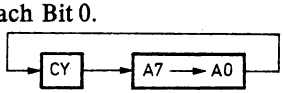
CY	S	Z	P	H
↕				

Der Inhalt des Registers A wird um eine Stelle nach rechts verschoben. Bit 0 wird ins CY und in Bit 7 eingetragen.

RAL 

CY	S	Z	P	H
↕				

Der Inhalt des Registers A wird mit CY zyklisch nach links verschoben. Bit 7 kommt ins CY-Flag und das CY-Flag nach Bit 0.

RAR 

Der Inhalt des Registers A wird mit CY zyklisch nach rechts verschoben. Bit 0 kommt ins CY-Flag und das CY-Flag nach Bit 7.

16-Bit-Befehle

			CY	S	Z	P	H
INX	RP	RP + 1 → RP					

Der Inhalt des Registerpaars RP wird um 1 erhöht.
RP kann sein: BC, DE, HL, SP

			CY	S	Z	P	H
DCX	RP	RP − 1 → RP					

Der Inhalt des Registerpaars RP wird um 1 erniedrigt.
RP kann sein: BC, DE, HL, SP

3.5.2.3. Rechen- und logische Operationen mit 2 Operanden

8-Bit-Befehle

			CY	S	Z	P	H
ADD	s	A + s → A	↕	↕	↕	↕	↕
ADI	n	A + n → A					

Der Inhalt des Akkumulators und der Inhalt des Registers oder der Zelle s werden addiert, und das Ergebnis wird ins Register A gebracht. Bei ADI tritt statt s der Direktoperand n.
s kann sein: A, B, C, D, E, H, L, M
M ist eine Speicherzelle mit Adresse in HL.

			CY	S	Z	P	H
ADC	s	A + s + CY → A	↕	↕	↕	↕	
ACI	n	A + n + C					

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden addiert. Zu diesem Ergebnis wird das CY-Bit in die niedrigste Stelle addiert. Das Gesamtergebnis kommt in das Register A. Bei ACI tritt statt s der Direktoperand n.
s kann sein: siehe 1-Wort-Befehl ADD s.

			CY	S	Z	P	H
SUB	s	A − s → A	↕	↕	↕	↕	↕
SUI	n	A − n → A					

Der Inhalt der Zelle oder des Registers s wird vom Register A subtrahiert. Das Ergebnis kommt in das Register A. Bei SUI tritt statt s der Direktoperand n auf.
s kann sein: siehe 1-Wort-Befehl ADD s.

			CY	S	Z	P	H
SBB	s	A − s − CY → A	↕	↕	↕	↕	↕
SBI	n	A − n − CY → A					

Der Inhalt des Registers oder der Zelle s wird vom Inhalt des A-Registers subtrahiert. Von der niedrigsten Stelle des Ergebnisses wird das C-Bit subtrahiert. Das letzte Ergebnis kommt in das Register A. Bei SBI tritt statt s der Direktoperand n.
s kann sein: siehe 1-Wort-Befehl ADD s.

			CY	S	Z	P	H
ANA	s	$A \wedge s \rightarrow A$		↕	↕	↕	↕
ANI	n	$A \wedge n \rightarrow A$					

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden bitweise durch ein logisches UND verknüpft. Das Ergebnis kommt in das Register A. Bei ANI tritt statt s der Direktoperand n.
s kann sein: siehe 1-Wort-Befehl ADD s.

			CY	S	Z	P	H
ORA	s	$A \vee s \rightarrow A$		↕	↕	↕	↕
ORI	n	$A \vee n \rightarrow A$					

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden bitweise durch ein logisches ODER verknüpft, das Ergebnis kommt in das Register A. Bei ORI tritt statt s der Direktoperand n.
s kann sein: siehe 1-Wort-Befehl ADD s.

			CY	S	Z	P	H
XRA	s	$A \oplus s \rightarrow A$		↕	↕	↕	↕
XRI	n	$A \oplus n \rightarrow A$					

Der Inhalt des Registers A und der Inhalt des Registers oder der Zelle s werden bitweise durch ein logisches EXKLUSIV-ODER verknüpft. Das Ergebnis kommt in das Register A. Bei XRI tritt statt s der Direktoperand n.
s kann sein: siehe 8-Bit-Befehl ADD s.

		CY	S	Z	P	H
CMP	s	↕	↕	↕	↕	↕
CPI	n					

Der Inhalt des Registers A wird mit dem Inhalt des Registers oder der Zelle s verglichen. Der Vergleich erfolgt durch die Bildung der Differenz $A - s$. Bei Gleichheit wird das Z-Bit gesetzt, bei Ungleichheit zurückgesetzt. Bei CPI tritt statt s der Direktoperand n.
s kann sein: siehe 1-Wort-Befehl ADD s.

16-Bit-Befehle

			CY	S	Z	P	H
DAD	RP	$HL + RP \rightarrow HL$	↕				

Der Inhalt des Registerpaares RP wird zu HL addiert.
RP kann sein: BC, DE, HL, SP.

3.5.2.4. Sprungbefehle

Unbedingte Sprünge

CY S Z P H

JMP nn nn → PC
Sprung nach Zelle nn

CY S Z P H

PCHL HL → PC

Sprung in die Zelle, deren Adresse in HL steht.
Bedingte Sprünge
JNZ nn: Sprung nach nn, wenn Z-Flag = 0
JZ nn: Sprung nach nn, wenn Z-Flag = 1
JNC nn: Sprung nach nn, wenn CY-Flag = 0
JC nn: Sprung nach nn, wenn CY-Flag = 1
JPO nn: Sprung nach nn, wenn P-Flag = 0 (Parity odd)
JPE nn: Sprung nach nn, wenn P-Flag = 1 (Parity even)
JP nn: Sprung nach nn, wenn S-Flag = 0 (positiv)
JN nn: Sprung nach nn, wenn S-Flag = 1 (negativ)

3.5.2.5. Unterprogrammbefehle

CY S Z P H

CALL nn PC → Stack
nn → PC
Sprung in ein Unterprogramm, dessen Startadresse nn ist. Der alte PC-Stand kommt in den STACK.
Bedingte Unterprogrammsprünge

CNZ nn CALL nn, wenn Z-Flag = 0
CZ nn CALL nn, wenn Z-Flag = 1
CNC nn CALL nn, wenn C-Flag = 0
CC nn CALL nn, wenn C-Flag = 1
CPO nn CALL nn, wenn P-Flag = 0
CPE nn CALL nn, wenn P-Flag = 1
CP nn CALL nn, wenn S-Flag = 0
CM nn CALL nn, wenn S-Flag = 1
Rücksprung aus dem Unterprogramm

CY S Z P H

RET STACK → PC

Rücksprung aus einem Unterprogramm nach der Absprungstelle im Hauptprogramm.

Bedingte Unterprogrammrücksprünge
RNZ RET-Befehl, wenn Z-Flag = 0
RZ RET-Befehl, wenn Z-Flag = 1

RNC RET-Befehl, wenn C-Flag = 0
RC RET-Befehl, wenn C-Flag = 1
RPO RET-Befehl, wenn P-Flag = 0
RPE RET-Befehl, wenn P-Flag = 1
RP RET-Befehl, wenn S-Flag = 0
RM RET-Befehl, wenn S-Flag = 1

3.5.2.6. Ein-/Ausgabebefehle

CY S Z P H

IN n (n) → A
Eingabe eines 8-Bit-Worts von der Toradresse n nach Register A.
OUT n A → (n)
Ausgabe Inhalt Register A zu einer Toradresse n.

3.5.2.7. Steuerbefehle

NOP Leerbefehl
DI Sperren des INTERRUPT-Eingangs (disable interrupt)
EI Freigabe des INTERRUPT-Eingangs (enable interrupt)

3.6. Programmunterbrechung

Der Baustein *8080* hat einen Programmunterbrechungseingang. Der Ablauf der Programmunterbrechung ist identisch mit dem Ablauf der Programmunterbrechung im Prozessor *U880* im MODE 0. Der einzige Unterschied besteht nach dem Einschalten der Spannungen bzw. nach RESET. Im Baustein *U880* ist nach Einschalten der Spannung und nach RESET der maskierte Eingang gesperrt. Ein INTERRUPT kann nur über den nichtmaskierten Eingang kommen. Nach RESET startet der Prozessor *U880* automatisch bei Zelle 0.
Im Baustein *8080* ist nach Einschalten der Spannungen und nach RESET der INTERRUPT-Eingang offen. Der Prozessor *8080* wird über INTERRUPT gestartet.

4. Periphere Schaltkreise zum *U880*

4.1. Grundfunktion peripherer Schaltkreise

Der Mikroprozessor steuert mit seinen Anschlußsignalen einen Mikrorechnerbus. Dieser besteht aus 16 Adreßleitungen, 8 Datenleitungen, den Leitungen für die Steuersignale (M1, MREQ, IORQ, RD, WR, RFSH, WAIT, INT, NMI, HALT, RESET, BUSRQ, BUSAK) und zusätzlichen Versorgungsleitungen.
Um periphere Geräte an den Bus anzuschließen, müssen die Bussignale ausgewertet und zur Steuerung der Geräte aufbereitet werden.
Dazu ist ein Zwischenglied zwischen dem Bus und dem Gerät erforderlich. Dieses Zwischenglied nennt man Controller oder auch Interface. Der Begriff Interface ist nicht ganz korrekt, da dieser eigentlich nur eine Schnittstelle, d. h., welche Signale anliegen, charakterisiert. Trotzdem wird der Begriff oft für den Controller verwendet.
Ein- und Ausgabe-Bausteine bilden das Interface zum Anschluß von peripheren Geräten an den Rechnerbus.
Bild 4.1 zeigt das Prinzip des Anschlusses von einem Bedienpult, eines Lochbandlesers, eines Magnetbandgerätes und eines Lochbandstanzers an den Prozessor *U880*. Auf die gleiche Weise lassen sich auch AD-Wandler, DA-Wandler, Drucker sowie Bildschirmeinheiten steuern.
Der Ein-/Ausgabe-Baustein erzeugt aus den Signalen des Rechnerbusses die Signale, die zur Steuerung peripherer Geräte erforderlich sind. Das periphere Gerät sendet Fertigmeldung und Fehlermeldungen dem E/A-Baustein, in dem diese Meldungen gesammelt und an den Rechnerbus weitergegeben werden. Die CPU kann solche Meldungen am peripheren Baustein abfragen (Polling). Eine zweite Möglichkeit besteht in der Bildung einer INTERRUPT-Anforderung an die CPU. Zur INTERRUPT-Bildung können die E/A-Bausteine der *U880*-Familie INTERRUPT-Ketten bilden.

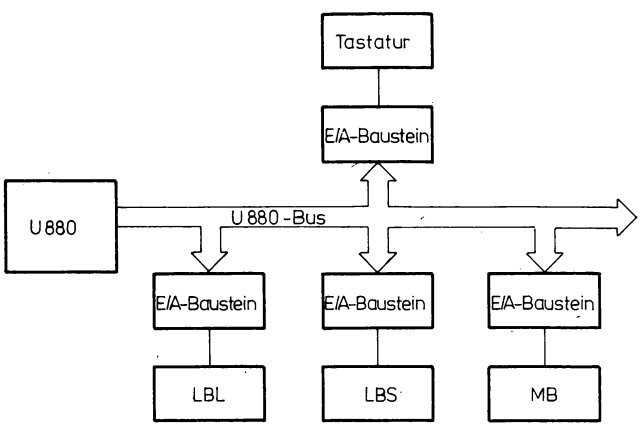


Bild 4.1
Anschluß von Bedienpult, Lochbandleser, Lochbandstanzer und Magnetbandgerät an den *U880*

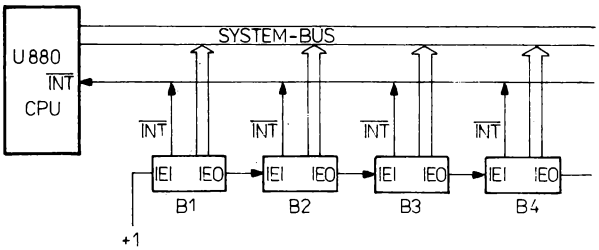


Bild 4.2
Zusammenschaltung von peripheren Bausteinen B1 bis B4 zu einer INTERRUPT-Prioritätenkette

Struktur einer INTERRUPT-Kette

Jeder E/A-Baustein hat die Anschlüsse IEI, IEO, $\overline{\text{INT}}$. Wenn IEI (interrupt enable input) H-Pegel führt, so kann der Baustein ein INTERRUPT-Signal bilden. IEO (interrupt enable output) ist »High«, wenn IEI auf H-Pegel liegt und keine INTERRUPT-Anforderung vom Baustein selbst kommt. Mit den beiden Signalen IEI und IEO lassen sich die Bausteine in einer Kette nach Bild 4.2 zusammenstellen. Der dem Prozessor am nächsten liegende Baustein hat die höchste Priorität.

Der folgende INTERRUPT-Ablauf (bezogen auf Bild 4.2) zeigt die Arbeitsweise der einzelnen Bausteine, damit die INTERRUPT-Kette alle Prioritätsfunktionen erfüllt.

- Baustein B3 bildet eine INTERRUPT-Anforderung (die CPU ist in der Betriebsart IM2).
- INT wird aktiv (INT-Leitung = »Low«).
- IEO von Baustein B3 wird »Low«. Damit werden INTERRUPT-Anforderungen von Baustein B4 und den weiter rechts liegenden Bausteinen gesperrt.
- Ist der INTERRUPT-Eingang der CPU offen, erfolgt INTERRUPT-Annahme. Die CPU führt einen INTERRUPT-Zyklus (mit IORQ und M1) aus.
- Während des INTERRUPT-Zyklus kann Baustein B3 (wenn nicht vorher B1 oder B2 eine INTERRUPT-Anforderung gebildet haben) seinen INTERRUPT-Vektor zur CPU geben. (Sonst gibt der Baustein seinen INTERRUPT-Vektor, der eine INTERRUPT-Anforderung gestellt hat und der CPU am nächsten liegt.)
- Mit der Aufnahme des INTERRUPT-Vektors durch die CPU erfolgen der Sprung in die zugehörige Behandlungsroutine und das Sperren des INTERRUPT-Eingangs der CPU.
- Am Ende der Behandlungsroutine kommen im allgemeinen die Befehle EI und RETI. Durch EI wird der CPU-INTERRUPT-Eingang wieder frei. RETI wird vom Baustein B3 erkannt. B3 setzt nun IEO wieder auf »High« und gibt damit den Rest der Kette frei.

- Ein besonderer Fall tritt dann ein, wenn während der Behandlungsroutine z. B. B2 eine INTERRUPT-Anforderung stellt; IEO von B2 wird »Low« und B3 wäre gesperrt. Damit B3 den Abschluß RETI seiner Routine erkennt, setzt B2 mit dem 1. Byte von RETI (RETI hat die Codierung ED 4D) sein IEO auf »High« und B3 kann beide Bytes von RETI entschlüsseln.

4.2. Paralleler Ein-/Ausgabe-Baustein *U855D*

4.2.1. Struktur

Bild 4.3 zeigt die Grundstruktur des PIO-Bausteins. Der PIO-Baustein hat die Aufgabe, das Rechnerinterface (Rechnerbus) in ein Interface zur Steuerung peripherer Geräte mit paralle-

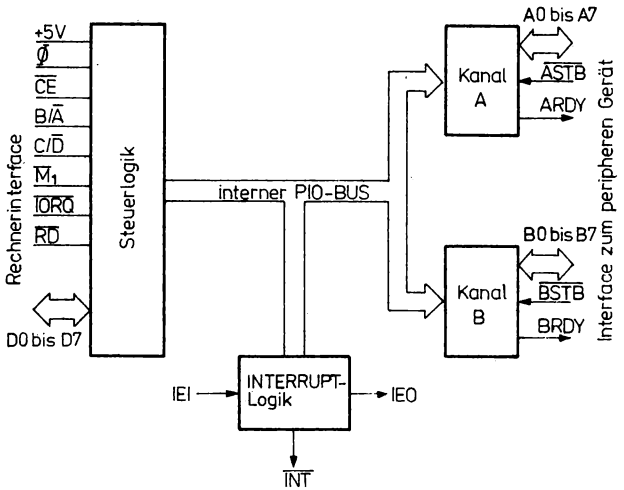


Bild 4.3
Grundstruktur des Bausteins *U855D*

ler Dateneingabe bzw. -ausgabe umzusetzen. Der Baustein verfügt über eine interne Steuerlogik, die mit Hilfe des Grundtaktes Φ arbeitet, sowie eine INTERRUPT-Steuerung mit der Möglichkeit zur Bildung von Prioritätsketten. Die Steuerlogik erzeugt aus den Signalen des Rechnerbusses die zur Peripheriesteuerung notwendigen Signale. Zur Peripheriesteuerung gibt es 2 Ein-/Ausgabe-Kanäle A und B, die im *Hand-shake*-Verfahren den Transfer zu einem peripheren Gerät realisieren. Jeder Kanal hat

- 1 Eingaberegister (8 Bit),
- 1 Ausgaberegister (8 Bit),
- 1 Selektregister (Auswahlregister; 8 Bit),
- 1 Maskenregister (8 Bit),
- 1 Maskensteuerungsregister (2 Bit),
- 1 Betriebsartenregister (2 Bit).

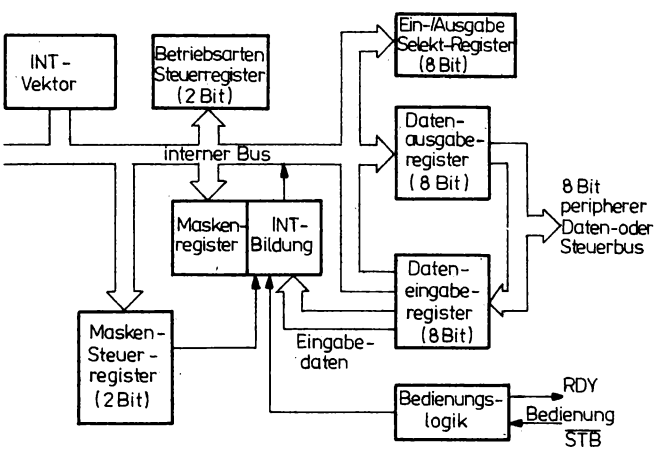


Bild 4.4
Logische Struktur eines Kanals des Bausteins *U855D*

Bild 4.4 zeigt die logische Struktur eines Kanals. Ein Kanal kann in folgenden Betriebsarten arbeiten:

- byteweise Ausgabe,
- byteweise Eingabe,
- byteweise bidirectional (über dieselbe Leitung können Informationen ein- bzw. ausgegeben werden),
- Einzelbitsteuerung.

4.2.2. Beschreibung der Anschlüsse

D_0 bis D_7 Datenbus zum Prozessor *U880* (bidirectional, tristate).

B/$\overline{A}$ – Kanalauswahl: legt fest, über welchen Kanal der Datentransfer zwischen *U880* und PIO stattfindet (Eingabesignal): »Low« = Tor A, »High« = Tor B.

C$\overline{D}$ – Auswahl Steuerwort/Datenwort: legt fest, ob das von *U880* kommende Wort ein Steuerwort oder ein Datenwort ist (Eingabesignal); »Low« = Datenwort, »High« = Steuerwort.

$\overline{CE}$ Chipauswahlsignal (Eingabesignal, aktiv »Low«).

$\overline{M1}$ Signal für M1-Zyklus: entspricht dem Signal M1 des Prozessors *U880*. Es dient zur Synchronisierung der PIO-Logik (Eingabesignal, aktiv »Low«).

$\overline{IORQ}$ Ein-/Ausgabe-Anforderungssignal: entspricht dem $\overline{IORQ}$ des Prozessors *U880*. Es dient der Ansteuerung des Bausteins bei einem Ein- und Ausgabebefehl. Bei gleichzeitigem Erscheinen von IORQ und M1 gibt das INTERRUPT erzeugende Tor automatisch einen INTERRUPT-Vektor an den Datenbus ab (Eingabesignal, aktiv »Low«).

$\overline{RD}$ Lesesignal: entspricht dem READ-Signal des Prozessors *U880* (Eingabesignal, aktiv »Low«).
Der Weg der Daten oder Befehle bei der Ein- und Ausgabe vom *U880* zum PIO wird durch die Signale $\overline{IORQ}$, $\overline{RD}$, B/$\overline{A}$, C/$\overline{D}$, $\overline{CE}$ gesteuert.

IEI INTERRUPT-Freigabesignal: dient zur Bildung von INTERRUPT-Prioritätsschaltungen (Eingabesignal, aktiv »High«).

IEO Signal für die Weitergabe der INTERRUPT-Freigabe, es ist das zu IEI gehörende Ausgangssignal.
Es ist »High«, wenn IEI gleich »High« ist und keine INTERRUPT-Anforderung vom Baustein selbst kommt. Im anderen Fall ist es »Low« (Ausgabesignal).

$\overline{INT}$ INTERRUPT-Anforderungssignal: wenn das Signal $\overline{INT}$ aktiv ist, fordert das PIO einen INTERRUPT im Prozessor (Ausgabesignal, aktiv »Low«).

A_0 bis A_7 Datenbuskanal A: A_0 bis A_7 dient zur Ein- und Ausgabe der Information zwischen Kanal A und einem peripheren Gerät, A_0 ist die niederwertigste Stelle.

$\overline{ASTB}$ Strobeimpuls (Torungsimpuls) für Kanal A (Eingabesignal, aktiv »Low«).

ARDY Bereitschaftssignal für Tor A: Bei Ausgabe sagt dieses Signal aus, daß das Ausgaberegister Daten enthält. Bei Eingabe bedeutet dieses Signal, daß der Inhalt des Eingaberegisters abgeholt worden ist und der Kanal bereit ist, neue Daten zu empfangen (Ausgabesignal, aktiv »High«).

B_0 bis B_7 Datenbuskanal B: B_0 bis B_7 dient zur Ein- und Ausgabe der Information zwischen Kanal B und einem peripheren Gerät. B_0 ist die niederwertigste Stelle.

$\overline{BSTB}$ Strobeimpuls (Torungsimpuls) für Kanal B (Eingabesignal, aktiv »Low«).

BRDY Bereitschaftssignal für Tor B: Bei Ausgabe sagt dieses Signal aus, daß das Ausgaberegister Daten enthält. Bei Eingabe bedeutet dieses Signal, daß der Inhalt des Eingaberegisters abgeholt worden ist und der Kanal bereit ist, neue Daten zu empfangen (Ausgabesignal, aktiv »High«).

Ein spezielles Löschsignal (RESET) gibt es nicht. Das Löschen der Register und der Flip-Flop des Bausteins geschieht entweder beim Anlegen der Betriebsspannung oder durch $\overline{M1}$, wenn nicht gleichzeitig eines der Signale $\overline{RD}$ bzw. $\overline{IORQ}$ aktiv ist. Dieses Löschen erzeugt folgenden Zustand:

- die Maskenregister sind gelöscht;
- beide Ausgaberegister sind gelöscht;
- das Register zur Bereitstellung des INTERRUPT-Vektors wird nicht gelöscht;
- Tor A und B werden hochohmig, die *Hand-shake*-Signale $\overline{ASTB}$, ARDY, $\overline{BSTB}$, BRDY werden inaktiv.

Der RESET-Zustand bleibt so lange bestehen, bis das PIO ein Steuerwort vom Prozessor erhält.

4.2.3. Arbeitsweise

Bevor eine Ein- bzw. Ausgabe zum peripheren Gerät vorgenommen werden kann, müssen der Arbeitsmodus und die INTERRUPT-Logik eingestellt sein. Dazu werden eine Reihe Steuerwörter in das PIO gespeichert. (Das PIO wird initialisiert.) Zuerst muß vom Prozessor ein Steuerwort zur Auswahl des Arbeitsmodus gegeben werden. Das Steuerwort hat folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
M	M	x	x	1	1	1	1

Die Stellen D_4 und D_5 sind ohne Bedeutung. Durch die Stellen D_6 und D_7 wird der Arbeitsmodus festgelegt.

	D_7	D_6	Arbeitsmodus
MODE 0	0	0	Ausgabe
MODE 1	0	1	Eingabe
MODE 2	1	0	bidirectional
MODE 3	1	1	Einzelbitsteuerung

MODE 0 Ausgabe:

Nach dem Übergeben des Steuerworts können bei der Arbeit mit Interrupt ein INTERRUPT-Vektor und ein INTERRUPT-Steuerwort folgen. Der INTERRUPT-Vektor hat folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	0

Das INTERRUPT-Steuerwort hat folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
M	x	x	x	0	0	1	1

Ist D_7 gleich 1, so wird die INTERRUPT-Bildung erlaubt (Bildung des Signals INT). Ist D_7 gleich 0, dann wird die INTERRUPT-Bildung gesperrt. Nach Speicherung von INTERRUPT-Steuerwort und INTERRUPT-Vektor ist der PIO-Baustein programmiert bzw. initialisiert. Es können die eigentlichen Ausgabebefehle zur Datenausgabe folgen.
Durch den Ausgabebefehl vom Prozessor wird das Ausgaberegister gefüllt und das Signal RDY (entweder ARDY oder BRDY) aktiv. Dieses Signal bleibt so lange aktiv, bis von einem peripheren Gerät das Signal $\overline{\text{STB}}$ ($\overline{\text{ASTB}}$, $\overline{\text{BSTB}}$) kommt. Mit dem Signal $\overline{\text{STB}}$ werden gewöhnlich die Daten vom Ausgaberegister in das periphere Gerät übernommen. Ist die Bildung einer INTERRUPT-Anforderung erlaubt, dann wird durch $\overline{\text{STB}}$ das Signal $\overline{\text{INT}}$ erzeugt. Das Signal $\overline{\text{INT}}$ führt zu einer Programmunterbrechung im Prozessor. Bei Annahme des INTERRUPT werden gleichzeitig die Signale $\overline{\text{M1}}$ und $\overline{\text{IORQ}}$ aktiv. Als Folge der Signale und $\overline{\text{IORQ}}$ gibt das PIO den INTERRUPT-Vektor auf den Datenbus. Mit Hilfe des INTERRUPT-Vektors bestimmt der Prozessor die Startadresse des INTERRUPT-Bedienprogramms.

MODE 1 Eingabe:
Nach dem Übergeben des Steuerwortes können bei der Arbeit mit INTERRUPT ein INTERRUPT-Vektor und ein INTERRUPT-Steuerwort folgen. Der INTERRUPT-Vektor hat folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	0

Das INTERRUPT-Steuerwort hat folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
M	x	x	x	0	0	1	1

Ist D_7 gleich 1, so wird die INTERRUPT-Bildung erlaubt (Bildung des Signals $\overline{\text{INT}}$). Ist D_7 gleich 0, dann wird die INTERRUPT-Bildung gesperrt. Nach Speicherung von INTERRUPT-Vektor und INTERRUPT-Steuerwort können Eingabebefehle zur Dateneingabe folgen.
Nach einem Eingabebefehl vom Prozessor ist das Eingaberegister leer und das Signal RDY (ARDY, BRDY) aktiv. Dieses Signal bleibt so lange aktiv, bis von einem peripheren Gerät das Signal $\overline{\text{STB}}$ ($\overline{\text{ASTB}}$ oder $\overline{\text{BSTB}}$) kommt. Mit der Vorderflanke des Signals $\overline{\text{STB}}$ können die Daten vom peripheren Gerät in das Eingaberegister übernommen werden.

Ist die Bildung einer INTERRUPT-Anforderung erlaubt, so wird durch $\overline{STB}$ das Signal $\overline{INT}$ erzeugt. Das Signal $\overline{INT}$ führt zu einer Programmunterbrechung im Prozessor. Bei Annahme des INTERRUPT werden gleichzeitig die Signale $\overline{M1}$ und $\overline{IORQ}$ aktiv. Als Folge der Signale $\overline{M1}$ und $\overline{IORQ}$ gibt das PIO den INTERRUPT-Vektor auf den Datenbus. Mit Hilfe des INTERRUPT-Vektors bestimmt der Prozessor die Startadresse des INTERRUPT-Bedienprogramms.

MODE 2 bidirectional:
Eine bidirectionale Ein- und Ausgabe ist nur über Kanal A möglich. Dabei werden die *Handshake*-Signale von Kanal A (ARDY, $\overline{ASTB}$) zur Ausgabe und die Signale von Kanal B (BRDY, $\overline{BSTB}$) zur Eingabe benutzt. Vor der Benutzung von Kanal A als bidirectionaler Datenbus muß Kanal B in den Modus Einzelbitsteuerung gebracht werden.
Nach dem Übergeben des Steuerwortes können bei der Arbeit mit INTERRUPT ein INTERRUPT-Vektor und ein INTERRUPT-Steuerwort folgen.
Der INTERRUPT-Vektor hat folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	0

Das INTERRUPT-Steuerwort hat wieder folgenden Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
M	x	x	x	0	0	1	1

Ist D_7 gleich 1, so wird die INTERRUPT-Bildung erlaubt (Bildung des Signals $\overline{INT}$). Ist D_7 gleich 0, dann wird die INTERRUPT-Bildung gesperrt.
Nach der Übergabe von INTERRUPT-Steuerwort und INTERRUPT-Vektor können die Ein- und Ausgabebefehle zum Datentransport gegeben werden. Nach einem Ausgabebefehl vom Prozessor ist das Ausgaberegister gefüllt und das Signal ARDY zur Kennzeichnung der Bereitschaft aktiv.
Eine Abnahme der Daten vom Ausgaberegister kann während der Zeit geschehen, in der $\overline{ASTB}$ und ARDY aktiv sind. Mit der Rückflanke von $\overline{ASTB}$ wird bei erlaubtem INTERRUPT das INT-Signal gebildet. Nach einem Eingabebefehl ist das Signal BRDY aktiv.
Mit dem Signal $\overline{BSTB}$ können neue Daten von einem peripheren Gerät in das Eingaberegister übernommen werden. Anschließend wird das Signal BRDY inaktiv. Auch in diesem Fall bildet man mit der Rückflanke von BSTB das INTERRUPT-Signal bei erlaubtem INTERRUPT.
Die Initialisierung für MODE 2 (bidirektional) ist in folgender Reihenfolge möglich:
Steuerwort Bitmode für Kanal B
INTERRUPT-Vektor für Kanal B
INTERRUPT-Steuerwort für Kanal B
Steuerwort bidirektional für Kanal A
INTERRUPT-Vektor für Kanal A
INTERRUPT-Steuerwort für Kanal A
Der INTERRUPT von Kanal B kommt bei BSTB, der von Kanal A bei ASTB.

Die Aus- und Eingabe von Daten erfolgt in der gleichen Weise wie in MODE 0 und MODE 1, wobei die Signale ARDY und ASTB bei Ausgabe und die Signale BRDY und BSTB bei Eingabe wirksam sind.

MODE 3 Einzelbitsteuerung:
Bei der Einzelbitsteuerung werden die Datenleitungen der Kanäle A und B als einzelne Steuerleitungen verwendet. Die *Hand-shake*-Signale haben in diesem Arbeitsmodus keine Bedeutung. Der MODE dient zur Aufnahme von Alarm- und Statussignalen sowie zur Ausgabe von Steuersignalen. Nach der Ausgabe des *Steuerwortes* mit dem Aufbau

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	x	x	1	1	1	1

muß ein weiteres Byte folgen, das festlegt, welche Datenleitungen des Kanals Ein- bzw. Ausgangsleitung sein sollen (Auswahlbyte). Ist eine Bit-Stelle dieses Bytes 0, so ist die entsprechende Datenleitung des Kanals Ausgangsleitung. Bei einer 1 in der Bit-Stelle ist die dazugehörige Datenleitung des Kanals eine Eingangsleitung. Wenn z. B. dieses Auswahlbyte den Aufbau

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	0	1	0	0	1

hat, so sind die Anschlüsse 0, 3 und 5 Eingangsleitungen und die Anschlüsse 1, 2, 4, 6 und 7 Ausgangsleitungen.
An das Ein-/Ausgabe-Auswahlbyte kann sich bei der Arbeit mit INTERRUPT ein INTERRUPT-Vektor mit dem Aufbau

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	0

anschließen. Nach dem INTERRUPT-Vektor folgt das INTERRUPT-Steuerwort. Es hat den Aufbau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
INTERRUPT erlaubt	ODER UND	Tief Hoch	Maske folgt	0	1	1	1

- Ist $D_7 = 1$, so ist die INTERRUPT-Bildung erlaubt.
- Bei $D_7 = 0$ ist die INTERRUPT-Bildung gesperrt.
- Ist $D_6 = 1$, so entsteht im PIO das Signal $\overline{INT}$, wenn alle als Eingang festgelegten Leitungen das durch D_5 definierte Potential haben (UND-Funktion).
- Bei $D_6 = 0$ wird im PIO das Signal $\overline{INT}$ gebildet, wenn eine der als Eingang gekennzeichneten Leitungen das durch D_5 definierte Potential hat (ODER-Funktion).

– Ist $D_5 = 1$, so führt ein »Hochpegel« an den Eingangsleitungen zum INTERRUPT.
– Bei $D_5 = 0$ führt ein »Tiefpegel« an den Eingangsleitungen zum INTERRUPT.
– Ist $D_4 = 1$, so muß dem Steuerwort eine INTERRUPT-Maske folgen.

INTERRUPT-Maske

M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

Zur INTERRUPT-Bildung werden in diesem Fall nur die Eingänge zugelassen, deren Bit-Stellen in der INTERRUPT-Maske 0 sind.
– Bei $D_4 = 0$ braucht keine INTERRUPT-Maske zu folgen.
Die Initialisierung für MODE 3 ist in folgender Reihenfolge möglich:
Steuerwort
Selektwort (muß folgen)
INTERRUPT-Vektor
INTERRUPT-Steuerwort
INTERRUPT-Maske

4.2.4. Beispiel zum *U855D*

An einem PIO Kanal A sollen 3 Steuerleitungen (Ausgabe) und 5 Sensorleitungen (Eingabe) angeschlossen werden.
Dazu werden die Leitungen A_0, A_1, A_2 als Ausgangsleitungen, die Leitungen A_3 bis A_7 als Eingangsleitungen festgelegt. Das Ein-/Ausgabe-Auswahlbyte muß folgenden Aufbau haben:
Ein-/Ausgabe-Auswahlbyte: 1 1 1 1 1 0 0 0
Im Prozessor soll ein INTERRUPT entstehen, wenn die Sensorsignale A_4, A_5 und A_7 gleichzeitig »Hochpegel« aufweisen. Für diesen Fall sieht das INTERRUPT-Steuerwort folgendermaßen aus:

INTERRUPT-Steuerwort	1 1 1 1 0 1 1 1

INTERRUPT-Maske	0 1 0 0 1 1 1 1

Das PIO sei nach Bild 4.5 mit dem Prozessor *U880* verbunden.
– Wenn das Adreß-Bit $A_0 = 1$ und die Adreß-Bit A_3 bis A_8 gleich 0 sind, wird dieses PIO angesteuert.
– Bei $A_1 = 0$ wird von diesem PIO Kanal A ausgewählt.
– Wenn $A_2 = 1$ ist, wird über den Datenbus ein Steuerwort, wenn $A_2 = 0$ ist, ein Datenwort transportiert.

Für die Steuer- und Datenworte nach Kanal A und B ergeben sich aus dieser Festlegung folgende Adressen:

Steuerwort nach Kanal A	Adresse 5
Datenwort von und nach Kanal A	Adresse 1
Steuerwort nach Kanal B	Adresse 7
Datenwort von und nach Kanal B	Adresse 3

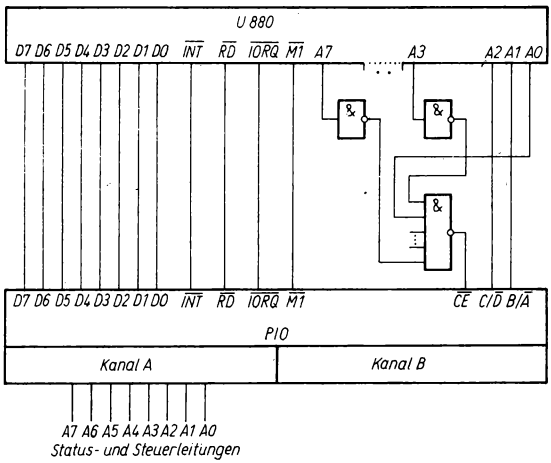


Bild 4.5
Zusammenschaltung eines PIO-Schaltkreises mit dem *U880*

Für die Initialisierung des geforderten Zustands sind folgende Befehle notwendig:

	Befehl	Wort auf Datenleitung
Ausgabe des Steuerwortes zur Festlegung von Bit-Mode	OUT 5	1 1 0 0 1 1 1 1
Eingabe/Ausgabe Auswahlbyte	OUT 5	1 1 1 1 1 0 0 0
Ausgabe INTERRUPT-Vektor	OUT 5	0 0 0 0 1 0 0 0
Ausgabe INTERRUPT-Steuerwort	OUT 5	1 1 1 1 0 1 1 1
Ausgabe INTERRUPT-Maske	OUT 5	0 1 0 0 1 1 1 1

Nach dieser Befehlsfolge wird im Prozessor ein INTERRUPT gefordert, wenn die Leitungen A_4, A_5, A_7 gleichzeitig »Hochpegel« aufweisen. Durch den Eingabebefehl IN 1 kann festgestellt werden, welche Sensorleitungen Hoch- bzw. Tiefpegel haben. Das durch den Befehl IN 1 in den Prozessor *U880* eingelesene Datenbyte setzt sich zusammen aus den Bit-Stellen 3 bis 7 des Eingaberegisters und den Bit-Stellen 0 bis 2 des Ausgaberegisters von Kanal A. Weitere Bausteine für die parallele Eingabe sind die Schaltkreise *8255* und *8212*. Der Baustein *8255* hat einen dem PIO ähnlichen Aufbau und läßt sich in gleicher Weise einsetzen.

4.3. Zeitgeberbaustein *U857D*

4.3.1. Struktur

Bild 4.6 zeigt die logische Struktur des CTC (Counter-Timer-Circuit – Zähler-Zeitgeber-Schaltkreis). Der Baustein CTC ist ein peripherer Baustein zum Prozessor *U880*. Er dient sowohl als Zähl- wie auch als Zeitgeberbaustein. Mit ihm lassen sich auf 4 getrennten Kanälen Zähler- sowie Zeitgeberfunktionen realisieren. Rechnerseitig hat der Baustein das zum *U880* passende Interface mit dem 8-Bit-Datenbus und den Steuersignalen $\overline{IORQ}$, $\overline{RD}$, $\overline{MI}$, System-

takt Φ, $\overline{RESET}$, $\overline{CE}$, CS_1, CS_0. Zur Realisierung von INTERRUPT-Ketten verfügt der Baustein über die Signale IEI, IEO, $\overline{INT}$.
Bild 4.7 zeigt den Aufbau eines Kanals. Hauptbestandteil des Kanals ist ein 8-Bit-Rückwärtszähler, der über das Zeitkonstantenregister eingestellt werden kann. Die Zählimpulse können über den Takteingang CLK oder über einen Vorteiler kommen. Der Ausgang ZC/TO zeigt den Nulldurchgang des Zählers nach außen. Zur Einstellung der Betriebsarten hat jeder Kanal ein Kanalsteuerregister.

4.3.2. Beschreibung der Anschlüsse

Im einzelnen haben die Anschlüsse folgende Bedeutung:

D_0 bis D_7 Datenbus zum Prozessor (bidirectional).

$\overline{M1}$, $\overline{IORQ}$, $\overline{RD}$ Steuersignale vom Prozessor (Eingang, aktiv »Low«).

$\overline{CE}$ Chipauswahlsignal (Eingang, aktiv »Low«).

CS_0, CS_1 Enthält die Nummer des auszuwählenden Kanals (Eingang, aktiv »High«).

CS_1	CS_0	Kanal-Nr.
0	0	0
0	1	1
1	0	2
1	1	3

$\overline{INT}$ INTERRUPT-Anforderung vom CTC an den Prozessor (Ausgang, aktiv »Low«).

IEI INTERRUPT-Freigabesignal (Eingabe, aktiv »High«). Ist IEI aktiv (»High«), dann kann der Baustein eine INTERRUPT-Anforderung bilden.

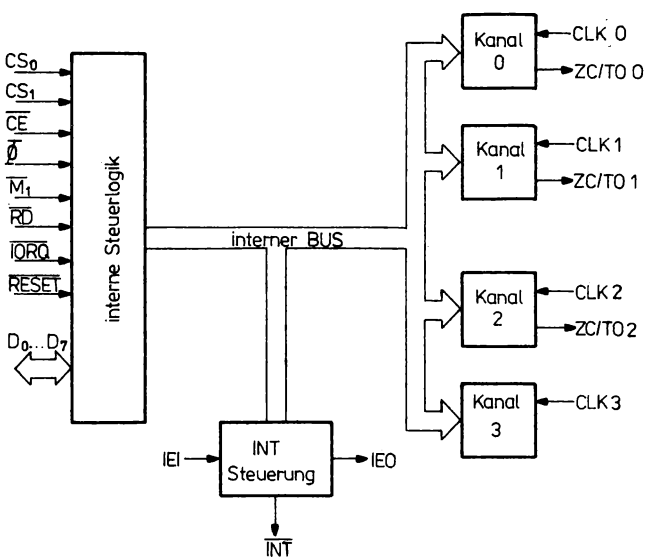


Bild 4.6
Grundstruktur des Bausteins *U857D*

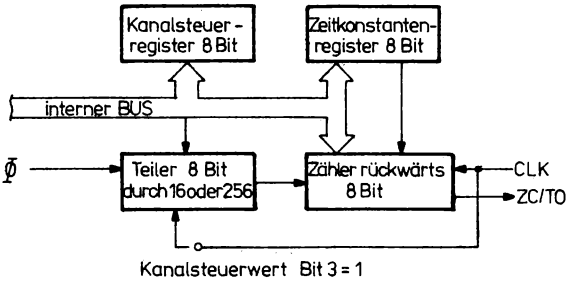


Bild 4.7
Logische Struktur eines Kanals des Bausteins *U857D* (Im Bild lies Kanalsteuerwort)

IEO	INTERRUPT-Freigabesignal für den nächsten in der Kette liegenden Ein-/Ausgabe-Baustein (Ausgabe, aktiv »High«). Der Ausgang IEO hat nur dann Hochpotential, wenn im Baustein keine INTERRUPT-Anforderung gebildet wird und IEI ebenfalls Hochpotential hat. Das heißt, existiert im Baustein und links vom Baustein keine INTERRUPT-Anforderung, dann werden die Bausteine auf der rechten Seite der Kette freigegeben.
CLK 0–3	Externer Zähleingang für die Kanäle 0 bis 3 (Eingang, aktiv »High«).
ZC/TO 0–2	Zeigt den Nulldurchgang der Zähler der Kanäle 0 bis 2 an (Ausgang, aktiv »High«).
$\overline{\text{RESET}}$	Rücksetzsignal. Dieses Signal stellt die Zähler auf 0, setzt die INTERRUPT-Freigabe-Bit zurück (Eingang, aktiv »Low«).

4.3.3. Arbeitsweise

Jeder Kanal kann in der Betriebsart Zähler oder Zeitgeber arbeiten. In der Betriebsart Zähler wird mit der steigenden Flanke des Eingangs CLK der Zähler um 1 zurückgezählt. Der Zähler läuft synchron mit dem Systemtakt Φ, d. h., das Rückwärtszählen nach der steigenden Flanke von CLK erfolgt mit dem nächsten Takt. Bei Nulldurchgang des Zählers wird eine INTERRUPT-Anforderung gebildet und die Zeitkonstante in den Zähler geladen. In der Betriebsart Zeitgeber wird der Rückwärtszähler über einen Vorteiler mit dem Systemtakt Φ getaktet. Der Vorteiler läßt sich vom externen Eingang CLK triggern. Der Triggerzeitpunkt wird durch Bit D_3 im Kanalsteuerwort bestimmt. Es gilt:

D_3	D_2	(Kanalsteuerwort)
0	0	Zeitmessung beginnt am Anfang des nächsten Maschinenzyklus
0	1	Zeitmessung beginnt am Anfang des nächsten Maschinenzyklus nach dem Laden der Zeitkonstante.
1	0	Zeitmessung beginnt am Anfang des nächsten Maschinenzyklus, nachdem der Eingang CLK die vorgesehene Flanke durchläuft.
1	1	Zeitmessung beginn am Anfang des nächsten Maschinenzyklus, nachdem die Zeitkonstante geladen ist und die vorgesehene Flanke CLK am Eingang erscheint.

Beim Nulldurchgang des Zählers wird ein INTERRUPT gebildet, der zur Realisierung eines Uhrenprogramms mit Hilfe des Mikroprozessors verwendet werden kann. Die INTERRUPT-Periode t_i errechnet sich nach

$$t_i = t_c \cdot P \cdot TK;$$

mit t_c – Systemtaktperiode, P – Vorteiler (16 oder 256), TK – Zeitkonstante.

Programmierung des CTC (Initialisierung)

a) Einschreiben des INTERRUPT-Vektors

Der INTERRUPT-Vektor hat folgendes Format:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	x	x	0

Die Bit-Stellen D_1 und D_2 werden im Baustein bei INTERRUPT-Anforderung mit der betreffenden Kanalnummer belegt.

b) Einschreiben des Kanalsteuerworts

Das Kanalsteuerwort hat folgenden Aufbau:

D_7

INT Freigabe, 0 gesperrt, 1 frei

D_6

Betriebsart, 0 Zeitgeber, 1 Zähler

D_5

0 Teiler 16, 1 Teiler 256 nur in der Betriebsart »Zeitgeber«

D_4

0 neg. Flanke, 1 pos. Flanke

D_3

Bestimmt Triggerzeitpunkt nur in der Betriebsart »Zeitgeber«

D_2

Zeitkonstante laden, 0 kein Laden, 1 das nächste Steuerwort für den angewählten Kanal wird als Zeitkonstante interpretiert. Wird eine Zeitkonstante während eines Zeitmeßvorgangs eingegeben, so erfolgt die Übernahme der Zeitkonstanten in den Zähler erst, wenn der Meßvorgang beendet ist.

D_1

Rücksetzen; 0 Kanalzähler zählt weiter, 1 Kanal unterbricht das Zählen, bis eine Zeitkonstante eingegeben wird.

ZC/TO = inaktiv; INTERRUPT ist gesperrt.

D_0

1 Kanalsteuerwort, 0 INTERRUPT-Vektor

c) Laden der Zeitkonstante

Ist im Kanalsteuerwort das Laden einer Zeitkonstanten gefordert, so wird die Zeitkonstante nach dem Kanalsteuerwort eingeschrieben. Die Zeitkonstante ist ein Dualwert zwischen 0 und 255.

Für die Zeitkonstante 256 ist 00H zu laden.

d) Lesen des Rückwärtszählers

Der momentane Wert des Rückwärtszählers kann zu jedem beliebigen Zeitpunkt vom Prozessor aus, mit Hilfe eines Eingabebefehls, gelesen werden. Der Wert repräsentiert die Anzahl der positiven Flanken vor der positiven Flanke von T2 des Lesezyklus.

INTERRUPT-Bildung

Liegt im Baustein eine Bedingung zur Bildung einer INTERRUPT-Anforderung vor (Nulldurchgang eines Zählers), so wird der $\overline{INT}$-Ausgang aktiv, wenn die INTERRUPT-Bildung erlaubt ist.

4.4. Serieller Ein-/Ausgabebaustein *U856D*

4.4.1. Struktur

Bild 4.8 zeigt die Grundstruktur des SIO-Bausteins (Seriell Input/Output). Der SIO-Baustein setzt das Rechnerinterface (Rechnerbus) in ein Interface zur Steuerung peripherer Geräte mit serieller Dateneingabe bzw. -ausgabe um. Der Baustein hat eine interne Steuerlogik, die mit Hilfe des Grundtakts arbeitet, sowie eine Interruptsteuerung mit der Möglichkeit zur Bildung von Prioritätsketten. Zur Peripheriesteuerung gibt es 2 Kanäle. Über jeden Kanal können Informationen bitseriell gesendet und empfangen werden.

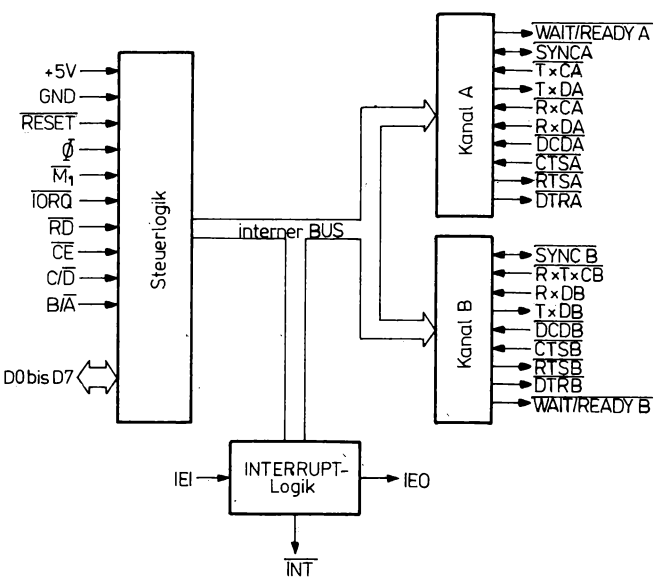


Bild 4.8
Grundstruktur des Bausteins *U856D*

4.4.2. Beschreibung der Anschlüsse

Signale des Rechnerinterface:

D_0 bis D_7	Datenbus zum Prozessor *U880.*
B/$\overline{A}$	Kanalauswahl; »Low« ≙ Tor A; »High« ≙ Tor B
Eingangssignal	
C/$\overline{D}$	Auswahl Steuerwort/Datenwort: Dieses Signal legt fest, ob das vom Pro-

Eingangssignal	zessor kommende Wort ein Steuerwort (»High«) oder ein Datenwort (»Low«) ist.
$\overline{CE}$ Eingangssignal	Chipauswahlsignal (aktiv ≙ »Low«).
$\overline{M1}$ Eingangssignal	Signal für M1-Zyklus: Dieses Signal entspricht dem Signal $\overline{M1}$ des Prozessors.
$\overline{RD}$ Eingangssignal	Lesesignal: Dieses Signal entspricht dem $\overline{RD}$-Signal des Prozessors (aktiv ≙ »Low«).
$\overline{IORQ}$ Eingangssignal	Eingabe-/Ausgabe-Anforderung vom Prozessor (aktiv = »Low«).
IEI Eingangssignal	INTERRUPT-Freigabesignal: Dieses Signal dient zur Bildung von IN-INTERRUPT-Prioritätsschaltungen.
IEO Ausgangssignal	Signal für die Weitergabe der INTERRUPT-Freigabe: Bei diesem Signal handelt es sich um das Ausgangssignal zu IEI. Es ist »High«, wenn IEI »High« ist und keine INTERRUPT-Anforderung vom SIO-Baustein kommt. Sonst ist es »Low«.
$\overline{INT}$ Ausgangssignal	INTERRUPT-Anforderungssignal (aktiv ≙ »Low«).
Φ Eingangssignal	Systemtakt.
$\overline{RESET}$	Rücksetzen der Empfänger- und Senderlogik. An T × DA und T × DB entsteht das Grundsignal. Steuerregister sind gelöscht. Alle Interrupts sind gesperrt (aktiv ≙ »Low«).
Signale der Kanäle	A und B:
R × DA, R × DB Eingangssignale	Empfangsdaten (aktiv ≙ »High«).
T × DA, T × DB Ausgangssignale	Sendedaten (aktiv ≙ »High«).
$\overline{R \times CA}$, $\overline{R \times CB}$ [9] Eingangssignale	Empfängertakt (aktiv ≙ »Low«).
$\overline{T \times CA}$, $\overline{T \times CB}$ [9] Eingangssignale	Sendertakt (aktiv ≙ »Low«).
$\overline{CTSA}$, $\overline{CTSB}$ Eingangssignale	Steuersignal für Datensender. Im Arbeitsmodus AUTOENABLE werden bei $\overline{CTS}$ = »High« Daten nicht gesendet. In anderen Arbeitsmode können diese Anschlüsse als Eingänge anderweitig verwendet werden. Die beiden Leitungen haben *Schmitt*-Trigger-Eingänge (aktiv ≙ »Low«).
$\overline{DCDA}$, $\overline{DCDB}$ Eingangssignale	Steuersignal für Datenempfang. Das Signal $\overline{DCD}$ arbeitet analog dem Signal $\overline{CTS}$, jedoch wird durch $\overline{DCD}$ der Empfänger des zugehörigen Kanals gesperrt (aktiv ≙ »Low«).
$\overline{RTSA}$, $\overline{RTSB}$ Ausgangssignale	Programmierbares Quittungssignal. Sobald das RTS-Bit eines Kanals gesetzt ist, geht der zugehörige $\overline{RTS}$-Ausgang in den »Low«-Zustand. Wird das RTS-Bit im Asynchronmodus rückgesetzt, geht der zugehörige $\overline{RTS}$-

[9] Aus Gründen begrenzter Pin-Anzahl stehen $\overline{T \times CB}$, $\overline{R \times CB}$ und $\overline{DTRB}$ nur 2 Pins zur Verfügung.
Standardversion: $\overline{T \times CB}$ und $\overline{R \times CB}$ gleich 1 Pin $\overline{R \times T \times CB}$.
Sonderversionen: siehe Bild 4.12.

Ausgang in den »High«-Zustand, sobald das Senderegister leer ist. Im Synchronmodus folgt der Anschluß dem RTS-Bit (aktiv ≙ »Low«).

$\overline{DTRA}$, $\overline{DTRB}$[9] Programmierbares Quittungssignal. Der Ausgang folgt dem Wert des DTR-Bit (aktiv ≙ »Low«).

Ausgangssignale

$\overline{SYNCA}$, $\overline{SYNCB}$ Synchronisationssignal (aktiv ≙ »Low«). Bei interner Synchronisation: »Ausgang«. Das Signal ist in dem Takt aktiv, in dem ein Synchronisationszeichen erkannt wird.

Bei externer Synchronisation: »Eingang«. Der Aufbau eines Zeichens beginnt mit der nach SYNC folgenden steigenden Flanke von $\overline{R \times C}$. Zwischen 2 aktiven SYNC-Signalen müssen mindestens 3 Takte liegen.

4.4.3. Aufbau eines Kanals

Bild 4.9 zeigt den Aufbau eines Kanals. Zu einem Kanal gehören folgende Register:
Verschieberegister für den Empfang,
Verschieberegister für das Senden,
drei Empfangspufferregister,
Sendepufferregister,
Register für die Speicherung des SYNC-Zeichens,
acht Register für Steuerinformationen (WR_0 bis WR_7),
drei Register für Statusinformationen (RR_0 bis RR_2).

Senden von seriellen Daten:
Die zu sendende Information gelangt zuerst in den Sendepuffer, anschließend kommt sie in das Verschieberegister und wird von da aus seriell über den Ausgang $T \times D$ gesendet. Die Sende-Bits erscheinen mit der fallenden Flanke von $\overline{T \times C}$, wobei durch den »Taktzähler Senden« eine Untersetzung programmierbar ist.

Empfang von seriellen Daten:
Die Daten kommen über den Eingang $R \times D$ ins Verschieberegister und von da aus in die 3 Empfängerpuffer. Die Eingangsdaten werden mit der steigenden Flanke von $R \times C$ abgetastet. Durch den »Taktzähler Empfang« läßt sich eine Untersetzung des Empfangstaktes programmieren.

Programmierung des SIO:
Die Initialisierung des SIO geschieht mit Hilfe von Steuerbytes ($C/\overline{D}$-Pin = »High«). Das 1. Steuerbyte nach RESET geht in das Register WR_0. Das 2. Steuerbyte wird in einem WR-Register gespeichert, dessen Nummer in den Bit-Stellen D_2, D_1, D_0 von WR_0 steht. Das nächste Steuerbyte geht wieder nach WR_0, dann wieder in das durch die Bit-Stellen D_2, D_1, D_0 angezeigte Register und so fort. Beim Lesen der RR-Register muß nach RESET zunächst ein Steuerbyte in das Register WR_0 gebracht werden, in dem die Nummer des zu lesenden RR-Registers angegeben ist.

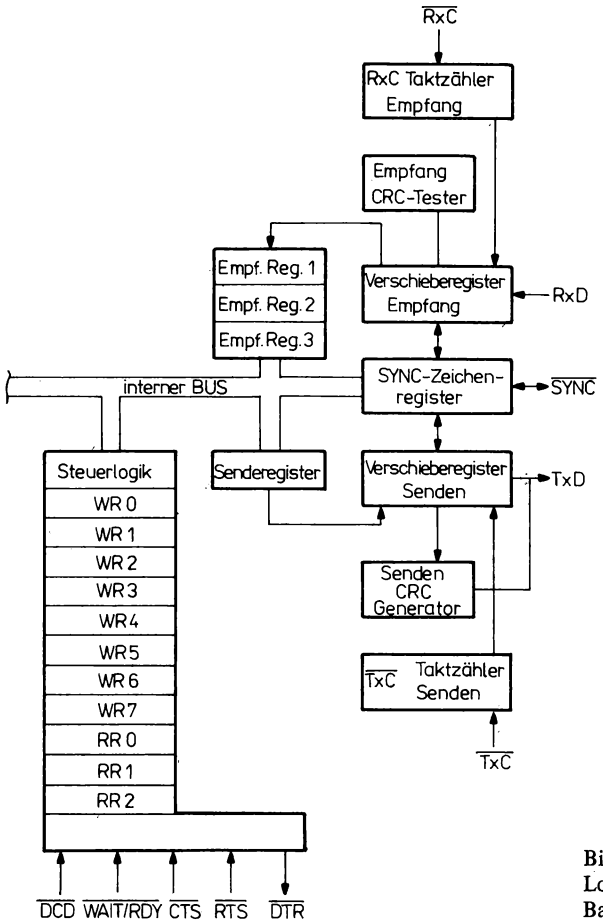


Bild 4.9
Logische Struktur eines Kanals des Bausteins *U856D*

4.4.4. Arbeitsweise

Asynchronmodus
Sendeformat

Nullinie	Start	D_0	D_1	D_2 bis D_n	Parity	STOP
			5, 6, 7, 8		programmierbar	1, 1½
			Bit/Zeichen		ja/nein	oder
					gerade/ungerade	2 Bit

Senden
- Senden beginnt mit TRANSMIT ENABLE,
- bei AUTOENABLE muß $\overline{CTS}$ »Low« sein,
- bei n < 8 Bit müssen vordere Bit 0 sein.

Empfang
- Empfang beginnt mit RECEIVER ENABLE,
- bei AUTOENABLE muß $\overline{DCD}$ »Low« sein.

Die empfangenen Bit werden in der Mitte einer Bit-Zeit abgetastet. Ist ½-Bit-Zeit nach der fallenden Flanke des Eingangssignals noch Tiefpegel, so wird das als Start-Bit erkannt, und die folgenden Bit werden empfangen.
- BREAK kann durch Setzen eines Bit in einem WR-Register gesendet werden. Es entsteht »Low«-Pegel, solange das Bit gesetzt ist (Nullinie ist »High«-Pegel).

Synchronmodus

Im Synchronmodus beginnt der Empfang erst nach Erkennen der Synchronisation. Vom Sender kommt entweder ein SYNC-Signal (externe Synchronisation) oder ein Synchronisationszeichen im Text (interne Synchronisation). Im Synchronmodus existiert nach RESET und RECEIVER ENABLE der HUNTMODE. In diesem Zustand werden keine Daten empfangen. Der HUNTMODE wird beendet, wenn Synchronisation eingetreten ist. Danach beginnt der Datentransfer. Ist die Synchronisation verlorengegangen, so kann der HUNTMODE gesetzt werden.

Monosynchronisation

Der Empfang beginnt, wenn das SYNC-Zeichen (8 Bit) erkannt ist. Das Vergleichszeichen steht in WR_7. Bei Erkennen des SYNC-Zeichens wird der SYNC-PIN aktiv (bis zum Ende des Taktzyklus, in dem das SYNC-Zeichen erkannt wurde).

Bisynchronisation

Der Empfang beginnt, wenn ein 16-Bit-SYNC-Wort (2 aufeinanderfolgende Zeichen) erkannt ist. Die Vergleichszeichen stehen in WR_6 und WR_7. Bei Erkennen der SYNC-Zeichen wird der $\overline{SYNC}$-PIN aktiv (vom Erkennen bis zum Ende des Taktzyklus, in dem sie erkannt wurden).

Externe Synchronisation

Der Empfang beginnt mit der 1. steigenden Flanke $\overline{R \times C}$, nachdem $\overline{SYNC}$ ($\overline{SYNC}$-PIN) aktiv geworden ist.

Bild 4.10 zeigt die Datenformate für die Betriebsarten des Synchronmodus.

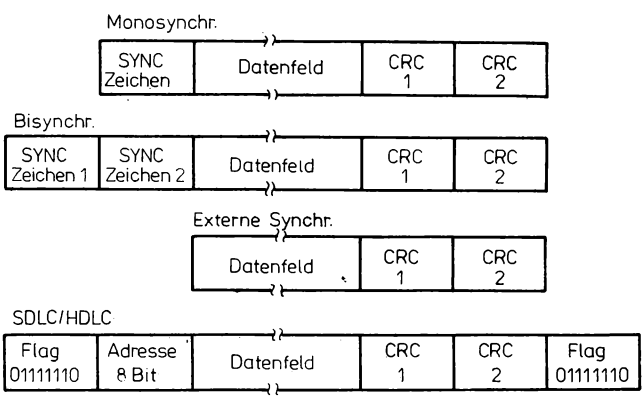


Bild 4.10
Datenformate im Synchronmodus des Bausteins *U856D*

Sender
- Mit der fallenden Flanke von $\overline{T \times C}$ wird ein Bit gesendet.
- Grundzustand (nach RESET) ist Markierungslinie.
- Nach Modusauswahl und TRANSMIT ENABLE ist Grundzustand »Senden von Synchronisationszeichen«.
- BREAK ergibt Grundpegel (»Low«-Pegel).

Empfänger
- Mit der steigenden Flanke von $\overline{R \times C}$ wird die Information abgetastet.
- Der Empfänger ist nach RESET im HUNTMODE. Er beginnt mit RECEIVER ENABLE.
- Nach Synchronisation beginnt die Datenübernahme.
- Synchronisation liegt vor, wenn
  - bei Monosynchronisation das Zeichen WR_7 erkannt ist;
  - bei Bisynchronisation das 16-Bit-Wort in WR_6 und WR_7 erkannt ist;
  - bei externer Synchronisation der $\overline{SYNC}$-PIN von außen »Low« gesetzt ist.
- Abbruch des Empfangs erfolgt, wenn:
  - RESET erscheint;
  - Bit RECEIVER ENABLE rückgesetzt wird;
  - im AUTOENABLE-Zustand $\overline{DCD}$ »High« wird;
  - der HUNTMODE gesetzt wird.

CRC-Bildung
- CRC-Berechnung beginnt 8-Bit-Zeiten, nachdem das Wort im Empfängerpuffer ist.
- CRC Codes: $x^{16} + x^{15} + x^2 + 1$ bzw.
  $x^{16} + x^{12} + x^5 + 1$ bei SDLC.
- CRC-Berechnung läßt sich sperren und freigeben durch das Bit TRANSMIT CRC ENABLE.
- Während CRC-Sendung ist das CRC/SYNC-Bit gesetzt. Das TRANSMIT-BUFFER-EMPTY-Bit ist gesetzt.
- SYNC-Zeichen gehen nicht in CRC ein.

4.4.5. INTERRUPT- und Statusbildung

Es gibt 4 Haupt-INTERRUPT-Ursachen:
1. Es kommt zum INTERRUPT, wenn der Sendepuffer keine Daten mehr zum Senden hat. Dieser INTERRUPT kann durch Nullsetzen des Bit TRANS INTERRUPT ENABLE im Steuerregister WR_1 gesperrt werden.
2. Externer INTERRUPT tritt auf bei Statuswechsel. Hierzu gehören 5 Ursachen, die im Statusregister RR_0 durch die entsprechenden Bit angezeigt werden:
   - Zustandswechsel (»Low«-»High« oder umgekehrt) am CTS-Eingang.
   - Zustandswechsel am DCD-Eingang.
   - Zustandswechsel am SYNC-Eingang (Asynchronmodus und Synchronmodus mit externer Synchronisation) bzw. am SYNC-Ausgang (Monosyncmodus und Bisyncmodus).

- BREAK oder eine Abbruch-Bit-Folge ist erkannt worden.
- Beginn des Sendens des CRC-Wortes.

Dieser INTERRUPT kann durch Nullsetzen des Bit EXT INTERRUPT ENABLE im Steuerregister WR_1 gesperrt werden.

3. Je nach eingestelltem INTERRUPT-MODE tritt ein Interrupt dann auf, wenn entweder das 1. Zeichen nach Einstellung des INTERRUPT-MODE in den Empfängerpuffer gelangt ist oder wenn mindestens 1 Zeichen im Empfängerpuffer zur Verfügung steht.
   Durch Nullsetzen der Bit D_3 und D_4 im Steuerregister WR_1 kann dieser INTERRUPT gesperrt werden.
4. Zum INTERRUPT kommt es bei einer besonderen Empfangsbedingung. Hierzu gehören 4 Ursachen, die im Statusregister RR_1 durch die entsprechenden Bit angezeigt werden:
   - Paritätsfehler,
   - Empfängerpufferüberlauf,
   - Formatfehler,
   - Blockende (nur SDLC).

Wann diese INTERRUPT zugelassen oder gesperrt sind, hängt vom eingestellten INTERRUPT-MODE ab.
Nach jedem INTERRUPT wird die Haupt-INTERRUPT-Ursache in den INTERRUPT-Vektor (Steuerregister WR_2) Bit 1 und 2 eingetragen, sofern das Bit STATUS AFFECTS VECTOR im Steuerregister WR_1 gesetzt ist.

INTERRUPT-Bildung Senden

- TRANSMIT BUFFER EMPTY (bei ASYNC):

INTERRUPT tritt auf, wenn der Sendepuffer gerade leer geworden ist, d. h., wenn keine zu sendenden Daten mehr im Sendepuffer sind.

- CTS (bei ASYNC und SYNC):

Es kommt zum externen INTERRUPT, wenn ein Zustandswechsel (»Low«-»High« oder umgekehrt) am CTS-Eingang aufgetreten ist.

- CRC/SYNC (bei SYNC):

Während des Sendens des CRC-Wortes ist das Bit TRANSMIT BUFFER EMPTY noch nicht gesetzt. Nachdem das CRC-Wort gesendet ist, wird das Bit TRANSMIT BUFFER EMPTY gesetzt, das wiederum einen INTERRUPT erzeugt. Wenn das CRC-Senden beginnt, wird das Bit SENDING CRC/SYNC gesetzt und ein Status-INTERRUPT erzeugt.

- TRANSMIT BUFFER EMPTY (bei SYNC):

Wenn der CRC-Generator nicht arbeitet, kommt es zum INTERRUPT, wenn der Sendepuffer leer geworden und das 1. SYNC-Zeichen in den Sendepuffer geladen ist. Es beginnt das automatische Senden von SYNC-Zeichen (Monosynchronmodus, Bisynchronmodus).
INTERRUPT tritt auf, wenn der CRC-Generator arbeitet und wenn der Sendepuffer leer geworden sowie das CRC-Wort gesendet worden ist. Dann beginnt das automatische Senden von SYNC-Zeichen.

INTERRUPT-Bildung Empfang

INTERRUPT-MODE 1

- RECEIVE CHARACTER AVAILABLE (bei ASYNC und SYNC):

INTERRUPT tritt auf, wenn das 1. Zeichen eines zu übertragenden Blockes, d. h. das 1. Zei-

chen nach Einstellen des INTERRUPT-MODE, im Empfängerpuffer angekommen ist.
– SYNC/HUNT (bei SYNC):
Es kommt zum externen INTERRUPT bei einem Zustandswechsel am SYNC-Ausgang vom Zustand »Zeichensynchronisation erreicht« in den Zustand »Herstellen Zeichensynchronisation« und umgekehrt.
– SYNC/HUNT (bei ASYNC):
Es kommt zum Status-INTERRUPT, wenn ein Zustandswechsel am SYNC-Eingang aufgetreten ist.
– BREAK/ABORT (bei ASYNC):
Ein Status-INTERRUPT tritt auf, wenn ein Break erkannt worden ist.
– DCD (bei ASYNC und SYNC):
Es kommt zum externen INTERRUPT, wenn ein Zustandswechsel am DCD-Eingang aufgetreten ist.
– Besondere Empfangsbedingung:
INTERRUPT tritt auf bei
- Empfängerpufferüberlauf (bei ASYNC und SYNC),
- Formatfehler (bei ASYNC).

Paritätsfehler führen nicht zum INTERRUPT.

INTERRUPT-MODE 2
– RECEIVE CHARACTER AVAILABLE (bei ASYNC und SYNC):
INTERRUPT tritt auf, wenn mindestens 1 Zeichen im Empfängerpuffer verfügbar ist. Dieser INTERRUPT kommt bei jedem Zeichen.
– DCD:
wie INTERRUPT-MODE 1
– SYNC/HUNT:
wie INTERRUPT-MODE 1
– Besondere Empfangsbedingung:
INTERRUPT bei Paritätsfehler (bei ASYNC und SYNC).
Der Paritätsfehler-INTERRUPT ist mit einer Eintragung im INTERRUPT-Vektor verbunden.

INTERRUPT-MODE 3
Wie INTERRUPT-MODE 2, jedoch verursacht der Paritätsfehler-INTERRUPT keine Eintragung im INTERRUPT-Vektor.

Statusbildung Senden

– TRANSMIT BUFFER EMPTY (bei ASYNC):
Das Bit wird gesetzt, wenn der Sendepuffer leer geworden ist. Es bleibt gesetzt, solange der Sendepuffer leer ist.
– TRANSMIT BUFFER EMPTY (bei SYNC):
Bei nicht arbeitendem CRC-Generator wird das Bit gesetzt, wenn der Sendepuffer leer geworden ist, d. h., wenn keine zu sendenden Zeichen mehr im Sendepuffer sind. Es wird mit dem automatischen Senden von SYNC-Zeichen begonnen (Monosync-MODE, Bisync-MODE). Das Bit ist gesetzt, solange der Sendepuffer leer ist, d. h., solange er keine Daten zum Senden hat.

Bei arbeitendem CRC-Generator wird das Bit gesetzt, wenn der Sendepuffer leer geworden und das CRC-Wort gesendet worden ist. Es beginnt das automatische Senden von SYNC-Zeichen.
– (Monosync-MODE, Bisync-MODE):
Wenn man den CRC-Generator nicht abschaltet, so wird über die SYNC-Zeichen ein neues Wort berechnet. Das Bit bleibt so lange gesetzt, bis der Sendepuffer leer ist, d. h., solange er keine Daten zum Senden hat.
– CTS (bei ASYNC und SYNC):
Das Bit wird gesetzt entsprechend dem Zustand des CTS-Eingangs zum Zeitpunkt des letzten Wechsels eines der 5 Statussignale DCD, CTS, SYNC/HUNT, BREAK/ABORT oder SENDING CRC/SYNC.
– ALL SENT (bei ASYNC):
Das Bit wird gesetzt, wenn alle Bit des Schieberegisters den Sender verlassen haben. Es verursacht keinen INTERRUPT.
– CRC/SYNC (bei SYNC):
Das Bit wird gesetzt, wenn, nachdem der Sendepuffer leer geworden ist, mit dem Senden des CRC-Wortes begonnen wird. Nach dem Senden des CRC-Wortes wird das Bit TRANSMIT BUFFER EMPTY gesetzt und mit dem Senden von SYNC-Zeichen begonnen (Monosync-MODE, Bisync-MODE). Das Bit bleibt so lange gesetzt, bis es durch das Kommando »RESET CRC/SYNC«, »RESET EXTERNAL/STATUS INTERRUPT« oder »CHANNEL RESET« zurückgesetzt wird. Voraussetzung für diese Statusanzeige ist, daß der CRC-Generator arbeitet.

Bit CRC/SYNC gesetzt und Bit TRANSMIT BUFFER EMPTY nicht gesetzt	CRC-Wort wird gesendet
Bit CRC/SYNC gesetzt und Bit TRANSMIT BUFFER EMPTY gesetzt	SYNC-Zeichen werden gesendet

Statusbildung Empfang
– RECEIVER CHARACTER AVAILABLE (ASYNC und SYNC):
Das Bit ist so lange gesetzt, wie mindestens 1 Zeichen im Empfängerpuffer verfügbar ist.
– DCD (ASYNC und SYNC):
Das Bit wird entsprechend dem Zustand des DCD-Signals zum Zeitpunkt des letzten Wechsels eines der 5 Statussignale DCD, CTS, SYNT/HUNT, BREAK/ABORT oder CSC/SYNC gesetzt.
– SYNC/HUNT (ASYNC):
Das Bit wird entsprechend dem Zustand am SYNC-Eingang zum Zeitpunkt des letzten Wechsels eines der 5 Statussignale DCD, CTS, SYNC/HUNT, BREAK/ABORT oder CRC/SYNC gesetzt.
– SYNC/HUNT (SYNC):
Das Bit wird gesetzt, wenn durch Setzen des Bit ENTER HUNTMODE der Zustand »Herstellen Zeichensynchronisation« eingestellt wird. Es wird rückgesetzt, wenn die Zeichensynchronisation erreicht ist, und bleibt rückgesetzt, auch wenn die Zeichensynchronisation ab-

bricht, bis zum erneuten Setzen des Bit ENTER HUNTMODE. Bei abgeschaltetem Empfänger oder nach einem CHANNEL-RESET-Kommando ist dieses Bit gesetzt.

– INTERRUPT PENDING (ASYNC und SYNC):

Dieses Bit wird gesetzt, wenn irgendeine INTERRUPT-Bedingung im gesamten SIO vorhanden ist. Es existiert nur im Kanal A.

– Besondere Empfangsbedingungen:

- Empfängerüberlauf (ASYNC und SYNC): Das Bit ist gesetzt bei Empfängerpufferüberlauf. Danach bleibt es so lange gesetzt, bis es durch das Kommando ERROR RESET rückgesetzt wird.
- Format- oder CRC-Fehler (SYNC): Das Bit zeigt das Vergleichsergebnis der CRC-Prüfung an.
- Format- oder CRC-Fehler (ASYNC): Das Bit wird gesetzt, wenn ein Formatfehler aufgetreten ist. Es wird nicht blockiert (bleibt für das Zeichen gesetzt, das den Fehler ausgelöst hat).
- Paritätsfehler (ASYNC und SYNC): Dieses Bit wird gesetzt, wenn ein Paritätsfehler aufgetreten ist. Danach bleibt es so lange gesetzt, bis das Kommando ERROR RESET dieses Bit wieder rücksetzt.

– BREAK/ABORT (ASYNC):

Das Bit wird gesetzt, wenn ein Break erkannt worden ist. Erst wenn dieser BREAK-Zustand nicht mehr besteht, kann dieses Bit durch das Kommando RESET EXTERNAL/ STATUS INTERRUPT rückgesetzt werden.

4.4.6. Beschreibung der Bit-Stellen der Steuerregister WR_0 bis WR_7 und der Statusregister RR_0 bis RR_2 (Bild 4.11)

WR_0:

D_0 bis D_2 Nr. des nächsten zu ladenden oder zu lesenden Registers.

D_3 bis D_5 RESET-Kommandos

SEND ABORT (nur SDLC). Es wird eine Folge von 8 bis 13 »1« gesendet.

RESET EXT STATUS INT löscht die im RR_0 bei Statusinterrupt eingetragenen Bit.

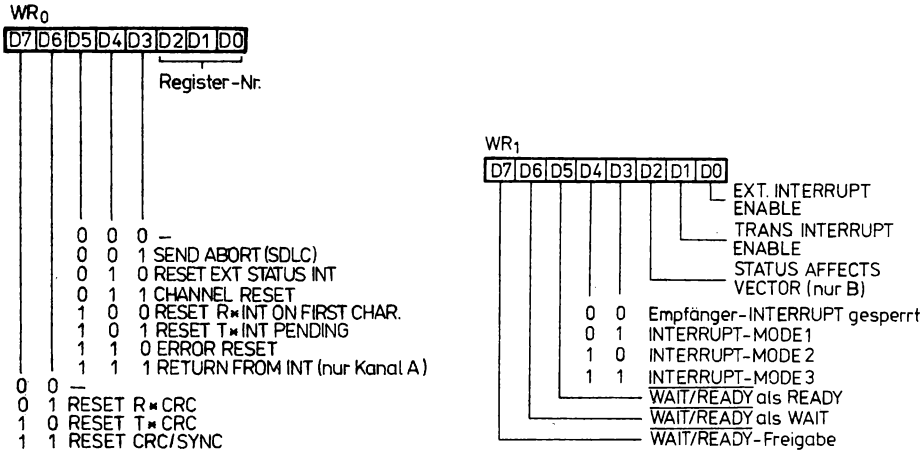
CHANNEL RESET führt zum Rücksetzen eines Kanals. Alle Steuer- und Statusregister sind gelöscht. RESET R × INT ON FIRST CHAR: Nach Erscheinen eines Interrupt im Interruptmodus 1 muß durch RESET R × INT ON FIRST CHAR der Kanal für weitere INTERRUPT freigegeben werden. RESET T × INT PENDING: Im INTERRUPT-MODE 2 erscheint bei leerem Sendepuffer ein INTERRUPT. Dieses Kommando verhindert, daß dieser INTERRUPT mehrmals kommt. Nach dem Kommando tritt so lange kein INTERRUPT auf, bis wieder ein Zeichen in den Sendepuffer gekommen ist. ERROR RESET veranlaßt das Rücksetzen der Fehler-Bit (Parität und Empfängerüberlauf) in RR_1.

RETURN FROM INT erzeugt einen RETI-Befehl auf dem Datenbus.

D_6 bis D_7 setzt die CRC-Logik zurück.

WR_1:

D_0 EXT INTERRUPT ENABLE. Gestattet INTERRUPT bei Wechsel von $\overline{DCD}$, $\overline{CTS}$, $\overline{SYNC}$, bei BREAK, bei Sendebeginn von CRC oder SYNC-Zeichen.



D_1 TRANS INTERRUPT ENABLE. Erlaubt INT, wenn der Sendepuffer leer ist.

D_2 STATUS AFFECTS VECTOR (nur Kanal B). Ist dieser Modus ausgewählt, so wird der INTERRUPT-Vektor in WR_2 wie folgt modifiziert:

V_3	V_2	V_1		
0	0	0	Sendepuffer leer	Kanal B
0	0	1	externer Statuswechsel	
0	1	0	empfangenes Zeichen verfügbar	
0	1	1	spezielle Empfangsbedingung[10]	
1	0	0	Sendepuffer leer	Kanal A
1	0	1	externer Statuswechsel	
1	1	0	empfangenes Zeichen verfügbar	
1	1	1	spezielle Empfangsbedingung[10]	

D_3 bis D_4 Durch D_3 und D_4 werden spezielle INTERRUPT-Bildungsmöglichkeiten für Empfänger eingestellt.

D_5 $D_5 = 1$ Der $\overline{\text{WAIT-/READY}}$-Ausgang wird aktiv, wenn der Empfänger leer ist.
$D_5 = 0$ Der $\overline{\text{WAIT-/READY}}$-Ausgang wird aktiv, wenn der Sendepuffer voll ist.

D_6 $D_6 = 0$ $\overline{\text{WAIT-/READY}}$-Ausgang arbeitet als WAIT (Ausgang).
$D_6 = 1$ $\overline{\text{WAIT-/READY}}$-Ausgang arbeitet als READY (Ausgang).
WAIT-Funktion: Das SIO kann keine Daten aufnehmen oder abgeben.
READY-Funktion: Das SIO ist zur Arbeit bereit.

D_7 Freigabe des WAIT-/READY-Ausgangs.

WR_2: Dieses Register enthält den INTERRUPT-Vektor. Er existiert nur im Kanal B. Bei STATUS AFFECTS VECTOR werden die Bit-Stellen V_1 bis V_3 entsprechend der INTERRUPT-Erzeugung eingetragen.

WR_3:

D_0 Erlaubt Empfangsfunktion.

D_1 Bei $D_1 = 1$ werden die SYNC-Zeichen nicht in die Empfangspufferregister geladen. CRC erfolgt normal.

[10] Spezielle Empfangsbedingung: Paritätsfehler, Empfängerüberlauf, CRC-/Formatfehler, Blockende (SDLC).

D_2 $D_2 = 1$ verhindert im SDLC-Modus Interrupt, wenn keine Übereinstimmung zwischen empfangener und programmierter Adresse oder wenn keine Globaladresse (11111111) vorliegt.

D_3 Mit $D_3 = 1$ beginnt die CRC-Kontrolle beim Übergang des letzten Zeichens vom Empfangsregister zum Puffer.

D_4 HUNTMODE. Die Übertragung wird abgebrochen, bis ein neues SYNC-Zeichen kommt.

D_5 AUTOENABLE. Bei $D_5 = 1$ arbeiten die DCD- und CTS-Eingänge als Empfangs- und Sendesteuerung.
Bei $D_5 = 0$ wirken DCD und CTS nur auf ihre Bit im Statusregister.

D_6/D_7 Bestimmt empfangene Bit/Zeichen.

WR_4:

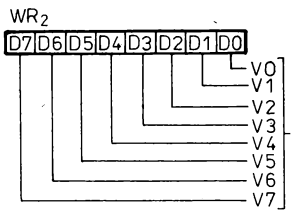
D_0 $D_0 = 1$, SIO arbeitet mit Paritäts-Bit,

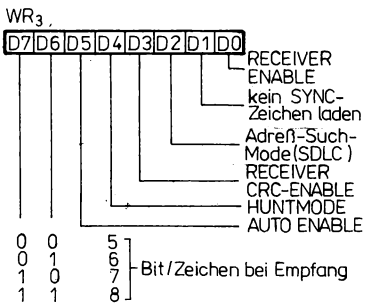
D_1 $D_1 = 0$, Parität ungerade,
$D_1 = 1$, Parität gerade.

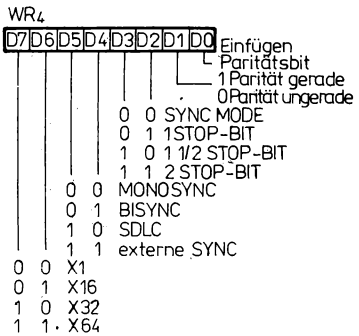
D_2/D_3 Bestimmt die Anzahl der Stop-Bit bei Senden.

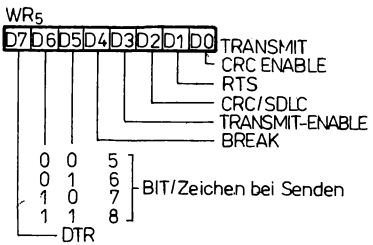
D_4/D_5 Bestimmt die Art des Synchronmodus.

D_6/D_7 Bestimmung des Multiplikationsfaktors zwischen Takt und Übertragungsrate.









WR_5:

D_0 TRANSMIT CRC-ENABLE. CRC-Freigabe bei Senden.

D_1 $D_1 = 1$, $\overline{RTS}$ geht auf 0.
$D_1 = 0$, $\overline{RTS}$ geht auf 1, wenn der Sender leer ist.

D_2 CRC/SDLC
$D_2 = 0$, CRC-Kontrolle mit SDLC-Polynom
$x^{16} + x^{12} + x^5 + 1$.
$D_2 = 1$, CRC-Kontrolle mit Polynom
$x^{16} + x^{15} + x^2 + 1$.

D_3 TRANSMIT ENABLE.
Sendeerlaubnis. Bei Rücksetzen wird das letzte Zeichen noch voll ausgegeben.

D_4 BREAK. Bewirkt Aussenden von »0« auf T × D.

D_5/D_6 Bestimmt auszugebende Bit/Zeichen.
Bei weniger als 5 Bit/Zeichen gilt:

1	1	1	1	0	0	0	D	sendet 1 Bit
1	1	1	0	0	0	D	D	sendet 2 Bit
1	1	0	0	0	D	D	D	sendet 3 Bit
1	0	0	0	D	D	D	D	sendet 4 Bit
0	0	0	D	D	D	D	D	sendet 5 Bit

D_7 DTR
$D_7 = 1$ bewirkt $\overline{DTR} = 0$ (aktiv).

WR_6 SYNC-Zeichen bei Monosynchronisation, die ersten 8 Bit bei Bisynchronisation.

WR_7 2. Zeichen bei Bisynchronisation oder das Synchronzeichen bei Monosynchronisation. Bei SDLC muß $WR_7 = 01111110$ sein.

RR_0:

D_0 RECEIVE CHARACTER AVAILABLE; gesetzt, wenn mindestens 1 Zeichen im Puffer.

D_1 INTERRUPT PENDING; wird bei jeder INT-Anmeldung gesetzt (nur Kanal A).

D_2 TRANSMIT BUFFER EMPTY; gesetzt, wenn der Sendepuffer leer ist (außer bei CRC).

D_3 DCD; zeigt Zustand $\overline{DCD}$.

D_4 SYNC/HUNT
$D_4 = 0$, Zeichensynchronisation erreicht,
$D_4 = 1$, keine Synchronisation.

D_5 CTS; zeigt Zustand $\overline{CTS}$.

D_6 TX UNDERRUN/EOM: Senderunterlauf bzw. Nachrichtenende.
BREAK/ABORT; wird gesetzt, wenn mit dem Senden von CRC-Worten begonnen wird, nachdem der Sendepuffer leer geworden ist.

D_7 CRC/SYNC
$D_7 = 1$ bei Erkennen von BREAK (bei ASYNC).
Im SDLC-Format bei Erkennen einer Abbruchfolge (7 oder mehr Einsen).

RR_1:

D_0 ALL SENT: Wird gesetzt, wenn alle Zeichen im Asynchronmode gesendet sind.

$D_1/D_2/D_3$ Falls bei Empfang im SDLC-MODE die Anzahl der Bit/Zeichen nicht aufgehen, zeigt $D_1/D_2/D_3$ die restlichen Bit an.

D_4 Paritätsfehler,

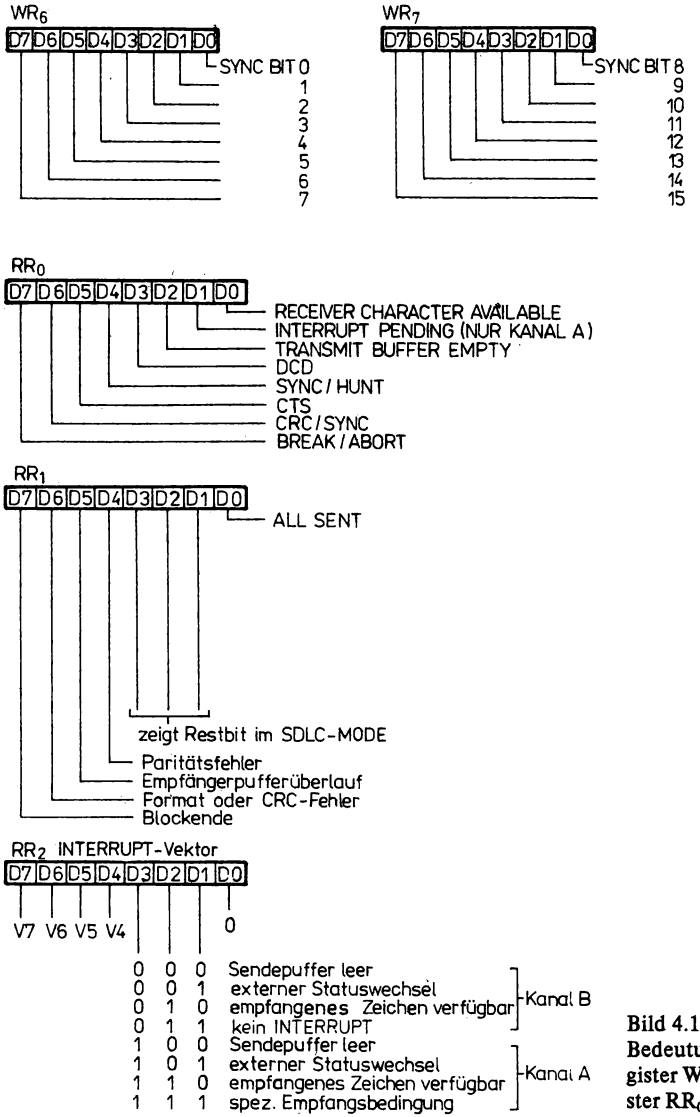


Bild 4.11
Bedeutung der Bitstellen der Steuerregister WR_0 bis WR_7 und der Statusregister RR_0 bis RR_2 im Baustein *U856D*

D_5 Empfängerüberlauf,

D_6 Formatfehler im Asynchronmode,
CRC-Fehler im Synchronmode.

D_7 Blockende bei SDLC mit gültigem CRC erkannt.

RR_2: Enthält INTERRUPT-Vektor. Die Bit sind entsprechend STATUS AFFECTS VECTOR verändert.

Der Baustein *U856* wird in mehreren Varianten produziert, die sich in der Taktfrequenz und den Anschlüssen unterscheiden. Bild 4.12 zeigt die unterschiedlichen Bondvarianten. Tabelle 4.1 gibt eine Übersicht zu den unterschiedlichen Varianten.

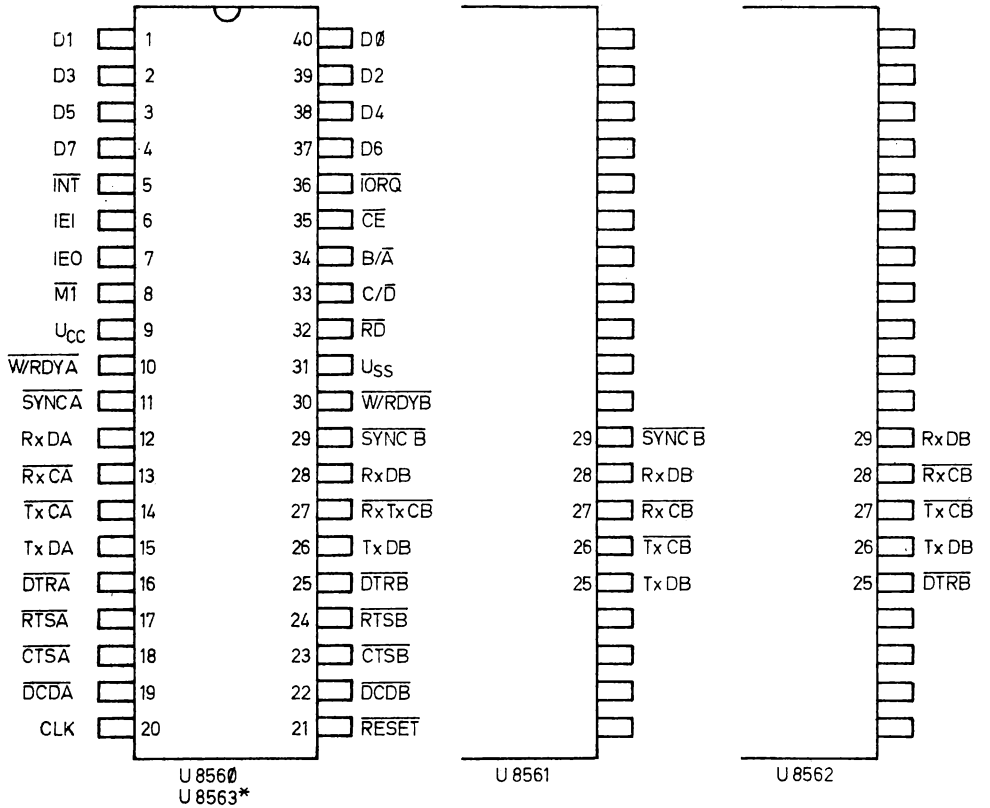


Bild 4.12
Bondvarianten des *U856* (Nicht gezeichnete bzw. unbezeichnete Anschlüsse sind mit der Variante *U8560* identisch)

4.5. DMA-Steuerbaustein *U858 D*

4.5.1. Funktion und Struktur

Der DMA-Baustein dient zur Steuerung der Übertragung von Daten von einer Funktionseinheit des Mikrorechners zu einer anderen. Als Funktionseinheiten können dabei periphere Einheiten oder der Arbeitsspeicher auftreten. Damit ist es möglich, mit Hilfe des DMA folgende Datenübertragungen durchzuführen:
- von einer peripheren Einheit zu einer anderen,
- von einem Bereich im Arbeitsspeicher zu einem anderen,
- von einer peripheren Einheit zum Arbeitsspeicher,
- vom Arbeitsspeicher zu einer peripheren Einheit.

Die Übertragung geschieht blockweise. Die Übertragungsgeschwindigkeit beträgt bis zu 1,25 Megabyte/s.

Tabelle 4.1.
Produzierte Varianten des U 856

Typ	Frequenz in MHz	Temperatur-bereich in °C	Bemerkung
UA 8560	4	0 bis 70	
UA 8561	4	0 bis 70	
UA 8562	4	0 bis 70	
UA 8563	4	0 bis 70	nur asynchroner Betrieb*)
UB 8560	2,5	0 bis 70	
UB 8561	2,5	0 bis 70	
UB 8562	2,5	0 bis 70	
UB 8563	2,5	0 bis 70	nur asynchroner Betrieb*)
VB 8560	2,5	−25 bis 85	
VB 8561	2,5	−25 bis 85	
VB 8562	2,5	−25 bis 85	

*) Variante U 8563 wird als DART (Dual Asynchroneous Receiver/Transmitter) bezeichnet.

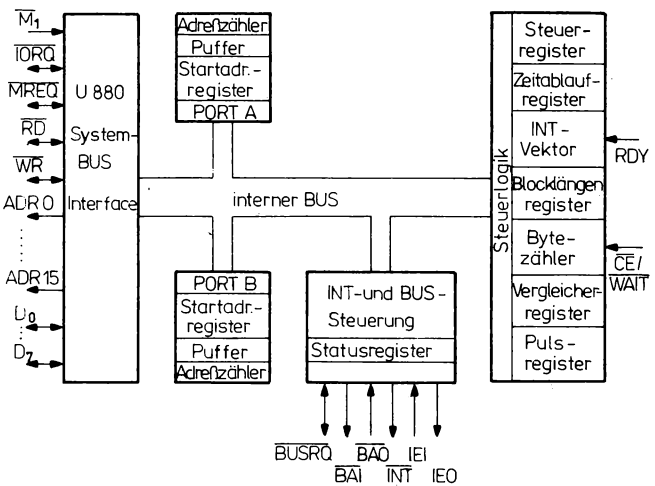


Bild 4.13
Grundstruktur des Bausteins *U858D*

Bild 4.13 zeigt die Grundstruktur des DMA-Bausteins. Der Baustein hat folgende Funktionseinheiten, die über einen internen Bus gekoppelt sind:
- 1 Interface mit Steuerung zum Systembus des Mikrorechners,
- 2 Tore; sie werden mit A und B gekennzeichnet.

Jedes dieser Tore kann die Steuerung einer Funktionseinheit (periphere Einheit oder Arbeitsspeicher) übernehmen. Es kann zum Lesen und zum Einschreiben benutzt werden.
- 1 Steuerlogik mit einem Registersatz,
- 1 INTERRUPT- und Bussteuerung.

Der DMA-Baustein hat folgende Register:
Steuerregister
Dieses Register enthält die notwendigen Steuerinformationen.

Zeitablaufregister
In diesem Register ist die Steuerinformation für die Art und Weise der Zyklensteuerung enthalten.
INTERRUPT-Vektorregister
Dieses Register enthält den INTERRUPT-Vektor.
Blocklängenregister
Dieses Register enthält die gesamte zu suchende oder zu übertragende Blocklänge.
Bytezähler
Die gelesenen Bytes werden im Bytezähler gezählt. Bei Übereinstimmung mit dem Blocklängenregister wird ein Bit im Statusregister gesetzt.
Vergleichsregister
Es enthält ein Byte, nach dem beim Suchvorgang gesucht wird.
Maskenregister
Dieses Register enthält eine Maske. Der Vergleich eines Bit zwischen gelesenem Byte und Vergleichsbyte findet statt, wenn im Maskenregister an der entsprechenden Stelle eine »0« steht.
Startadreßregister für Tor A und B
Sie enthalten die Startadressen (16 Bit) für die beiden Tore. Bei E/A-Operationen sind es nur 8 Bit.
Adreßzähler für Tor A und B
Diese Zähler enthalten die laufenden Adressen der beiden Tore (16 Bit).
Pulssteuerregister
Sobald die im Pulssteuerregister angegebene Anzahl Byte abgearbeitet ist, wird am Ausgang $\overline{INT}$ ein Impuls ausgegeben. Die CPU reagiert nicht, da durch $\overline{BUSRQ}$ der DMA aktiviert ist.
Statusregister
Dieses Register enthält die gesammelte Statusinformation.

4.5.2. Beschreibung der Anschlüsse

A_0 bis A_{15}	Adreßbus (Tri-state-Ausgang).
D_0 bis D_7	Datenbus (Tri-state, bidirectional).
$\overline{M1}$	Signal für M1-Zyklus (Eingabesignal, aktiv »Low«).
$\overline{IORQ}$	Eingabe-/Ausgabe-Anforderung (Ein- bzw. Ausgabesignal aktiv »Low«).
$\overline{MREQ}$	Speicheranforderung (Ein- bzw. Ausgang, aktiv »Low«).
$\overline{RD}$	Lesen (Ein- bzw. Ausgang, aktiv »Low«).
$\overline{WR}$	Schreiben (Ein- bzw. Ausgang, aktiv »Low«).
$\overline{CE/WAIT}$	(Eingang, aktiv »Low«); dient als Bausteinfreigabesignal. Es kann, wenn $\overline{BAI}$ = »Low« ist, auch als »WAIT« verwendet werden. Die Auswahl erfolgt durch Programm.
$\overline{BUSRQ}$	Busanforderung (Ein-/Ausgang, aktiv »Low«); Signal zur Anforderung einer DMA-Funktion.
$\overline{BAI}$	Busanforderungsbestätigung (Einzug, aktiv »Low«); zeigt an, daß die Buskontrolle an den DMA-Baustein übergeben wurde.
$\overline{BAO}$	Weitergabe $\overline{BAI}$ (Ausgang, aktiv »Low«); gibt $\overline{BAI}$-Signal an den nächsten Baustein weiter.

$\overline{INT}$	INTERRUPT-Anforderung (Ausgang, aktiv »Low«).
IEI	INTERRUPT-Freigabe (Eingang, aktiv »High«).
IEO	Weitergabe INTERRUPT-Freigabe (Ausgang, aktiv »High«).
RDY	Quittung (Eingang aktiv »High« oder »Low«, je nach Programmierung). RDY teilt dem DMA mit, ob die periphere Einheit zum Lesen oder Schreiben bereit ist.

4.5.3. Arbeitsweise

Busanforderung
Wenn RDY aktiv ist, erzeugt die DMA das Signal $\overline{BUSRQ}$. Die CPU aktiviert danach $\overline{BUSAK}$. Mit $\overline{BUSAK}$ muß $\overline{BAI}$ aktiv werden. Nach $\overline{BAI}$ beginnt mit der nächsten Taktflanke von T1 ein DMA-Zyklus. Nach der Busanforderung läuft die eingestellte Betriebsart ab.

Betriebsarten
Alle Betriebsarten können in den Funktionsarten Übertragung, Suchen, Übertragung und Suchen laufen. Es gibt folgende Betriebsarten:
Blockweise Die Übertragung ist abgeschlossen, wenn der Block übertragen oder das gesuchte Byte gefunden ist (unabhängig von RDY).
Andauernd (burst) Die Blockübertragung arbeitet nur, wenn RDY aktiv ist.
Transparent Die DMA-Operation wird während der Refresh-Zyklen ausgeführt.
Byteweise Die CPU übernimmt die Buskontrolle nach jeder Einzelbyteoperation.

Blockübertragung
Bei Speicherzugriff steuert die DMA einen Speicherlesezyklus und speichert das Byte im Lesetor. Danach erfolgen bei »Übertragung« die Übergabe zum Schreibtor und ein Speicherschreibzyklus. Bei »Suchen« kommt es zum Vergleich mit dem Byte im Vergleicherregister, und bei Gleichheit wird ein Status-Bit gesetzt. Adreß- und Bytezähler werden dabei erhöht. Bei Ein-/Ausgabe laufen Ein- bzw. Ausgabezyklen ab. Die Zyklenlänge läßt sich programmieren, jedoch muß die Mindestlänge analog den CPU-Zyklen eingehalten werden. Durch die Verlängerung spart man eine zusätzliche WAIT-Logik. Nach RESET arbeiten die Zyklen analog den CPU-Zyklen.
Die Busfreigabe geschieht bei »Übertragung« nur am Blockende, bei »Suchen« bei Gleichheit oder am Blockende. Bei Gleichheit wird vor der Busfreigabe noch eine Byte-Operation ausgeführt.
Andauernde Übertragung
Der Ablauf in dieser Betriebsart unterscheidet sich von der »Blockübertragung« dadurch, daß es während RDY = »0« zu einer Busfreigabe kommt.

Byteweise Übertragung
Die Busfreigabe ist unabhängig von RDY nach jedem Lesezyklus bei »Suchen« und nach jedem Schreibzyklus bei »Übertragen«. Nachdem $\overline{BUSRQ}$ inaktiv ist, kann ein erneutes $\overline{BUSRQ}$ erst kommen, wenn vorher $\overline{BAI}$ inaktiv geworden ist.

Programmierung des DMA (Initialisierung)
Das Einstellen der Funktion des DMA geschieht durch eine Reihe Steuerbytes, mit deren

Hilfe die einzelnen Register gefüllt werden. Die unterschiedlichen Steuerbytes sind durch die Bit-Stellen D_0, D_1 und D_7 gekennzeichnet. Die Steuerbytes werden in Steuerregister gespeichert. Sie werden 1A, 1B, 2A, 2B, 2C, 2D genannt.

Kennzeichnung der Steuerbytes:

	D_7	D_1	D_0
1A	0	≠ 0	≠ 0
1B	0	0	0
2A	1	0	0
2B	1	0	1
2C	1	1	0
2D	1	1	1

Übersicht über die Steuerbytes

Steuerbyte 1A

D_7 0

D_6 Blocklänge, »High«-Byte folgt

D_5 Blocklänge, »Low«-Byte folgt

D_4 Adresse A, »High«-Byte folgt

D_3 Adresse A, »Low«-Byte folgt

D_2 0B → A; 1A → B

D_1 D_0 $\begin{matrix}0\\0\end{matrix}$ – ; $\begin{matrix}0\\1\end{matrix}$ Transfer; $\begin{matrix}1\\0\end{matrix}$ Suche; $\begin{matrix}1\\1\end{matrix}$ Suche + Transfer

Steuerbyte 1B

D_7 0

D_6 Zeitablaufbyte folgt

D_5 Toradresse ist fest

D_4 0 Toradresse wird decrementiert
1 Toradresse wird incrementiert

D_3 D_2 $\begin{matrix}0\\0\end{matrix}$ B = Speicher; $\begin{matrix}0\\1\end{matrix}$ A = Speicher; $\begin{matrix}1\\0\end{matrix}$ B = E/A; $\begin{matrix}1\\1\end{matrix}$ A = E/A

D_1 0

D_0 0

Zeitablaufbyte

D_7 0 = $\overline{WR}$-Ende ½ Zyklus früher

D_6 0 = $\overline{RD}$-Ende ½ Zyklus früher

D_5 frei

D_4 frei

D_3 0 = $\overline{MREQ}$-Ende ½ Zyklus früher

D_2 0 = $\overline{IORQ}$-Ende ½ Zyklus früher

D_1 D_0 $\begin{matrix}0\\0\end{matrix}$ Zykluslänge = 4; $\begin{matrix}0\\1\end{matrix}$ Zykluslänge = 3; $\begin{matrix}1\\0\end{matrix}$ Zykluslänge = 2

Bei »Übertragung« muß das Zeitablaufbyte zweimal folgen, erst für Tor A, dann für Tor B.

Steuerbyte 2A

D_7 1
D_6 DMA-Freigabe
D_5 INTERRUPT-Freigabe
D_4 Vergleichsbyte folgt
D_3 Maskenbyte folgt
D_2 Stop bei Vergleich
D_1 0
D_0 0

Steuerbyte 2B

D_7 1
D_6, D_5: Betriebsart; 0 0 byteweise; 0 1 blockweise; 1 0 burst; 1 1 –
D_4 INTERRUPT-Steuerbyte folgt
D_3 Adresse B, »High«-Byte folgt
D_2 Adresse B, »Low«-Byte folgt
D_1 0
D_0 1

INTERRUPT-Steuerbyte

D_7 frei
D_6 INTERRUPT vor BUSRQ
D_5 Status verändert INTERRUPT-Vektor
D_4 INTERRUPT-Vektor folgt
D_3 Pulszählerbyte folgt
D_2 Pulse werden erzeugt
D_1 INTERRUPT bei Blockende
D_0 INTERRUPT bei Gleichheit

INTERRUPT-Vektor

D_7 V_7
D_6 V_6
D_5 V_5
D_4 V_4
D_3 V_3
D_2, D_1: 0 0 bei INTERRUPT auf RDY; 0 1 bei INTERRUPT auf Gleichheit; 1 0 bei INTERRUPT auf Blockende; 1 1 bei INTERRUPT auf Gleichheit oder Blockende
D_0 0

Steuerregister 2C

D_7 1
D_6 frei
D_5 0 Stop bei Blockende; 1 wiederholen bei Blockende
D_4 0 $\overline{CE}$ nur; 1 $\overline{CE/WAIT}$

D_3 0 RDY aktiv »Low«; 1 RDY aktiv »High«
D_2 frei
D_1 frei
D_0 0

Steuerregister 2D

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	0				Chip zurücksetzen;
1	0	0	0	1				gleicht Zeitverhalten DMA-Tor A an CPU an;
1	0	0	1	0				gleicht Zeitverhalten DMA-Tor B an CPU an;
1	0	0	1	1				0 → Bytezähler, laden Startadressen[11];
1	0	1	0	0				Ausführung wird bei momentanem Adreßstand und voller Blocklänge begonnen (Restart);
0	1	0	1	0				INTERRUPT-Freigabe;
0	1	0	1	1				INTERRUPT sperren;
0	1	0	0	0				alle INTERRUPT werden zurückgesetzt;
0	0	0	0	1				Chipfreigabe, Operationen werden freigegeben;
0	0	0	0	0				Chipsperre, Operationen werden blockiert;
0	1	1	1	0				Lesemaske folgt;
0	1	0	0	1				das erste Register, das durch Lesemaske frei ist, wird als nächstes gelesen;
0	1	1	1	1				als nächstes Register wird das Statusregister gelesen;
0	1	1	0	0				erzwungenes RDY, es wird intern ein RDY erzeugt;
0	1	1	0	1				vom DMA kommt keine Busanforderung, solange das RETI nicht kommt;
0	0	0	1	0				Rücksetzen Vergleichs-Bit und Blockende-Bit im Statusregister.

Statusregister
D_7 frei
D_6 frei
D_5 Blockende-Bit, 0 = Blockende
D_4 Vergleichs-Bit, 0 = Vergleich
D_3 0 INT-Pending
D_2 frei
D_1 0 RDY = Aktiv
D_0 0 DMA-Zyklus nicht erschienen
1 DMA-Zyklus ist erschienen

Lesen der DMA-Register
Folgende DMA-Register können sequentiell in der aufgezählten Reihenfolge gelesen werden:

[11] Bekommt der betreffende Kanal eine Festadresse, so ist das Laden der Startadresse (für Festadresse) nur möglich, wenn dieser Kanal als Eingabekanal definiert ist.

Statusregister,
Bytezähler niederwertiger Teil;
Bytezähler höherwertiger Teil,
Adresse A niederwertiger Teil,
Adresse A höherwertiger Teil,
Adresse B niederwertiger Teil,
Adresse B höherwertiger Teil.
Das Weiterschalten von Register zu Register geschieht intern. Soll ein Register nicht gelesen werden, so kann es durch Setzen von »0« an der entsprechenden Stelle der Lesemaske übersprungen werden. Nach RESET und Lesen Statusregister weist der interne Zeiger immer auf das Statusregister.

Lesemaske:
D_7
D_6 Adresse B HWT
D_5 Adresse B NWT
D_4 Adresse A HWT
D_3 Adresse A NWT
D_2 Bytezähler HWT
D_1 Bytezähler NWT
D_0 Statusregister

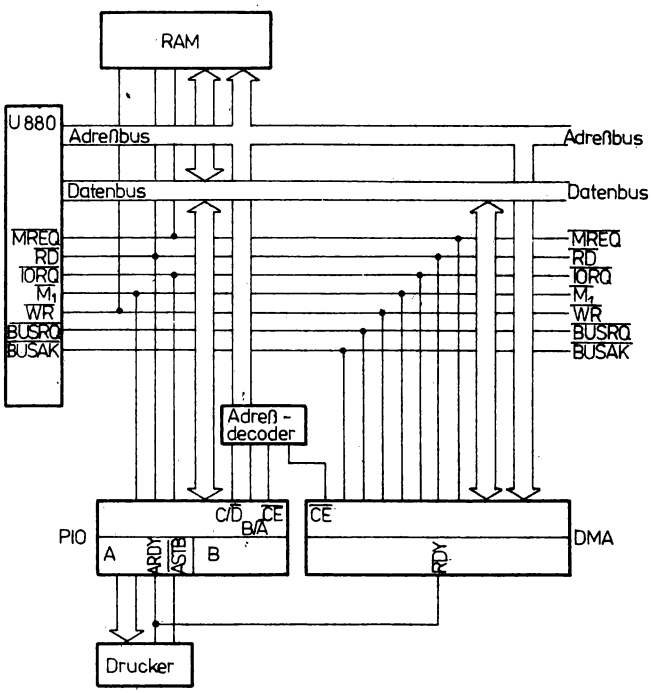


Bild 4.14
Anschluß eines DMA-Bausteins an den Systembus *U880*

4.5.4. Beispiel zum DMA-Baustein

Ein DMA-Baustein soll die Ausgabe eines Datenblocks aus dem Speicher auf einen über ein PIO angeschlossenen Drucker steuern. Bild 4.14 zeigt die notwendigen Verbindungen zwischen *U880*, PIO und DMA. Der Drucker ist am Kanal A des PIO angeschlossen und arbeitet im *Hand-shake*-Betrieb mit den Signalen ARDY und ASTB, Kanal A des PIO wird in die Betriebsart »Ausgabe« initialisiert. INTERRUPTS werden gesperrt. Der DMA wird in die Betriebsart »Byteweise« und in die Funktionsart »Übertragung« initialisiert. Kanal A soll den Speicher, Kanal B das PIO steuern. Solange am PIO das Signal ARDY aktiv ist (Daten sind nicht vom Drucker abgeholt), soll der DMA den Bus freigeben. Dazu wird ARDY mit RDY des DMA verbunden und für RDY aktiv »Low« festgelegt. Durch den Adreßdecoder sollen folgende Adressen festgelegt werden:

PIO	Datenwort	Kanal A	Adresse 00 H
	Steuerwort	Kanal A	Adresse 40 H
	Datenwort	Kanal B	Adresse 80 H
	Steuerwort	Kanal B	Adresse C2 H
DMA	Bausteinauswahl		Adresse 30 H

Zur Initialisierung der Übertragung vom Speicher zum Drucker müssen folgende Bit-Muster von der CPU *U880* ausgegeben werden:

	Befehl	Wort auf Datenbus
Ausgabe Steuerwort zur Festlegung der Betriebsart »Ausgabe« im PIO Kanal A	OUT 40 H	00001111
INTERRUPT-Sperre für PIO Kanal A	OUT 40 H	00000011
Steuerbyte 1A für DMA Übertragung B → A Da Kanal B mit einer festen Adresse versehen wird, muß er erst als Eingabekanal definiert werden.	OUT 30 H	00000001
Steuerbyte 1B für DMA Kanal B Kanal B zur Peripheriesteuerung mit fester Adresse	OUT 30 H	00101000
Steuerbyte 2B Byteweise Übertragung Festlegen Adresse für Kanal B	OUT 30 H	10000101
Adresse PIO Datenkanal A zum DMA Kanal B	OUT 30 H	PIO-Adresse
Steuerbyte 2D Laden Toradresse B	OUT 30 H	11001111
Steuerbyte 1A für DMA	OUT 30 H	01111101

Übertragung A → B		
Kanal A »Low«-Byte	OUT 30H	»Low«-Byte
Kanal A »High«-Byte	OUT 30H	»High«-Byte
Blocklänge »Low«-Byte	OUT 30H	»Low«-Byte
Blocklänge »High«-Byte	OUT 30H	»High«-Byte
Steuerbyte IB für DMA-Kanal A Kanal steuert Speicher, Adresse wird incrementiert, kein Zeitsteuerbyte	OUT 30H	00010100
Steuerbyte 2C RDY aktiv »Low« kein WAIT-Status	OUT 30H	10000010
Steuerbyte 2D Laden Toradresse A Löschen Blockzähler	OUT 30H	11001111
Steuerbyte 2D Starten DMA-Operation	OUT 30H	10000111

5. Periphere Schaltkreise zum *8080*

5.1. Serieller Ein-/Ausgabebaustein *8251*

5.1.1. Funktion

Ein neben dem SIO häufig verwendeter Baustein zur Umsetzung von paralleler in serielle Information und umgekehrt ist der Schaltkreis *8251* (USART = Universal Synchronous Asynchronous Receiver Transmitter). Er hat nicht so viele Möglichkeiten wie der SIO-Baustein, ist aber für viele Anwendungsfälle ausreichend, speziell für den Anschluß peripherer Geräte über serielle Schnittstellen. Mit dem *8251* lassen sich folgende Betriebsarten realisieren:

Asynchron (Bild 5.1)
- 5 bis 8 Zeichen
- Taktuntersetzung × 1, × 16 oder × 64
- Erzeugung BREAK
- 1, 1½ oder 2 Stopbits

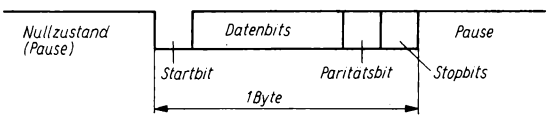
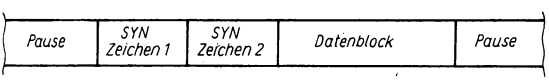


Bild 5.1
Asynchrones Datenprotokoll

Synchron (Bild 5.2)
- 5 bis 8 Zeichen
- 1 oder 2 Synchronisationszeichen
- externe Synchronisation

(jedoch kein SDLC und keine CRC-Prüfung).



- Es sind ein- oder zwei Synchronisationszeichen möglich
- In der Pause werden die Synchronisationszeichen übertragen
- Es existiert nur 1 Kanal für serielle Sende- und Empfangsdaten

Bild 5.2
Synchrones Datenprotokoll

5.1.2. **Anschlußbelegung** (Bild 5.3)

Beschreibung der Anschlüsse

D_0 bis D_7 Datenbus (Bidirektional).

$C/\overline{D}$ Unterscheidung Steuerwort – Datenwort
1 = Steuerwort, 0 = Datenwort.

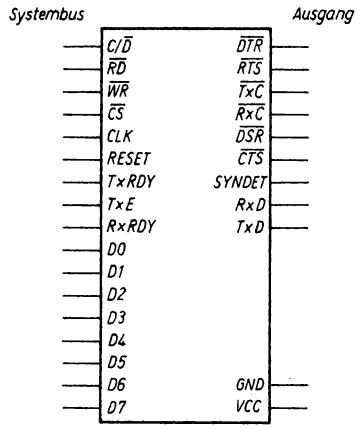


Bild 5.3
Anschlußbelegung des *8251*

$\overline{RD}$ aus dem Baustein Daten lesen (aktiv »Low«).

$\overline{WR}$ in den Baustein Daten schreiben (aktiv »Low«).

$\overline{CS}$ Bausteinfreigabe (aktiv »Low«).

CLK Takteingang.

RESET Rücksetzen der Betriebsarteneinstellung (aktiv »High«).

T × RDY Sender ist zur Aufnahme eines Bytes bereit, wird beim Laden eines Zeichens zurückgesetzt (aktiv »High«).

T × E kein Zeichen zum Senden im Baustein (Sender leer), wird beim Empfang eines Zeichens vom Prozessor zurückgesetzt (aktiv »High«).

R × RDY Baustein hat ein Zeichen empfangen, wird beim Lesen des Zeichens zurückgesetzt (aktiv »High«).

$\overline{DTR}$ (Data Terminal Ready).
$\overline{DTR}$ ist Ausgangssignal und läßt sich durch Programm einstellen (aktiv »Low«).

$\overline{RTS}$ (Request To Send).
$\overline{RTS}$ ist Ausgangssignal und läßt sich durch Programm einstellen (aktiv »Low«).

$\overline{CTS}$ (Clear To Send).
$\overline{CTS}$ ist Eingangssignal und läßt sich vom Programm testen (aktiv »Low«).

$\overline{DSR}$ (Data Set Ready).
$\overline{DSR}$ ist Eingangssignal und läßt sich vom Programm testen (aktiv »Low«).

T × C Sendetakt.

R × C Empfangstakt.

SYNDET (Synchronisation Detect)
SYNDET läßt sich als Ausgang oder Eingang programmieren, es ist bei externer Synchronisation »Eingang« und erklärt die Gültigkeit der empfangenen Information;
der High-Pegel kann nach 1–2 Taktperioden abgeschaltet werden;
bei interner Synchronisation ist es Ausgang, nach Erkennen des Synchronisationszeichens wird SYNDET »High« und nach Zustandslesen »Low« (aktiv »High«).

R × D Empfangseingang.

T × D Sendeausgang.

5.1.3. Logische Struktur (Bild 5.4)

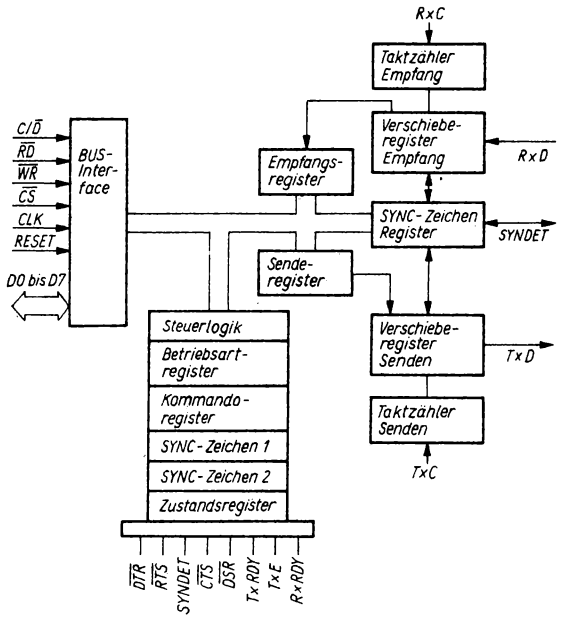


Bild 5.4
Logische Struktur des *8251*

5.1.4. Arbeitsweise

Das Senden eines Bits erfolgt mit der H/L-Flanke von T × D. Mit der L/H-Flanke von R × C werden Empfangsleitungen getestet.
Für Kurzschlußbetrieb (Sender mit Empfänger verbunden und T × C mit R × C verbunden) würde sich Bild 5.5 zum Senden und Empfangen eines Bit-Musters ergeben.
Im Asynchronbetrieb wird ein Datenprotokoll nach Bild 5.1 erzeugt. Zum Senden und Empfangen läßt sich eine Untersetzung × 16 oder × 64 programmieren. Damit wird nur jeder 16. oder 64. Takt zur Übertragung ausgewertet.
Im Synchronbetrieb wird ein Datenprotokoll nach Bild 5.2 erzeugt. Eine Untersetzung des Taktes T × C oder R × C ist nicht möglich. In der Betriebsart interne Synchronisation werden in der Pause SYNC-Zeichen gesendet. Werden die SYNC-Zeichen im Empfänger erkannt, wird das Signal SYNDET eingeschaltet. Bei externer Synchronisation erfolgt die Synchronisation mit SYNDET. Ein Eingangsimpuls an SYNDET gibt den Beginn der Übertragung an.

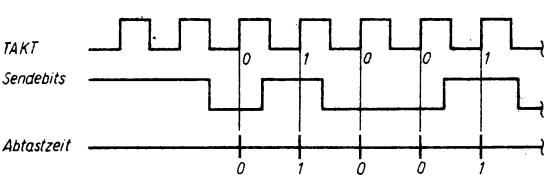


Bild 5.5
Senden und Empfangen von Bit mit dem *8251*

5.1.5. Programmierung

Zur Programmierung des *8251* ist folgende Reihenfolge einzuhalten:

1. Einstellen der Betriebsart (Bild 5.6).
2. Eingabe der SYNC-Zeichen, wenn die entsprechende SYNC-Betriebsart eingestellt ist. Es werden nur die SYNC-Zeichen angenommen, die von der eingestellten Betriebsart verlangt werden.
3. Alle nun folgenden Steuerworte sind Kommandos. Jede neue Eingabe in den *8251* ändert das Kommando. Eine Rückkehr zum Anfang erfolgt nur mit dem Rücksetzbit im Kommando. Kommandos ($C/\overline{D} = 1$) und Datenworte ($C/\overline{D} = 0$) können sich abwechseln. Bild 5.7 zeigt das Kommandowort.

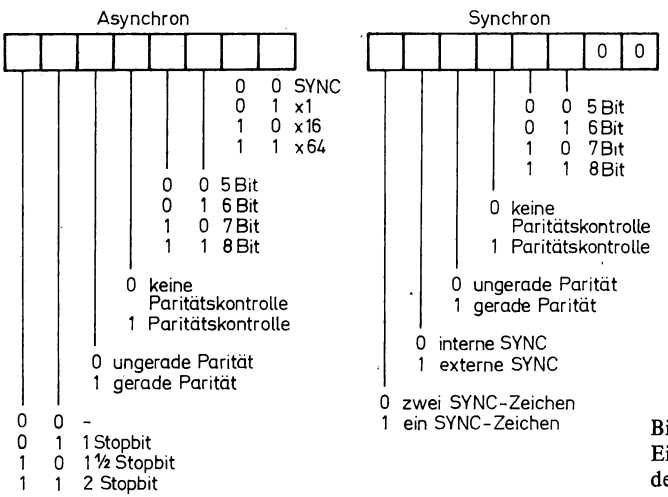


Bild 5.6
Einstellung der Betriebsarten des *8251*

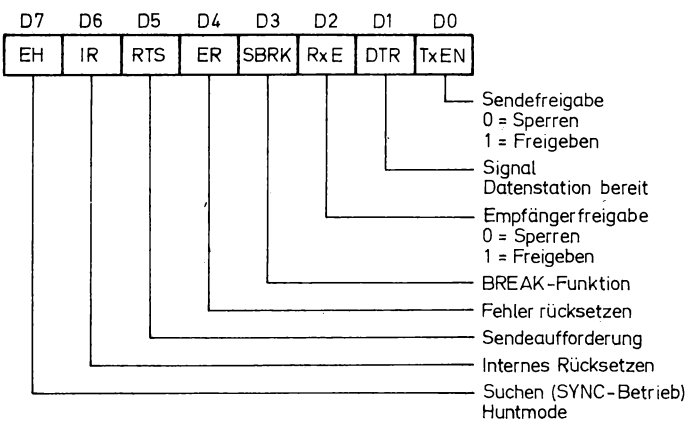


Bild 5.7
Kommandowort des *8251*

5.2. Paralleler Ein- und Ausgabebaustein *8255*

5.2.1. Logische Struktur

Der Interfacebaustein *8255* (Bild 5.8) ermöglicht die parallele Ein-/Ausgabe. Er hat 3 Ein-/Ausgabetore (A, B und C) zu je 8 Bit. Tor A und B sind frei programmierbar. Die Anschlüsse von Tor C werden teilweise als Status- und Steuersignale für Tor A und B benutzt.

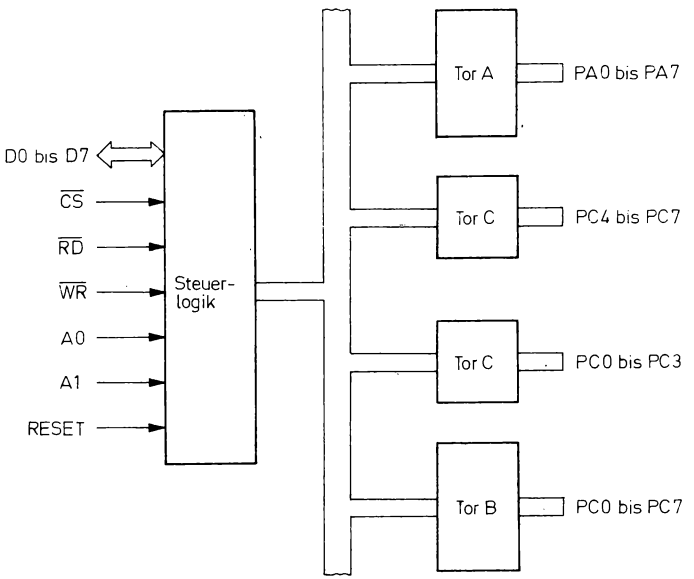


Bild 5.8
Übersichtsschaltplan des *8255*

5.2.2. Anschlußsignale

D_0 bis D_7	sind die Datenleitungen zum Prozessor
$\overline{CS}$	(Chip Select) Bausteinauswahlsignal
$\overline{RD}$	Freigabe zu Lesen von Daten aus dem *8255*
$\overline{WR}$	Einschreiben von Daten in den *8255* vom BUS
A1 A0	Torauswahl 00 = Tor A Daten 01 = Tor B Daten 10 = Tor C Daten 11 = Steuerwort zum 8255
RESET	Löscht alle internen Register und setzt die Tore in den »Input MODE«
PA_0 bis PA_7	Datenleitungen Tor A
PC_0 bis PC_7	Datenleitungen Tor C
PB_0 bis PB_7	Datenleitungen Tor B

5.2.3. Arbeitsweise

Der Schaltkreis kann in 3 Betriebsarten arbeiten:

MODE 0 Einfache Ein-/oder Ausgabe über alle 3 Tore ohne Handshake

MODE 1 Eingabe oder Ausgabe über Tor A oder B. Leitungen von Tor C nutzt man als Steuersignale

MODE 2 Bidirektionale Ein-/Ausgabe nur über Tor A.
Die Leitungen von Tor C dienen als Steuersignale

Die Betriebsart läßt sich mit dem Steuerwort 1 (Bild 5.9) einstellen. Durch eine zusätzliche Operation »Bit-set/Reset« können die Leitungen von TOR C als Signalleitungen verwendet werden, in denen einige als Ausgabe- und einige als Eingabeleitungen definiert sind. Das zugehörige Steuerwort 2 (Bild 5.10) lautet:

MODE 0

In MODE 0 kann die Eingabe oder Ausgabe entsprechend der Einstellung durch das Steuerwort erfolgen. Es gibt keine Handshakesignale.

MODE 1

Ein-/Ausgabe mit Handshake. Tor A und B arbeiten entsprechend der Einstellung im Steuerwort. Für die Ein-/ Ausgabe existiert ein Eingabe- bzw. Ausgabepufferregister.
Einige Leitungen von Tor C dienen als Steuersignale mit folgenden Funktionen:

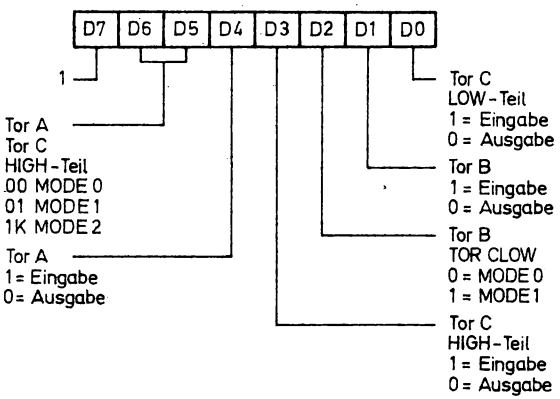


Bild 5.9
Aufbau des Steuerwortes 1 des PPI *8255*

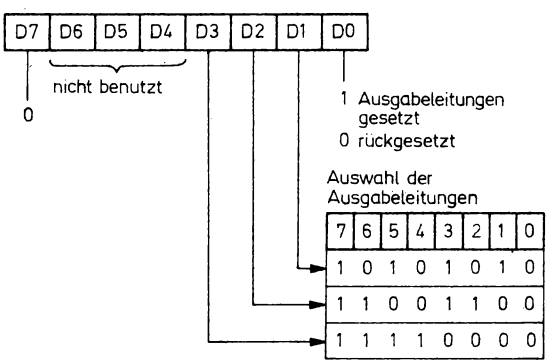


Bild 5.10
Aufbau des Steuerwortes 2 des PPI *8255*

PC_0	INTR	bei Eingabe und Ausgabe für Kanal B
PC_1	IBF	bei Eingabe für Kanal B
	$\overline{OBF}$	bei Ausgabe für Kanal B
PC_2	$\overline{STB}$	bei Eingabe für Kanal B
	$\overline{ACK}$	bei Ausgabe für Kanal B
PC_3	INTR	bei Eingabe und Ausgabe für Kanal A
PC_4	$\overline{STB}$	bei Eingabe für Kanal A
PC_5	IBF	bei Eingabe für Kanal A
PC_6	$\overline{ACK}$	bei Ausgabe für Kanal A
PC_7	$\overline{OBF}$	bei Ausgabe für Kanal A

Dabei bedeuten:

INTR (INTERRUPT REQUEST – Ausgabesignal) INTERRUPT-Anforderung an den Prozessor

IBF (Input Buffer Full – Ausgabesignal) Eingabepuffer voll

$\overline{OBF}$ (Output Buffer Full – Ausgabesignal) Ausgabepuffer voll

$\overline{STB}$ (Strobe – Eingabesignal) Low schreibt Daten in den Eingabepuffer

$\overline{ACK}$ (Acknowledge – Eingabesignal) Daten wurden von peripherem Gerät aus dem Ausgabepuffer gelesen

MODE 2

MODE 2 realisiert bidirektional Ein- und Ausgabe über Tor A. Folgende Leitungen von Tor C dienen als Steuerleitungen

PC_3	INTR	(INTERRUPT REQUEST – Ausgabesignal)
PC_4	$\overline{STB}$	(Strobe – Eingabesignal) Low-Pegel schreibt die Daten vom peripheren Gerät in den Eingabepuffer.
PC_5	IBF	(Input Buffer Full – Ausgabesignal) Eingabepuffer voll
PC_6	$\overline{ACK}$	(Acknowledge – Eingabesignal) Low gibt die Daten vom Ausgangspuffer auf den bidirektionalen Datenbus. Sonst sind die Leitungen hochohmig.
PC_7	$\overline{OBF}$	(Output Buffer Full – Ausgabesignal) Ausgabepuffer voll

Tabelle 5.1
Zuordnung der Leitungen von Tor C in den Betriebsarten MODE 1 und MODE 2

	MODE 1 Eingang	MODE 1 Ausgang	MODE 2
PC0	$INTR_B$	$INTR_B$	frei
PC1	$\overline{IBF}_B$	$\overline{OBF}_B$	frei
PC2	$\overline{STB}_B$	$\overline{ACK}_B$	frei
PC3	$INTR_A$	$INTR_A$	INTR
PC4	STB_A	frei	$\overline{STB}$
PC5	IBF_A	frei	IBF
PC6	frei	$\overline{ACK}_A$	$\overline{ACK}$
PC7	frei	$\overline{OBF}_A$	$\overline{OBF}$

Tabelle 5.1. zeigt die vollständige Zuordnung der Leitungen von Tor C in MODE 1 und MODE 2.
Die freien Leitungen können durch die Funktion »Bit set/Bit reset« benutzt werden.

INTERRUPT
Durch die Signale STB und ACK wird eine INTERRUPT-Anforderung an den Prozessor über die Leitungen INTR erzeugt. Diese Anforderung kann getrennt für Ein- und Ausgabe maskiert werden. Zur Freigabe ist eine Bit-set-Operation für ein entsprechendes Bit im Tor C nötig. Durch eine Bit-reset-Operation wird die entsprechende Interruptanforderung gesperrt. Für die Zuordnung der Interruptanforderungen zu den Bitstellen von Tor C gilt:

MODE 1	Eingabe	$INTR_A$	PC_4
MODE 1	Eingabe	$INTR_B$	PC_2
MODE 1	Ausgabe	$INTR_A$	PC_6
MODE 1	Ausgabe	$INTR_B$	PC_2
MODE 2	Eingabe	INTR	PC_4
MODE 2	Ausgabe	INTR	PC_6

6. Allgemeine Busschaltkreise

6.1. Überblick

Außer den Schaltkreisen, die bestimmte Funktionen der Peripheriesteuerung übernehmen, benötigt man zum Aufbau eines Mikrorechners Schaltkreise zur Bussteuerung (Steuerung der Richtung des Datenbus, Zwischenspeicherung von Adresse und Steuersignalen) und zur Entschlüsselung (Adreßentschlüsselung, Prioritätsentschlüsselung).

6.2. Bussteuerschaltkreise

6.2.1. Registerschaltkreis *8212* (Bild 6.1)

Der Schaltkreis *8212* arbeitet in 2 Betriebsarten, die durch MD (Mode) gesteuert werden.
Betriebsart 1: Eingang MD hat H-Pegel
Die Information der einzelnen Registerstellen kann direkt an den Ausgängen DO1 bis DO8 abgenommen werden. Wenn $\overline{DS1}$ L-Pegel und DS2 H-Pegel hat, dann wird die Information von den Eingängen DI1 bis DI8 in die D-Flip-Flop eingegeben.
Betriebsart 2: Eingang MD hat L-Pegel
Mit H-Pegel an STR (Strobe) wird die Information in die D-Flip-Flop gespeichert.
Wenn $\overline{DS1}$ L-Pegel und DS2 H-Pegel führt, so kann die Information, die im Register gespeichert ist, an den Ausgängen DO1 bis DO8 abgenommen werden.
Im Fall 2 setzt außerdem das Signal STR mit der HL-Flanke das Flip-Flop SRFF zurück und damit den $\overline{INT}$-Ausgang auf L-Pegel. Durch $\overline{DS1} \cdot DS2$ wird das Flip-Flop SRFF wieder gesetzt. Die Leitung $\overline{INT}$ dient als INTERRUPT-Anforderung für Mikroprozessorschaltkreise. Durch L-Pegel am Eingang $\overline{CLR}$ (Löschen) werden die D-Flip-Flop des Registers rückgesetzt und das Flip-Flop SRFF gesetzt ($\overline{INT}$ = H, d. h. nicht aktiv). Die Ausgänge DO1 bis DO8 sind mit einer Tri-state-Steuerung versehen, d. h., wenn die Ausgangsleitungen gesperrt sind, ist ihr Ausgang hochohmig.

Zusammenstellung der Funktionen des Bausteins *8212*

MD = H: $DE \cdot \overline{DS1} \cdot DS2 \rightarrow$ Registerinhalt
DA = Registerinhalt
MD = L: $DE \cdot STR \rightarrow$ Registerinhalt, DA = Registerinhalt $\cdot \overline{DS1} \cdot DS2$
$STR\,(H \rightarrow T) \rightarrow \overline{SRFF} \rightarrow$ INTERRUPT-Anforderung
$\overline{CLR}$ = L: Löschen Register und Setzen SRFF
Das INT-Signal ist aktiv ($\overline{INT}$ = L), wenn $INT = \overline{DS1} \cdot DS2 \vee \overline{SRFF} = 1$ ist.

Anwendung des Bausteins *8212*

Der Baustein *8212* läßt sich sehr universell für die Bussteuerung und Datenpufferung anwenden. Bild 6.2 zeigt den Einsatz des Bausteins als Datenpuffer. Die Eingabe in das Register

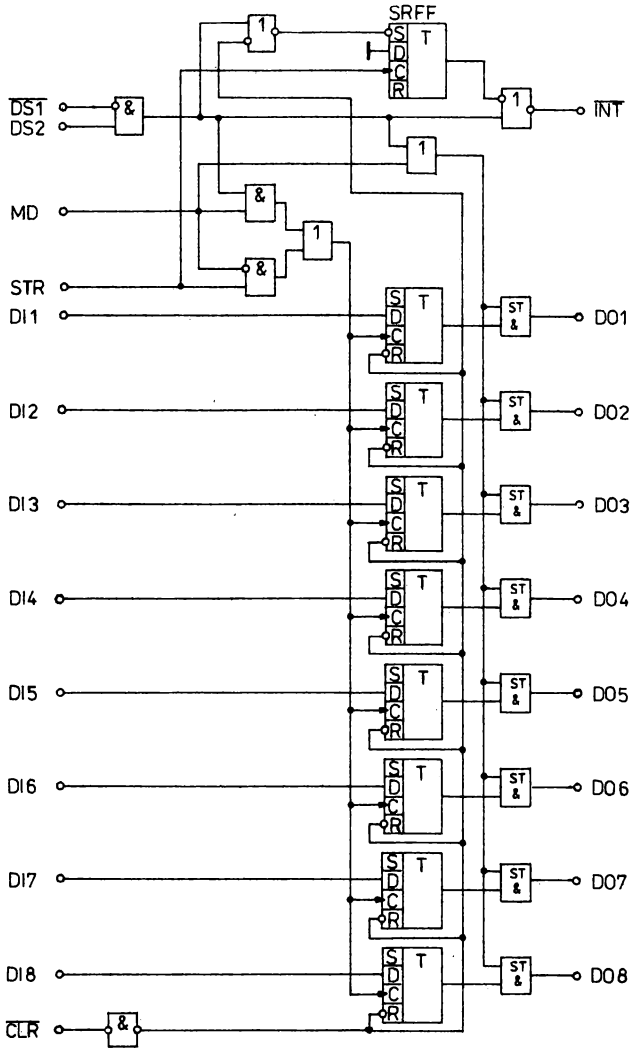


Bild 6.1
Innenschaltung des Bausteins *8212*

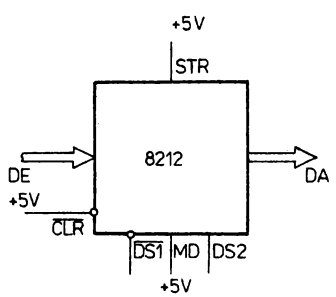


Bild 6.2
Der Baustein *8212* als Datenpuffer

geschieht mit $\overline{DS1} \cdot DS2 = H$. Die Ausgabe aus dem Register ist jederzeit möglich.
Bild 6.3 zeigt die Verschaltung des Bausteins *8212* als Eingabepuffer. Die Eingabe in das Register läßt sich über das Signal STR vornehmen. Das Signal STR aktiviert gleichzeitig das INT-Signal. Damit wird angezeigt, daß der Datenpuffer voll ist.
Die Ausgabe beginnt in dem Moment, wo $\overline{DS1} \cdot DS2 = H$ ist. Mit diesem Signal wird gleichzeitig das SRFF-Flip-Flop gesetzt. Das INT-Signal verschwindet jedoch erst mit der HL-Flanke von $\overline{DS1} \cdot DS2$.
Bild 6.4 zeigt die Verschaltung des Bausteins *8212* im Hand-shake-Prinzip für einen Ausgabepuffer. Die Eingabe ins Register geschieht mit dem Signal $\overline{DS1} \cdot DS2 = H$. Der Inhalt des Registers liegt ständig an DA. Den zur Abnahme des Datenwortes verwendeten Strobeimpuls führt man gleichzeitig an den Strobeeingang, wodurch das INT-Signal aktiv wird (als Rückmeldung, daß die Daten abgeholt wurden).

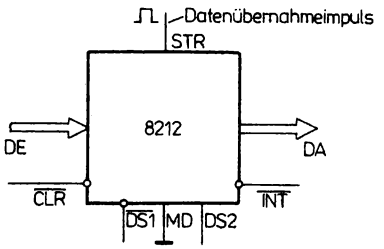


Bild 6.3
Der Baustein *8212* als Eingabetor

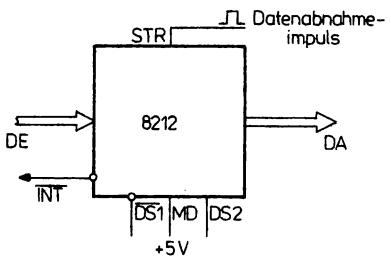
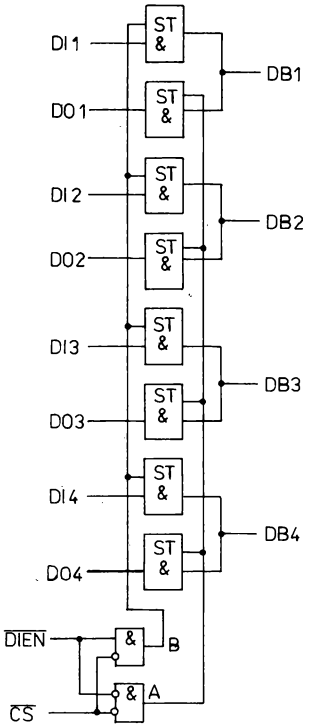


Bild 6.4
Der Baustein *8212* im Hand-shake-Prinzip (Im Bild lies Segmentausgänge a...g)

6.2.2. Bustreiberschaltkreis *8216*

Der *8216* ist ein Treiberschaltkreis für bidirektionale Bussysteme, d. h. zur Realisierung des Signalaustausches in 2 Richtungen.
Bild 6.5 zeigt den Aufbau des Bustreiberbausteins *8216*. Wenn bei A H-Potential vorliegt, so arbeitet der Baustein in Richtung DI → DB und bei H-Potential an B in Richtung DB → DO. Liegt A bzw. B auf L-Potential, so sind die Ausgänge der entsprechenden Treiber hochohmig.
Die Richtung des Informationsflusses wird durch das Signal $\overline{DIEN}$ bestimmt, während man mit $\overline{CS}$ (chip-select) den Baustein freigibt. Aus Tabelle 6.1 ist die Funktionstabelle des Bausteins *8216* zu ersehen.



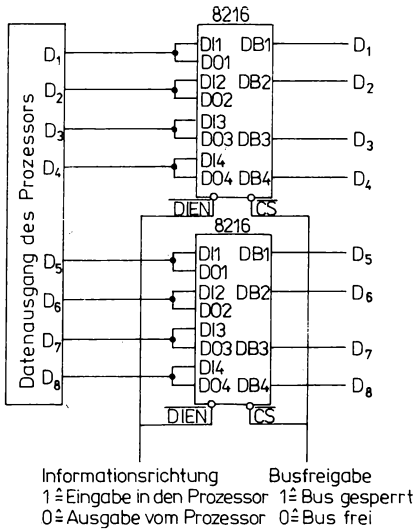


Bild 6.6
Steuerung eines 8-Bit-Datenbusses mit 2 × *8216*

Bild 6.5
Der Bustreiberbaustein *8216*

Tabelle 6.1.
Funktionstabelle des Bustreiberbausteins 8216

$\overline{CS}$	$\overline{DIEN}$	
L	L	DI → DB
L	H	DB → DO
H	L	hochohmig
H	H	hochohmig

Bild 6.6 zeigt die Verwendung von 2 Bausteinen *8216* zur Steuerung eines 8-Bit-Datenbusses. Während die Eingangsspannungen des Bausteins *8216* für H-Potential nur mindestens 2 V betragen müssen, liefert der Baustein H-Potential von mindestens 3,65 V.

6.2.3. 8-Bit-Register *DS 8282* und *DS 8283* (Bild 6.7)

Mit Hilfe des Anschlusses $\overline{OE}$ (Output Enable) läßt sich der Ausgang der Register öffnen oder sperren. Mit STB werden die Daten eingespeichert. Ist STB High, so folgen die Ausgangsdaten den Eingangsdaten. Der Schaltkreis *DS8283* unterscheidet sich vom *DS8282* nur dadurch, daß die Ausgangsdaten invertiert sind.

6.2.4. Bustreiber *DS 8286* und *DS 8287* (Bild 6.8)

DS 8286 (nichtinvertierend) und *DS 8287* (invertierend) sind bidirektionale Bustreiberschaltkreise. Mit $\overline{OE}$ (Output Enable) kann der Ausgang geöffnet oder gesperrt werden. Über T (Transmit) wird die Richtung festgelegt (High = Senden, Low = Empfangen).

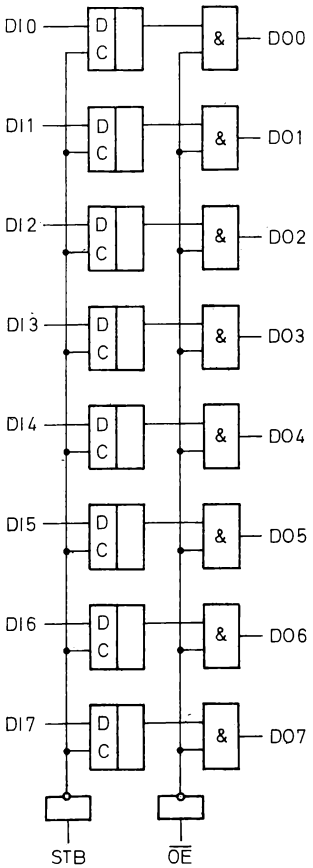


Bild 6.7
Logische Struktur des *DS 8282*

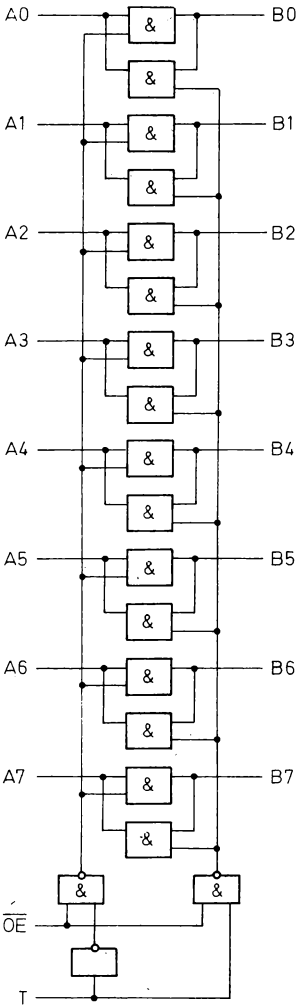


Bild 6.8
Logische Struktur des *DS 8286*

6.3. Codier- und Decodierschaltungen

Umcodierungen werden beim Aufbau von Mikrorechnern sehr häufig gebraucht. Beispiele sind die Realisierung der Anzeige durch LED-Elemente, die Zuordnung von Ziffern- und Funktionstasten zu den entsprechenden Zahlendarstellungen und die Entschlüsselung von dualen Adressen. Für viele dieser Funktionen gibt es integrierte Schaltkreise. Stehen keine speziellen Schaltkreise zur Verfügung, so lassen sich entsprechende Schaltungen auch mit logischen Grundschaltkreisen aufbauen.
Durch die Codierschaltung wird eine Darstellungsform für Zahlen oder Zeichen in eine andere Darstellungsform umgewandelt.
Bild 6.9 zeigt eine einfache Codierung Dezimalstellung → BCD-Code. Eine Dezimalziffer wird durch den ihr zugeordneten Schalter dargestellt. Beim BCD-Code bildet man duale Bit-Kombinationen mit 4 Dualstellen.
Aus Bild 6.10 ist die Schaltung des Bausteins *74147* zur Umwandlung einer Dezimaldarstellung 1 aus 10 in eine BCD-Darstellung mit 4 Bit und aus Tabelle 6.2 die Funktionstabelle des Bausteins *74147* zu ersehen.
Der Baustein *7442* (Bild 6.11) decodiert binäre Zahlen zu Dezimalzahlen.
In der Elektronik wird sehr häufig die 7-Segment-Anzeige verwendet. Aus Bild 6.12 ist die Zifferndarstellung bei der 7-Segment-Anzeige zu ersehen. Ein 7-Segment-Decoder muß aus dem BCD-Code die Signale zur Ansteuerung der 7 Segmente liefern.
Die Funktion der Ziffern 0 bis 9 findet man in Tabelle 6.3. Hinweis: Beim *40511* sind die Ausgänge H-aktiv.
Die Bausteine *7446, 7447, 7446A* und *7447A* sind integrierte 7-Segment-Decoder. Die Darstellung der Hexadezimalziffern für diese Schaltkreise enthält Bild 6.13. Bild 6.14 zeigt das Anschlußbild der Decoder.
Die Anschlüsse LT, BI/RBO und RBI erfüllen folgende Funktionen:

LT (Lamp Test) Bei Anlegen eines L-Pegels werden für die Dauer dieses Pegels alle Segmentausgänge angesteuert (Anzeige einer »8«).

RBI (Ripple Blanking Input) Über den Eingang RBI ist die Dunkeltastung der Ziffer »0« möglich. Führt RBI L-Pegel, sind die Segmentausgänge bei Erkennen einer »0« an den Dateneingängen gesperrt (Anzeige dunkel).

BI/RBO Dieser Anschluß besteht aus 2 Teilen. BI (Blanking Input) ist ein Eingang. Wird er auf L-Pegel gelegt, so werden die Segmentausgänge unabhängig vom Wert der

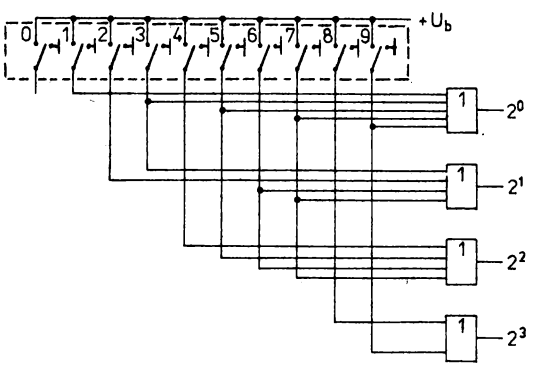


Bild 6.9
Codierung Dezimal → BCD-Code

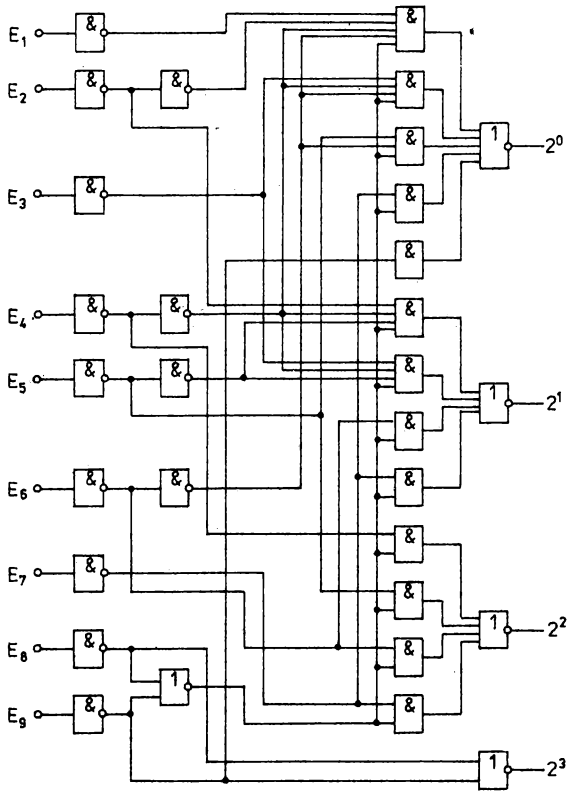


Bild 6.10
Aufbau des Bausteins *74147* zur Codierung Dezimal → BCD-Code

Tabelle 6.2.
Funktionstabelle des Bausteins 74147 (Codierer: Dezimal → BCD)

E1	E2	E3	E4	E5	E6	E7	E8	E9	A3	A2	A1	A0
H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	L	L	H	H	L
H	H	H	H	H	H	H	L	H	L	H	H	H
H	H	H	H	H	H	L	H	H	H	L	L	L
H	H	H	H	H	L	H	H	H	H	L	L	H
H	H	H	H	L	H	H	H	H	H	L	H	L
H	H	H	L	H	H	H	H	H	H	L	H	H
H	H	L	H	H	H	H	H	H	H	H	L	L
H	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

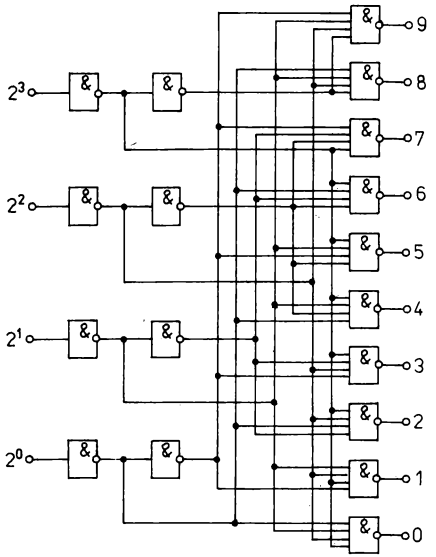


Bild 6.11
Aufbau des Bausteins *7422* zur Umwandlung BCD → Dezimal

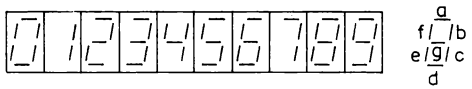


Bild 6.12
Zifferndarstellung der Ziffern 0 bis 9 einer 7-Segment-Anzeige

Tabelle 6.3.
Funktionstabelle der 7-Segment-Decoder 7446/7447

Ziffer	Eingänge				Segmente						
	D	C	B	A	a	b	c	d	e	f	g
0	L	L	L	L	H	H	H	H	H	H	L
1	L	L	L	H	L	H	H	L	L	L	L
2	L	L	H	L	H	H	L	H	H	L	H
3	L	L	H	H	H	H	H	H	L	L	H
4	L	H	L	L	L	H	H	L	L	H	H
5	L	H	L	H	H	L	H	H	L	H	H
6	L	H	H	L	H	L	H	H	H	H	H
7	L	H	H	H	H	H	H	L	L	L	L
8	H	L	L	L	H	H	H	H	H	H	H
9	H	L	L	H	H	H	H	H	L	H	H

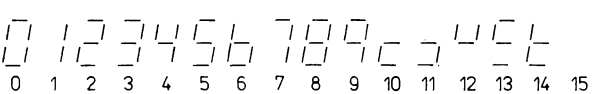


Bild 6.13
Darstellung von Hexadezimalziffern für die Bausteine *7446* und *7447*

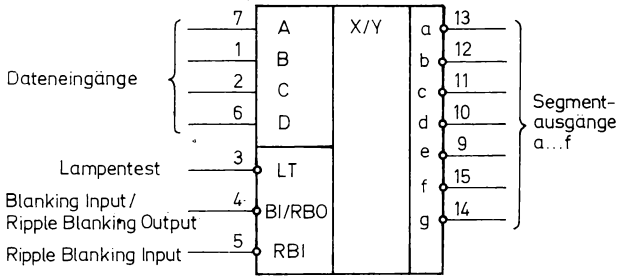


Bild 6.14
Anschlußbild von *7446* und *7447*

Eingangsdaten gesperrt (dunkelgesteuerte Anzeige). Durch einen z. B. von der Umgebungshelligkeit frequenzgesteuerten Multivibrator kann somit die scheinbare Helligkeit der Anzeige geregelt werden. RBO (Ripple Blanking Output wirkt als Ausgang, der stets dann L-Pegel führt, wenn am Eingang RBI L-Pegel liegt und die Dateneingänge L-Pegel haben (Datenwert »0«).

Mit den Anschlüssen RBI und BI/RBO läßt sich, wie Bild 6.15 zeigt, eine automatische Anzeigeunterdrückung vorlaufender Nullen bei einem mehrstelligen Display realisieren:

a – Vorlaufende Null wird angezeigt (Schalter S1 ist offen)

Durch den offenen Schalter S1 führt RBI des 1. Schaltkreises H-Pegel. Dadurch wird die an den Dateneingängen anliegende »0« an die Segmentausgänge weitergeleitet und die »0« wird angezeigt. BI/RBO des 1. Schaltkreises führt ebenfalls H-Pegel, da der RBI-Eingang auf H-Pegel liegt. Damit ist für die weiteren Schaltkreise die Linie RBI – BI/RBO auf H-Pegel geschaltet (d. h. alle Ziffern werden angezeigt.

b – Vorlaufende Null wird unterdrückt (Schalter S1 ist geschlossen)

Damit führt RBI des 1. Schaltkreises L-Pegel. Da an den Dateneingängen des 1. Schaltkreises eine »0« anliegt, werden die Segmentausgänge gesperrt (Anzeige dunkel).

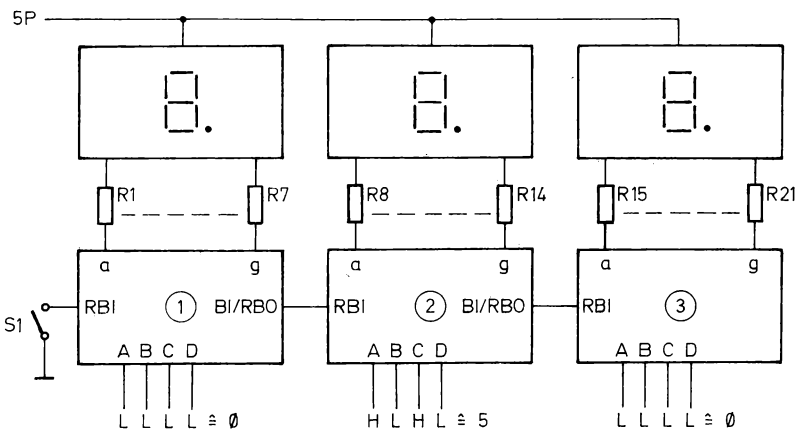


Bild 6.15
Anzeigeunterdrückung einer vorlaufenden Null

Durch die erfüllten Bedingungen (RBI = L und Daten-»0«) wird auch der Ausgang BI/RBO = L.
RBI des 2. Schaltkreises führt damit L-Pegel. Da aber an den Dateneingängen eine »5« anliegt, werden die Segmentausgänge für die Anzeige der Ziffer angesteuert. Der Ausgang BI/RBO des 2. Schaltkreises führt H-Pegel, da nun eine Bedingung für das Aktivwerden dieses Ausgangs erfüllt wurde.
RBI des 3. Schaltkreises führt H-Pegel. Aus diesem Grund wird die Daten-»0« an die Segmentausgänge weitergegeben (Anzeige »0«). Die Flußspannung U_F der Anzeigesegmente beträgt, abhängig vom verwendeten Halbleitermaterial, etwa 2,6 bis 2,8 V. Weil der zulässige Strom nicht überschritten werden darf (Zerstörung der Halbleiterschicht), sind zwischen Segmentanzeige und Decoder Widerstände zu schalten. Die Größe der Widerstände R_V kann nach der Gleichung

$$R_V = \frac{U_B - U_F}{I_S}$$

berechnet werden.
Oft liefern bestimmte Meßgeräte Ziffern in einer anderen Darstellung, als sie benötigt werden. Dann sind sogenannte »Umcodierer« notwendig. Es gibt folgende Umcodierer:
BCD-Code → Aiken-Code;
BCD-Code → 3-Exzeß-Code;
Aiken-Code → BCD-Code;
3-Exzeß-Code → BCD-Code.
Die aus der Sicht der Mikrorechentechnik wichtigsten Nachteile der beschriebenen TTL-Decoder sind:
- hoher Eigenstromverbrauch (64 mA),
- Stromabgabe der Dateneingänge bei L-Pegel: 1,6 mA,
- keine Anzeige von Hexadezimalziffern, und man benötigt
- 7 externe Widerstände zur Segmentstrombegrenzung.

Durch neue Technologien konnten diese Mängel beseitigt werden.

Decodertypenreihe *D/E345* bis *348*

Diese Typenreihe eignet sich zum Ansteuern von 7-Segment-Anzeigen mit gemeinsamer Anode (z. B. *VQB 18, VQB 28, VQE 12, VQE 14, VQE 22, VQE 24*). Durch die Anwendung der I^2L-Technologie beträgt der Eigenstrombedarf nur etwa 10 mA, die Stromabgabe der Dateneingänge bei L-Pegel beträgt 0,4 mA, die Einstellung des Segmentstroms erfolgt mit einem Programmierwiderstand bzw. wurde im Schaltkreis als Konstantstromsenke ausgeführt. Tabelle 6.4. enthält das angebotene Typenspektrum.
Bei den Schaltkreisen mit fest eingestellter Konstantstromsenke gibt es 2 Ausmeßtypen, die mit dem Suffix m = 8 bis 12 mA bzw. p = 10 bis 14 mA versehen sind.
Bei den Schaltkreisen mit programmierbarem Segmentstrom ist die Stromsenke von 0 bis 40 mA linear einstellbar. Die Programmierung der Segmentströme ermöglicht der Widerstand R_V (Bild 6.16).
Die Schaltkreise befinden sich in einem 16poligen Plastgehäuse und sind (bis auf Pin 3) mit den Schaltkreisen *7446* und *7447* pinkompatibel. Mit den Anschlüssen BI/RBO und RBI lassen sich vorlaufende Nullen unterdrücken (siehe *7446* und *7447*).

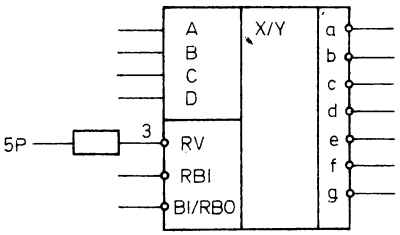


Bild 6.16
Programmierung der Segmentströme

Tabelle 6.4.
I²L-Decodervarianten

Typ	Anzeige	Ausgangsstrom	Temperaturbereich
D 345	H	fest	N
E 345	H	fest	E
D 346	H	programmierbar	N
E 346	H	programmierbar	E
D 347	M	fest	N
E 347	M	fest	E
D 348	M	programmierbar	N
E 348	M	programmierbar	E

H ≙ Hexadezimalziffernanzeige; M ≙ meßtechnisch orientierte Anzeige; N ≙ 0 bis 70 °C; E ≙ −25 bis 85 °C

Decoder *U/V 40511*

Für die Ansteuerung von 7-Segment-Anzeigen mit gemeinsamer Katode wurde der Schaltkreis *U/V 40511* entwickelt, dessen stromsparender Logikteil in CMOS-Technik ausgeführt wurde, während die bipolaren Ausgangsstufen bis zu 25 mA je Segment abgeben können. Die Segmentausgänge sind also im Gegensatz zum *7446/7447* H-aktiv. Der Schaltkreis enthält außer dem eigentlichen Decoder, der auch Hexadezimalziffern anzeigt, noch einen Zwischenspeicher (Latch). Das bei LE (Latch Enable) = L eingelesene Datenmuster wird auch dann weiter angezeigt, wenn sich bei LE = H das Datenmuster am Eingang ändert. Dadurch wird der Multiplexbetrieb mehrstelliger Anzeigen wesentlich erleichtert. Der Schaltkreis befindet sich in einem 16poligen Plastgehäuse und ist (bis auf den Pin 5 = LE) mit den Typen *7446/7447* pinkompatibel.

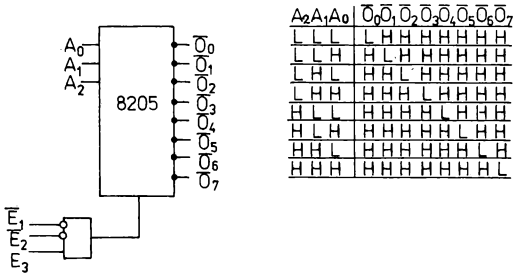
Die elektrischen Werte (Betriebsspannung, Eingangspegel) entsprechen denen anderer CMOS-Schaltkreise. Der Präfix U kennzeichnet den Temperaturbereich (0 bis 70 °C), seit 1984 wird der Schaltkreis mit dem Präfix V (−25 bis 70 °C) gefertigt.

Wichtige Funktionen bei der Adreßentschlüsselung sind die Umcodierungen

Binär → Oktal;

Binär → Hexadezimal.

Im Fall »Binär → Oktal« werden aus einer Oktalziffer 8 Einzelsignale (3 zu 8), im Fall »Binär → Hexadezimal« 16 Einzelsignale (4 zu 16) gebildet. Der Baustein *8205* (Bild 6.17) realisiert die Funktion 3 zu 8.



$A_2A_1A_0$	$\overline{O}_0\overline{O}_1\overline{O}_2\overline{O}_3\overline{O}_4\overline{O}_5\overline{O}_6\overline{O}_7$
L L L	L H H H H H H H
L L H	H L H H H H H H
L H L	H H L H H H H H
L H H	H H H L H H H H
H L L	H H H H L H H H
H L H	H H H H H L H H
H H L	H H H H H H L H
H H H	H H H H H H H L

Bild 6.17
3-zu-8-Decoder *8205* mit Funktionstabelle

6.4. Prüfbiterzeugung – Paritätsprüfer

Die häufigste Methode zur Feststellung von Fehlern bei einer Datenübertragung ist die Prüfbitkontrolle. Dabei wird die Anzahl der zur Übertragung verwendeten Bit durch ein Zusatzbit, das sogenannte *Prüfbit*, ergänzt. Dieses Prüfbit wird nun so gesetzt, daß die Anzahl der Einsen im Gesamtwort (Information + Prüfbit) gerade oder ungerade ist. Im Sender setzt man das Prüfbit dazu, im Empfänger läßt sich dann die gerade oder ungerade Parität überprüfen. Bild 6.18 zeigt den Aufbau des Bausteins *74180*, der zur Überprüfung und Erzeugung der Prüfbit eines 8-Bit-Wortes (7 Bit + Prüfbit) verwendet werden kann. Aus Tabelle 6.5 läßt sich die zugehörige Funktionstabelle ersehen.

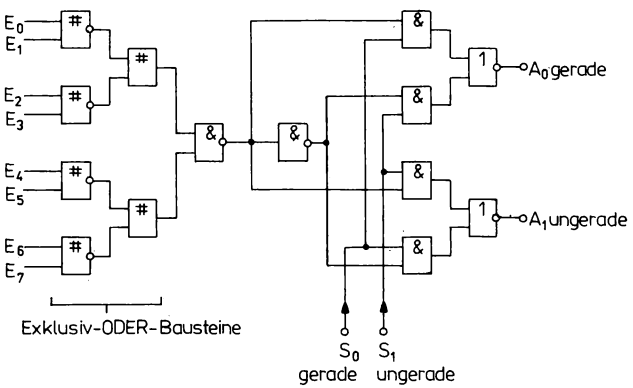


Bild 6.18
Aufbau des Bausteins *74180* (Paritätsbildung)

Tabelle 6.5.
Funktionstabelle des Paritätsbausteins 74180

E_0 bis E_7	S_0	S_1	A_0	A_1
gerade	H	L	H	L
ungerade	H	L	L	H
gerade	L	H	L	H
ungerade	L	H	H	L
beliebig	H	H	L	L
beliebig	L	L	H	H

Soll z. B. zu einem 7-Bit-Wort ein 8. Bit so gesetzt werden, daß geradzahlige Parität entsteht, so verbindet man die 7 Bit mit E_0 bis E_6, setzt $E_7 = 0$, $S_0 = 1$ und $S_1 = 0$. Damit wird $A_1 = 1$, wenn E_0 bis E_6 ungerade sind, und $A_1 = 0$, wenn E_0 bis E_6 gerade sind. Es gilt $A_0 = A_1$; A_1 ist unmittelbar das gesuchte Prüfbit. Zur Prüfbitkontrolle verbindet man die 8 Datenleitungen mit E_0 bis E_7, setzt $S_0 = 1$ und $S_1 = 0$. Damit ist bei geradzahliger Anzahl von Einsen $A_0 = 1$, $A_1 = 0$ und bei ungeradzahliger Anzahl $A_0 = 0$, $A_1 = 1$.
Mit 2 Bausteinen *74180* läßt sich auch eine 16-Bit-Prüfbitlogik aufbauen.

6.5. Taktgeneratoren

Die meisten Rechnerschaltkreise arbeiten taktgesteuert. Die Erzeugung des Grundtakts übernimmt ein externer Taktgenerator. Bild 6.19 zeigt eine einfache Variante zur Erzeugung einer unstabilisierten Taktserie. Liegt am Punkt A H-Pegel, so hat Punkt B L-Pegel. Der Kondensator C entlädt sich über R, bis Punkt A L-Pegel erhält. Jetzt hat B H-Pegel, und der Kondensator C lädt sich über R und den Negator 1 wieder auf. Die Umladung von C wird durch Negator 2 geringfügig unterstützt.
Bild 6.20 zeigt eine Schaltung, in der die Auf- und Entladung von C durch R_1 und R_2 geschieht. Außerdem wird durch Hinzuschalten eines Quarzes die erzeugte Taktserie frequenzstabilisiert. Liegt die Quarzfrequenz zu hoch, so kann man die Taktfrequenz mit D-Flip-Flop untersetzen. Bild 6.21 zeigt eine solche Untersetzung mit 2 D-Flip-Flop.
Sollen aus dem Urtakt 2 Taktserien hergestellt werden, so läßt sich das mit einem D-Flip-Flop nach Bild 6.22 erreichen.
Der Baustein *8224* ist ein integrierter Taktgenerator für den Mikroprozessor *8080*. Er erzeugt die notwendigen Taktimpulse Φ_1 und Φ_2 und dient gleichzeitig zur Verarbeitung des Zeitsignals SYNC und zur Erzeugung der Signale RESET, READY und $\overline{\text{STSTB}}$ (Bild 6.23).

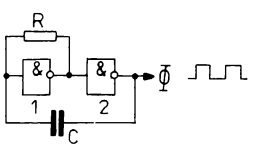


Bild 6.19
Erzeugung einer unstabilisierten Taktserie

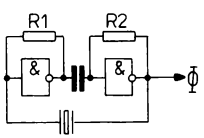


Bild 6.20
Einfacher Taktgenerator

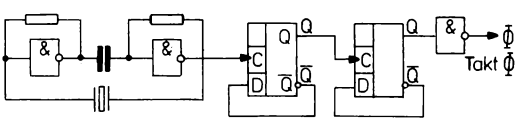


Bild 6.21
Taktgenerator mit 2facher Untersetzung

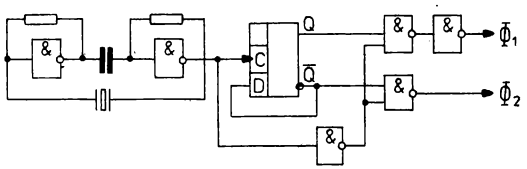


Bild 6.22
Taktgenerator zur Erzeugung von 2 Taktserien Φ_1 und Φ_2

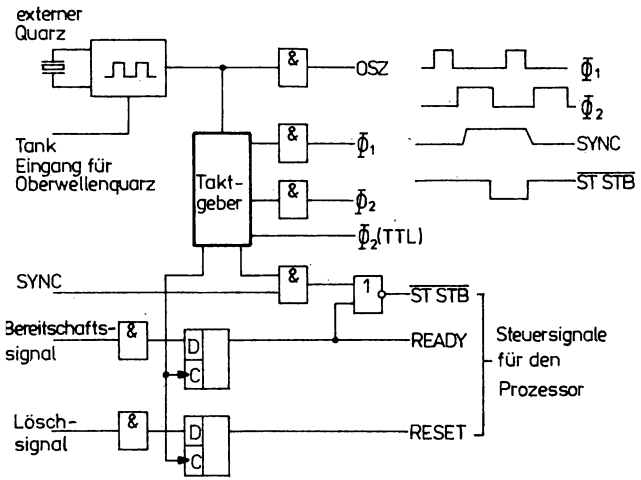


Bild 6.23
Integrierter Taktgenerator *8224* für den Prozessor *8080*

7. Assemblersprache

7.1. Überblick

In der Assemblersprache werden die einzelnen Maschinenbefehle durch Textzeilen formuliert. Jeder Befehl entspricht einer Zeile, in der der Operationscode und die Operanden durch Abkürzungen geschrieben werden. Statt der Speicheradresse, in die der Maschinenbefehl gespeichert wird, bekommt die Assemblerzeile einen Namen (Marke). Da der Rechner jedoch nur Maschinenbefehle versteht, muß ein Assemblerprogramm in Maschinenbefehle umgewandelt werden. Dies realisiert der Assembler.

7.2. Aufbau einer Assemblerzeile (Bild 7.1)

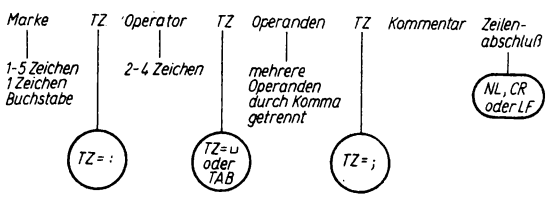


Bild 7.1
Aufbau einer Assemblerzeile [TZ ≙ Trennzeichen, TAB ≙ Tabulatorzeichen, NL ≙ New Line (Zeichen für neue Zeile), CR ≙ Carriage Return (Zeichen für Wagenrücklauf), LF ≙ Line Feed (Zeichen für Zeichenwechsel)]

- Marke: frei wählbarer Name/1–5 Zeichen/1. Zeichen = Buchstabe
- Trennzeichen (TZ) zwischen Marke und Operator: (Doppelpunkt)
- Operator: Festgelegte Bezeichnung für einen Befehl
- Trennzeichen zwischen Operator und Operanden: Leerzeichen oder Tabulator
- Operanden:

	Darstellung	Erläuterung (Beispiele)
1. Zahlen:	Dezimal	14
	Oktal	14O oder 14Q
	Hexadezimal	(die erste Ziffer darf kein Buchstabe A–F sein) 25H, 0A3H
	Binär	1011B
2. Symbolische Größen	Entspricht Marke oder definiertem Symbol	
3. aktueller Stand des Speicherplatzzählers:	#	Das Zeichen # entspricht der Adresse, in der das 1. Byte des Operationscodes steht

4. Zeichen im SIF-1000-Code:	'A' oder 'AX'	statt A bzw. X wird der entsprechende SIF-1000-Code abgespeichert
5. Registernamen:	einfache Register	A, B, C, usw.
	Doppelregister	DE, HL, IX, usw.
6. Ausdrücke:	Zulässige Operationen + −	

Satz: Die Anzahl der positiven minus der Anzahl der negativen verschiebbaren Symbole (Namen und #) darf nur 0 oder 1 sein.

Beispiele:			
	A + 5	erlaubt	(Anzahl = 1)
	A + # − C	erlaubt	(Anzahl = 1)
	A + #	nicht erlaubt	(Anzahl = 2)

7. externe Adressen: Beispiel P1. MA

Programmname Zieladresse im Programm P1

- Trennzeichen zwischen mehreren Operanden , (Komma)
- Trennzeichen zwischen Operanden und Kommentar ; (Semikolon)
- Kommentar: beliebiger Text, der nicht mit verarbeitet wird
- Zeilenabschluß: Neue Zeile (NL)
  Wagenrücklauf (CR)
  Zeilenvorschub (LF)

7.3. Pseudoanweisungen

Pseudoanweisungen sind Anweisungen der Assemblersprache, die sofort bei der Übersetzung ausgeführt werden. Das Ergebnis wird in das Maschinencodeprogramm eingefügt. Die Pseudoanweisungen sind nicht in der Maschinensprache enthalten.
Man benutzt sie zum Einfügen von Zahlen in das Programm sowie zur Steuerung des Assemblerablaufs.

Zahlen- und Textdefinitionen

Bytedefinition:

DB	n	Die Zahl n (8 Bit) wird ins Programm eingefügt.

Adressendefinition:

DA	nn	Die Zahl nn (16 Bit) wird ins Programm eingefügt, niederwertiger Teil zuerst.

Textdefinition:

DB	'TEXT'	Die Zeichenkette TEXT wird in der Reihenfolge von links nach rechts ins Programm eingefügt.

Bereitstellung von RAM-Speicher im Programm:

MA: BER	nn	Im Programm werden nn Zellen unter der Adresse MA bereitgestellt.

Programmorganisation

PN	name	Anfang eines Programms Vom Programmnamen sind nur 2 Zeichen signifikant.
END		Ende des Programms

ORG	nn	Der Speicherplatzzähler wird auf die Adresse nn gestellt. Die nächste Anweisung kommt in die Zelle nn.
SYMB: EQU	nn	Dem Namen SYMB wird die Zahl nn zugeordnet. (Symboldefinition) Diese Zuordnung gilt für das gesamte Programm.
SYMB: DEF	nn	Wie EQU wird dem SYMB die Zahl nn zugeordnet. Die Zuordnung gilt aber nur bis zur nächsten DEF-Anweisung

Übersetzungssteuerung

IF A	Die Anweisung 1 bis n wird übersetzt, wenn A ≠ 0.
Anweisung 1	Wenn A = 0, werden die Anweisungen beim Übersetzen weggelassen
Anweisung n	
ENIF	
TITL 'TEXTKETTE'	Die Textkette erscheint als Überschrift in der Programmliste.
EJEC	Der Drucker für die Programmliste macht einen Formularvorschub.

7.4. Makroorganisation

Ein MAKRO ist eine Befehlsfolge, die der Assembler bei der Übersetzung in das Programm einsetzt.

Makrodefinition

Ein MAKRO wird definiert durch:

```
NAME: MACR    Liste der formalen Parameter
      Anweisung 1

      Anweisung n
      ENDM
```

Die formalen Parameter können in den Anweisungen 1 bis n beliebig auftreten. In der Spalte »Marke« steht der Name des MAKRO. Er ist unter gleichen Bedingungen wie eine Marke frei wählbar. In der Spalte »Operator« steht der Pseudobefehl MACR. Er sagt aus, daß die folgenden Anweisungen 1 bis n zu dem MAKRO mit diesem definierten Namen gehören. Die formalen Parameter sind Zeichenkettenparameter, die innerhalb des Makros verwendet werden können.

Beispiel:

Dezimale Addition der Zelleninhalte A1 und A2. Es soll durch das MAKRO mit Namen »DEZA« der Inhalt der Zellen A1 und A2 addiert und in Zelle A1 gespeichert werden.

(A1) + (A2) → (A1)

Das MAKRO hat folgendes Aussehen:

```
DEZA:  MACR    Z1, Z2;     Definitionszeile, Z1 und Z2 sind formale Parameter
       LD      A, (Z1)   ┐
       LD      HL, Z2    │
       ADC     (HL)      ├ Makrokörper
       DAA               │
       LD      (Z1), A   ┘
       ENDM;              Ende des MAKRO
```

Makroaufruf:

Der Aufruf erfolgt durch den Namen des MAKRO als Pseudobefehl und der folgenden Liste der aktuellen Parameter. Soll zum Beispiel der Inhalt der Zellen 30H und 31H addiert werden, so geschieht dies durch den Aufruf DEZA 30H, 31H.

Durch den Aufruf wird an die Stelle im Programm, an der der Aufruf steht, ein Programmstück (Makroerweiterung genannt) eingesetzt, in dem die formalen Parameter Z1 und Z2 durch die aktuellen Zeichenketten 30H und 31H ersetzt werden. Das Programmstück hat daher dieses Aussehen:

```
LD    A, (30H)
LD    HL, 31H
ADC   (HL)
DAA
LD    (30H), A
```

Sätze:

- Bei der Übersetzung wird statt des Makroaufrufes ein Programmteil eingesetzt, in dem die formellen Parameter der Makrodefinition durch die aktuellen Parameter in der gleichen Reihenfolge ersetzt werden. Den Programmteil nennt man Makroerweiterung. Sind beim Aufruf mehr aktuelle Parameter als formelle vorhanden, werden die überflüssigen weggelassen. Sind beim Aufruf weniger aktuelle Parameter als formelle vorhanden, werden die fehlenden mit »0« belegt.
- Makrobefehle müssen vor ihrem Aufruf definiert sein. Es ist möglich, eine Makrobibliothek anzulegen. Dann genügt es, vor dem Aufruf das Makro aus der Bibliothek ins Programm zu holen.

7.5. Assembler

Die Abkürzungen für die Operationscode und Operanden werden durch den Assembler festgelegt. So können sie sich deshalb vom Assembler zu Assembler geringfügig unterscheiden.

Die folgende Übersicht zeigt die wesentlichen Unterschiede zwischen den Schreibweisen von Operationen und Operanden der Assembler *M 80* im Betriebssystem CP/M und dem Assembler *MAPS K 1520.*

```
M 80                          MAPS K 1520
-------------------------------------------------------------------------

Transportoperationen
LD r,(HL)                     LD r,M
LD (HL),r                     LD M,r (statt (HL);M)
LD (HL),n                     LD M,n
EX AF,AF'                     EXAF

Rechenoperationen
OP A,r                        OP r
OP A,n                        OP n
CP s                          CMP s

Sprünge
JP ADR                        JMP ADR
JP NZ,ADR                     JPNZ ADR
JP Z,ADR                      JPZ ADR
usw.                          usw.
CALL NZ,ADR                   CANZ ADR
CALL Z,ADR                    CAZ ADR
usw.                          usw.
RET NZ                        RNZ
RET Z                         RZ
usw.                          usw.

Relative Sprünge
JR C,MARKE                    JRC MARKE-#
JR C,$+5                      JRC #+5

E/A-Befehle
IN A,(n)                      IN n
IN r,(C)                      IN r
OUT (n),A                     OUT n
OUT (C),r                     OUT r

Pseudooperationen
ORG nn                        ORG nn
EQU nn                        EQU nn
DEFL nn                       DEF nn
DEFB n                        DB n
DEFW nn                       DA nn
DEFM "TEXT"                   TITL "TEXT"
DEFS nn                       BER nn
IF nn                         IF nn
ENDIF                         ENIF
MACRO P0,P1                   MACR P0,P1
ENDM                          ENDM
END                           END
```

7.6. Programmaufbereitung

Hat man ein Programm geschrieben, wird es vor der Verarbeitung durch einen Rechner zunächst auf einen Datenträger (Lochband, Lochkarte, Magnetband, Diskette usw.) gebracht. Das so erfaßte Programm ist als Text auf dem Datenträger und heißt Quellcode (QC). Wenn das Programm arbeiten soll, muß es in einer arbeitsfähigen Form im Arbeitsspeicher stehen. Diese Form heißt Maschinencode (MC). Neben Quellcode und Maschinencode gibt es noch verschiedene Zwischencode (OC = Objektcode, Bibliothekscode), die in den einzelnen Schritten bei der Umwandlung von Quell- in Maschinencode gebraucht werden.
Der Objektcode eines Programms kann an beliebiger Stelle des Speichers geladen werden. Die Adressen ändern sich entsprechend der Anfangsladeadresse (Leitadresse des Programms).
Der Bibliothekscode ist durch Merkmale versehen, die zur Einordnung in eine Bibliothek benötigt werden.

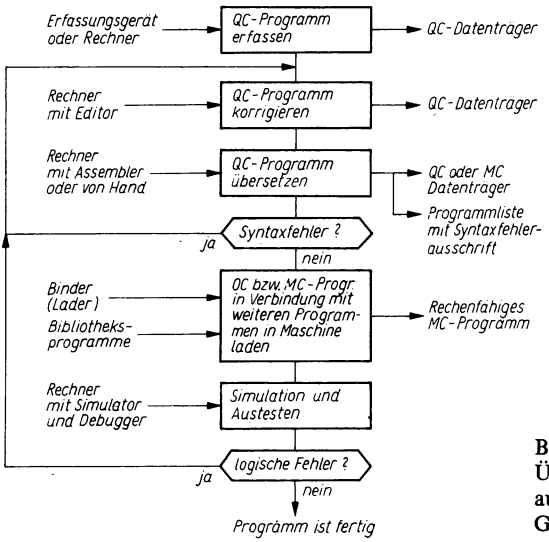


Bild 7.2
Übersicht der Schritte für die Programmaufbereitung und der dazu notwendigen Geräte und Systemprogramme

Für die einzelnen Bearbeitungsschritte gibt es auf sogenannten Entwicklungssystemen (das sind Rechner zur Programmentwicklung) und Personalcomputern Programme, die die Arbeit in den einzelnen Bearbeitungsschritten unterstützen (Bild 7.2).

Systemprogramme zur Programmaufbereitung

Editor
Der Editor ist ein Programm, mit dem man einen Quellcode, d. h. einen Text, korrigieren kann. Das Programm enthält Kommandos zum Streichen, Auswechseln und Hinzufügen von Zeichen, Worten und Zeilen. Als Ergebnis entsteht ein neuer Quellcode.

Assembler
Der Assembler ist ein Programm, mit dem der Quellcode in den Objektcode oder Maschinencode umgewandelt wird. Zusätzlich entsteht u. a. eine Programmliste.

Binder (Linker, Taskbuildner)
Der Binder ist ein Programm, mit dem man mehrere Programme (Hauptprogramm, Unterprogramme und Programme aus der Bibliothek) zu einem rechenfähigen Programm zusammenstellen kann.

Debugger
Der Debugger ist ein Programm zum Korrigieren des Maschinencodes. Er enthält Kommandos für das Lesen und Beschreiben von Speicherzellen zur Überprüfung von Programmteilen.

Simulator
Der Simulator ist ein Programm, mit dem man ein Maschinenprogramm rechnen lassen kann. Während der Rechnung kann man den Verlauf im Rechner protokollieren.

Bibliothekar
Der Bibliothekar ist ein Programm zur Verwaltung einer Programmbibliothek.

Fileprozessor
Der Fileprozessor ist ein Programm zur Verwaltung und zum Transfer von Filestrukturen. Mit ihm lassen sich Files löschen, kopieren, verändern und hinzufügen.

Dienstprogramme
Dienstprogramme sind kleine Systemprogramme, mit deren Hilfe man spezielle Funktionen ausführen kann;
z. B.
- Doppeln von Lochbändern, Lochkarten, Magnetbändern, Magnetplatten.
- Initialisieren von Magnetplatten.
- Kopieren von Band nach Platte, Speicher nach Platte usw.
- Drucken von Verzeichnissen.

Monitor
Der Monitor ist der Steuerteil des Betriebssystems. Er verwaltet sowohl die einzelnen Systemprogramme und Nutzerprogramme als auch die Ressourcen der Maschine wie Speicher, Ein-/Ausgabegeräte und externe Speicher.

8. Software

Zur Arbeit mit einem Computer sind 2 Komponenten erforderlich – die Hardware und die Software (Bild 8.1).
Die Hardware (das Gerät) ist so aufgebaut, daß es die Informationen und den Weg zur Lösung eines Problems speichern kann. Der Lösungsweg (Programm), gespeichert in Form von Maschinenbefehlen, wird durch die Hardware entschlüsselt und abgearbeitet. Die Gerätetechnik ist programmierbar. Damit bildet die Hardware die gerätetechnische Basis zur Lösung der unterschiedlichsten Probleme. Die Lösung eines Problems wird durch ein Programm realisiert.
Um den Computer schnell einsetzbar zu machen, werden viele allgemeine Probleme bereits durch fertige Programme zugänglich gemacht. In ein Programm gibt der Nutzer nur spezielle Daten bzw. Anweisungen für den speziellen Anwendungsfall ein. Der Nutzer hat damit mit der Maschinensprache des Computers gar nichts mehr zu tun. Er bedient lediglich die für die allgemeinen Probleme zuständigen Programme. Die Programme bilden die Software. Durch die Software kann man den sehr allgemein einsetzbaren Computer für einen bestimmten Einsatzfall präparieren.
Die wichtigsten unterschiedlichen Einsatzgebiete der Mikrorechentechnik sind
a) Einbausysteme zur Gerätesteuerung,
b) OEM-Systeme zur Meßwerterfassung und Prozeßsteuerung,
c) Büro-, Personalcomputer oder Arbeitsplatzcomputer.
Die Software für Fall a) besteht aus speziellen Steuerroutinen sowie einfachen Programmbausteinen.
Die Software für Fall b) sind vor allem Echtzeitsysteme, Programmpakete für Arithmetik und Graphik und Steuerprogramme.
Die Software für Fall c) besteht aus allgemeiner Nutzersoftware zur Handhabung von Programmsprachen und Anwenderkomponenten zur Textverarbeitung, Datenbautechnik, Numerik, Kalkulaturtechnik, Graphik usw.
In der folgenden Übersicht sind die wichtigsten Komplexe der Software zusammengestellt:

- selbständige Arbeitsroutinen (PROM-fähig)
  (Unterprogramme, Programmbausteine)
  Arithmetik
  Numerik

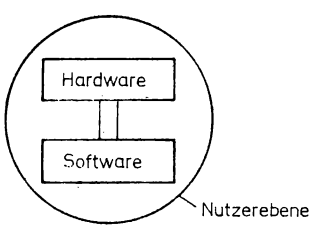


Bild 8.1
Komponenten eines Computersystems

Logische Funktion
Textgenerierung
Graphische Funktion
Ein-/Ausgaberoutinen
Rechenkopplung
Prozeßbedienung

- Selbständige Programmpakete (PROM-fähig)
  Arithmetik-Festkommapakete
  Arithmetik-Gleitkommapakete
  Numerikpakete
  Textverarbeitung
  Statistik
  Graphik
  Menütechnik
- Monitorprogramme
- selbständige Programme zur speziellen Rechnernutzung
  Assembler
  Reassembler
  Editoren
  Kopier- und Konvertierungsprogramme
  Basicinterpreter
- Betriebssysteme
  Systeme für Personal- und Arbeitsplatzcomputer
  Echtzeitsysteme
  Systeme zur Rechnerkopplung
  Mehrprozessorsysteme
  Spezialsysteme (Feldrechner, Losesysteme für Differentialgleichungen)
- Dienstprogramme
  Filebehandlung
  Gerätebehandlung

- Programmentwicklungssoftware
  Interpreter
  Compiler
  Linker
  Locater
  Bibliotheksprogramme
  Debugger
  Textsysteme

- Nutzerkomponenten und Pakete
  Textverarbeitung
  Datenbanktechnik
  Kalkulatortechnik
  Graphik

Numerik
Statistik

Die Routinen (meist als einzelne Routinen einer Programmbibliothek) bilden die Basis für die Anwendung als Gerätesteuerrechner. Hierbei kommt es darauf an, die Software für die Signalverarbeitung zu entwickeln und auf PROM bereitzustellen. Für die einzelnen Funktionen werden dabei Routinen eingesetzt, die unter einem kleinen Steuersystem arbeiten. Wortlänge und Zahlendarstellung richten sich nach den Gerätedaten. Um die unterschiedlichen Zahlendarstellungen einander anzupassen, werden Konvertierungsroutinen benötigt. Arithmetische Funktionen kommen in dem Umfang vor, wie sie für die Steuerung des Gerätes benötigt werden; häufig schnelle Routinen mit kurzen Wortlängen, z. B. Logarithmusfunktion 12 oder 16 Bit für die Umsetzung von linearen in logarithmische Skalen, trigonometrische und dazugehörige Umkehrfunktionen bei der Umwandlung von kartesischen in Polarkoordinaten.
Für die Realisierung gibt es verschiedene Verfahren. Dabei sind solche Verfahren besonders geeignet, die sich unmittelbar auf die Verarbeitung von Bit-Mustern stützen.
Die folgende Zusammenstellung zeigt die gebräuchlichsten Verfahren zur Lösung von arithmetischen Problemen auf Mikrorechnern. Grundrechenarten: (Addition, Subtraktion, Multiplikation, Division)
– Lösung durch Verschiebe-, Addier- und Subtrahieroperationen von Bit-Mustern
Zahlenkonvertierung
– Lösung durch einfache Multiplikation und Division, die auf Addition, Subtraktion und Verschiebung zurückgeführt werden.
Standardfunktion: (Logarithmus, Exponentialfunktion, trigonometrische und Umkehrfunktion, hyperbolische Funktion)
Lösungsverfahren
– Approximation durch Polynome (Taylor, Tschebyscheff)
– Approximation durch Kettenbrüche
– Cordicalgorithmen
Cordicalgorithmen eignen sich besonders für Festkommaoperationen, da sie unmittelbar mit Bit-Mustern arbeiten.
Programmbausteine lassen sich je nach Anwendungsgebiet in Stufen unterteilen.
Unterteilung Programmbausteine für numerische Probleme:
1. Ebene: Basisprogramme
– Grundfunktion zur Zahlenverarbeitung
● elementare Multiplikation und Division
● Grundoperationen für N-Byte (Verschiebung, Addition, Subtraktion, Normalisierung)
2. Ebene: Arithmetik
– Programmbausteine für Fest- und Gleitkommazahlen im Dual- und BCD-Format
● Grundrechenarten
● Zahlenumwandlung
● Standardfunktionen
3. Ebene:
– allgemeine numerische Lösungsverfahren
Unterteilung Programmbausteine für Peripheriesteuerung:
1. Ebene: Steuerroutinen

2. Ebene: Ein-/Ausgaberoutinen
3. Ebene: Komplexe Ein-/Ausgabefunktionen

Programmpakete fassen eine Gruppe von Operationen eines Gebietes zusammen. Sie eignen sich bei Einsätzen von Mikrorechnern für Meßwerterfassung, Prozeßsteuerung und Spezialrechnern, wie Bürocomputer, Personalcomputer, Textverarbeitungssystemen usw. Am häufigsten werden Programmpakete für arithmetische Funktionen benötigt. Über sie läuft die Auswertung von Meßdaten sowie die Zusammenstellung von Ergebnissen.
Innerhalb eines Arithmetikpaketes ist die Zahlendarstellung fest. Zahlen, die nicht in der betreffenden Darstellung vorliegen, müssen in die Darstellung des Pakets umgeformt werden. Arithmetikpakete realisieren i. a. folgende Funktionen für eine feste Zahlendarstellung:

Konvertierung von Zahlen	Grundrechenarten
– BCD – Dual	– Addition
– Dual – BCD	– Subtraktion
– Text (ASCII) – DUAL	– Multiplikation
– DUAL – TEXT (ASCII)	– Division
– Festkomma – Gleitkomma	
– Gleitkomma – Festkomma	

Standardfunktionen	
– Wurzel	– Hyperbolischer Sinus (sinh)
– Sinus (sin)	– Hyperbolischer Cosinus (cosh)
– Cosinus (cos)	– Hyperbolischer Tangens (tanh)
– Tangens (tan)	– natürlicher Logarithmus
– Arcus-Sinus (arcsin)	– Exponentialfunktion
– Arcus-Tangens (arctan)	– Potenzfunktion
– area sinh	– ganzer Teil einer Zahl
– area cosh	
– area tanh	

Dienstprogramme zur Programmaufbereitung werden in Mikrorechnerentwicklungssystemen eingesetzt. Mikrorechnerentwicklungssysteme sind speziell für die universelle Unterstützung der Programmentwicklung vorgesehen.
Bild 8.2 zeigt die Struktur eines solchen Systems. Entwicklungssysteme sind für den universellen Dialogbetrieb ausgelegt. Sie besitzen die in Bild 8.2 angeführten Funktionsteile. Der Emulator-Adapter ermöglicht das Anschließen des zu entwickelnden Gerätes an das Entwicklungssystem. Damit können Programme in das Anwendersystem übertragen und im Anwendersystem unter Echtzeitbedingungen ablaufen. Der Ablauf kann durch die Echtzeitüberwachung kontrolliert und protokolliert werden. Der Dialog zur Programmentwicklung erfolgt im Wechselspiel zwischen Bediener und Rechner. Für diese Dialogarbeit gibt es verschiedene Ausbaustufen:

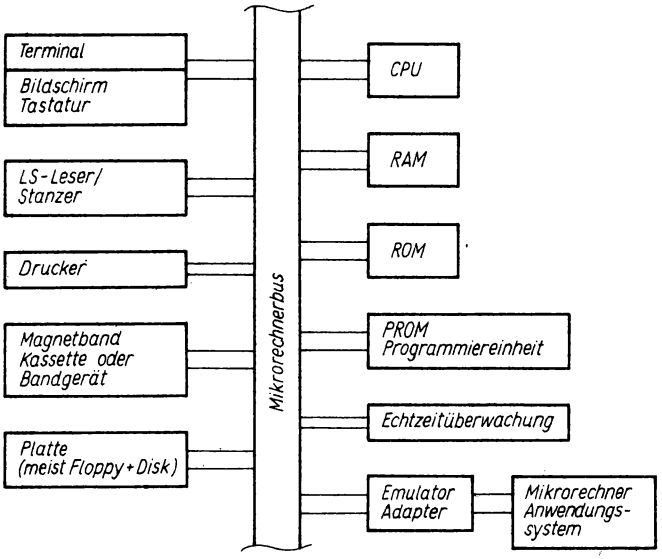


Bild 8.2
Struktur eines Mikrorechnerentwicklungssystems

Kommandosteuerung
Auf dem Bildschirm erscheint ein spezielles Kennzeichen (z. B. Punkt (.) oder Doppelkreuz (#)). Der Bediener gibt nun über die Tastatur ein Kommando.
Das Kommando besteht aus einem Operationsteil (meist abgekürzt), aus Parametern (durch Komma oder Lehrzeichen getrennt) und aus dem Abschlußzeichen (meist »Return«). Nach dem Geben des Abschlußzeichens führt das System das Kommando aus.
Frage-Antwortsystem
Auf dem Bildschirm erscheint eine Frage, die durch Eingabe von Parametern beantwortet werden muß. Nach Eingabe der Parameter wird die entsprechende Funktion ausgelöst.

Menütechnik
Auf dem Bildschirm erscheinen eine Frage und, z. B. in Klammern gesetzt, die möglichen Antworten. Der Bediener muß jetzt nur eine der möglichen Antworten eintippen, um die entsprechende Funktion auszulösen. Meistens ist die Antwort einfach Ja (Y = Yes) oder Nein (N = No).
Die Software zu einem Entwicklungssystem besteht im wesentlichen aus einem leistungsfähigen Betriebssystem und Dienstprogrammen wie

- Transferprogramme
- Assembler
- Reassembler
- Interpreter
- Compiler
- Simulatoren
- Emulatoren
- Echtzeitanalyseprogramme
- Programme zur Fileorganisation
- Initialisierungsprogramme
- Copier- und Konvertierungsprogramme
- Bibliotheksorganisation
- Binder oder Taskbuildner
- Debugger (Fehlersuchprogramme)
- PROM-Programmierroutinen

(Dienstprogramme zur Systemüberprüfung)

- Testprogramme
- Fehlersuchroutinen

(Programme zur Datenaufbereitung)

- Textbehandlungsprogramme
- Datenbanksysteme
- Sortierprogramme

9. Programmbeispiele

9.1. Programme zur Zahlenumwandlung

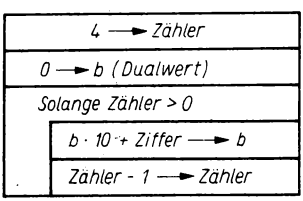
Die Darstellung der Zahlen im Mikrorechner wird durch den Anwendungsfall bestimmt. Sie wird durch das betreffende Programm definiert. Dabei treten oft rein duale Darstellungen in Verbindung mit BCD-Darstellungen oder Textdarstellungen (ASCII-Code) auf. Zur Realisierung einheitlicher Verarbeitung sind Umwandlungen von einer in die andere Darstellung unumgänglich. Im folgenden werden einige solcher Umwandlungen und Operationen für die betreffenden Zahlendarstellungen gezeigt.

Umwandlung BCD – DUAL
Bei der Umwandlung BCD (Dezimal) – DUAL unterscheidet man u. a. die Verfahren für den ganzen Teil und den gebrochenen Teil der Zahl. Daher erfolgt die Umsetzung von ganzem und gebrochenem Teil meistens getrennt.

Beispiel 1
Umwandlung einer ganzen BCD-Zahl (4 Stellen ohne Vorzeichen) in eine Dualzahl
Eingangsparameter: BCD-Zahl in HL
Ausgangsparameter: DUAL-Zahl in HL
Verfahren:
Schreibt man die Dezimalzahl mit den Ziffern $d_3\ d_2\ d_1\ d_0$ in der Form

$$b = d_3 \cdot 10^3 + d_2 \cdot 10^2 + d_1 \cdot 10^1 + d_0 = (((0 \cdot 10 + d_3) \cdot 10 + d_2) \cdot 10 + d_1) \cdot 10 + d_0$$

so erkennt man folgenden Algorithmus:



Dabei ist mit der höchstwertigen Ziffer (d_3) zu beginnen. Vor dem Abarbeiten des Algorithmus sind die Ziffern zu trennen.

```
        PN    A1
;UMWANDLUNG BCD-DUAL
;UMWANDLUNG EINER INTEGER BCD-ZAHL (4 STELLEN UNSIGNIERT)
;IN EINE DUALZAHL (16 BIT)

DBI2:   PUSH HL
        LD    HL,0
        ADD   HL,SP
        XOR   A                 ;ABLEGEN DER ZIFFERN D3, D2, D1, D0
        RRD                     ;IN DEN STACK
        PUSH AF                 ;ZIFFER D0 IN DEN STACK
        RRD
        PUSH AF                 ;ZIFFER D1 IN DEN STACK
        INC   HL
        RRD
        PUSH AF                 ;ZIFFER D2 IN DEN STACK
        RRD
        PUSH AF                 ;ZIFFER D3 IN DEN STACK
;ALGORITHMUS
        LD    HL,0              ;B = 0
        LD    B,4               ;ZAEHLER = 4
ZYK     LD    E,L
        LD    D,H
        ADD   HL,HL             ;MULTIPLIKATION MIT 10
        ADD   HL,HL
        ADD   HL,DE
        ADD   HL,HL
        POP   DE                ;ZIFFER HOLEN
        LD    E,D
        LD    D,0
        ADD   HL,DE             ;ZIFFER ADDIEREN
        DJNZ  ZYK #
        POP   DE
        RET
        END
```

Beispiel 2

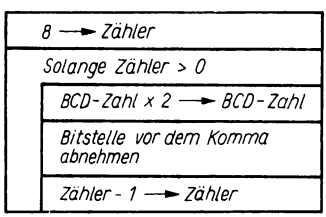
Umwandlung einer gebrochenen BCD-Zahl (2 Stellen ohne Vorzeichen) in eine Dualzahl mit 8 Stellen.

Eingangsparameter: BCD-Zahl in Register A

Ausgangsparameter: Dual-Zahl in Register A

Verfahren:

Durch Multiplikation mit 2 erscheinen die Dualziffern der Reihe nach vor dem Komma. Für 8 Dualstellen ergibt sich folgender Algorithmus:



Durch (BCD-Zahl·2 = BCD-Zahl + BCD-Zahl) wird die Bit-Stelle vor dem Komma in das CY-Bit gebracht. Diese Stellen brauchen nur noch in ein freies Register (z. B. Register C) der Reihe nach von rechts nach links eingeschoben werden.

```
        FN    A2
;UMWANDLUNG EINER BCD-ZAHL /Z/ < 1 (2 STELLEN UNSIGNIERT)
;IN EINE DUALZAHL (8 BIT)

DBI1:   LD    B,8             ;ZAEHLER = 8
MA1:    ADD   A               ;WERT * 2
        DAA
        RL    C
        DJNZ  MA1-#           ;RUECKSPRUNG, SOLANGE ZAEHLER   0
        LD    A,C             ;RESULTAT NACH A
        RET
        END

FUER DIE KONVERTIERUNG MIT RUNDUNG FOLGT NACH DEM BEFEHL DJNZ MA1-#

        ADD   A               ;ZUSAETZLICHE BCD-ADD ZUM EVENTUELLEN
        DAA                   ;AUFRUNDEN
        LD    A,C             ;RESULTAT NACH A
        ADC   00              ;AUFRUNDEN
        RET
        END
```

Umwandlung DUAL – BCD

Auch hier wird i. a. der ganze und der gebrochene Teil einer Zahl getrennt behandelt.

Beispiel 3

Umwandlung einer ganzen Dualzahl (16 Bit) in eine BCD-Zahl mit 5 Stellen.

Eingangsparameter: Dualzahlen in HL

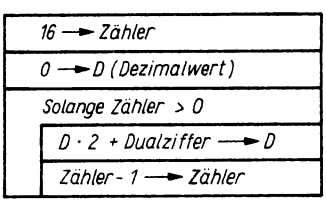
Ausgangsparameter: BCD-(Dezimal-)Zahl in AHL

Verfahren:

Schreibt man die Dualzahl mit den Ziffern $b_{15}\ b_{14}\ b_{13} \ldots b_1\ b_0$ in der Form

$$D = b_{15} \cdot 2^{15} + b_{14} \cdot 2^{14} + \ldots b_1 \cdot 2^1 + b_0 = (\ldots((0 \cdot 2 + b_{15}) \cdot 2 + b_{14}) \cdot 2 + \ldots b_1) \cdot 2 + b_0,$$

so erkennt man folgenden Algorithmus:



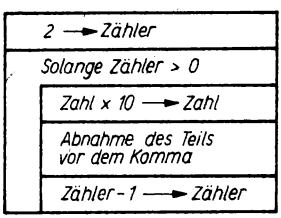
Die Dualziffern müssen mit der höchstwertigen beginnen. Durch Dualwert + Dualwert (HL + HL) kommen die Dualziffern der Reihe nach in das CY-Flag, das nur noch zu 2D addiert zu werden braucht.

```
        PN   A3
;DUAL - BCD
;UMWANDLUNG EINER DUALZAHL (INTEGER 16 BIT UNSIGNIERT)
;IN EINE BCD-ZAHL (5 STELLEN)
;DUALZAHL IN HL
;BCD-ZAHL IN AHL

BDI2:   XOR  A
        LD   D,A
        LD   B,16
ZYK:    ADD  HL,HL
        ADC  A
        DAA
        LD   E,A
        LD   A,D
        ADC  A
        DAA
        LD   D,A
        RL   C
        LD   A,E
        DJNZ ZYK-#
        EX   DE,HL
        LD   A,C
        RET
        END
```

Beispiel 4

Umwandlung einer gebrochenen Dualzahl mit 8 Bit ohne Vorzeichen in eine BCD-Zahl mit 2 Stellen.
Eingangsparameter: Dualzahl in Register A
Ausgangsparameter: BCD-Zahl in Register A



Verfahren: Multipliziert man den gebrochenen Teil einer Zahl mit 10, so tritt die vorderste Dezimalziffer vor das Komma.

```
        PN   A4
;UMWANDLUNG EINER DUALZAHL ECHT GEBROCHEN 8 BIT
;IN EINE DEZIMALZAHL 2 STELLEN
;DEZIMALZAHL UND DUALZAHL IN A

UDBE1:  LD   B,02H          ;ZAEHLER = 2
        LD   H,A            ;DUALZAHL NACH H
        XOR  A
        LD   L,A
ZYK:    LD   E,L
        LD   D,H            ;DUALZAHL NACH D
        RLA                 ;ZWISCHENWERT MAL 2
        ADD  HL,HL
        ADC  A
        ADD  HL,HL          ;DUALZAHL MAL 10
        ADC  A
        ADD  HL,DE
```

```
        ADC   00H
        ADD   HL,HL
        ADC   A
        DJNZ  ZYK-#             ;ZAEHLER ABARBEITEN
        ADD   HL,HL
        ADC   00H               ;EVENTUELL AUFRUNDEN
        RET
        END
```

Beispiel 5

Umwandlung einer echt gebrochenen Dualzahl mit 16 Bit ohne Vorzeichen in eine BCD-Zahl mit 4 Stellen.

Eingangsparameter: Dualzahl in HL

Ausgangsparameter: BCD-Zahl in HL

Das Programm läuft nach dem gleichen Verfahren wie Beispiel 4, nur wird der Zähler am Anfang auf 4 gestellt.

```
        PN    A5
;UMWANDLUNG DUAL NACH BCD
;16 BIT DUALZAHL UNSIGNIERT ERGIBT 4-STELLIGE BCD-ZAHL (ECHT GEBROCHEN)
;DUALZAHL IN HL
;BCD-ZAHL IN HL
;SONDERFALL: HL = 0000H, CY = 1 BEDEUTET, DASS DAS GERUNDETE
;BCD-RESULTAT (1,)0000BCD BETRAEGT. SONST CY = 0.

BDE2:   LD    B,04H             ;ZAEHLER = 4
        XOR   A
ZYK:    SL    A
        RL    C
        LD    E,L
        LD    D,H
        ADD   HL,HL
        RLA
        RL    C
        ADD   HL,HL
        RLA
        RL    C
        ADD   HL,DE
        ADC   00H
        ADD   HL,HL
        RLA
        RL    C
        DJNZ  ZYK-#
        ADD   HL,HL
        ADC   00H
        DAA
        LD    L,A
        LD    A,C
        ADC   00H
        DAA
        LD    H,A
        RET
        END
```

Beispiel 6

Zur Ausgabe einer Zahl auf Bildschirm oder Drucker muß die Zahl im ASCII-Code vorliegen.

Lösung:


```
        PN   A6
;UMWANDLUNG EINER GANZEN POSITIVEN DUALZAHL
;IN DIE ASCII-DARSTELLUNG
;EINGABEPARAMETER: DUALZAHL IN HL
;                  ADRESSE ASCII-PUFFER IN DE
;AUSGABEPARAMETER: ASCII-ZAHL IM PUFFER

DAS2:   PUSH DE
        XOR  A
        LD   D,A
        LD   B,10H          ;ZAEHLER FUER 16 BIT
ZYK     ADD  HL,HL
        ADC  A
        DAA
        LD   E,A
        LD   A,D
        ADC  A
        DAA
        LD   D,A
        RL   C              ;UEBERTRAG AUS REGISTER DE NACH REGISTER C
        LD   A,E
        DJNZ ZYK-#
        LD   A,C
        EX   DE,HL
;PROGRAMMABSCHNITT 2: UMWANDLUNG BCD - ASCII
        LD   C,05H          ;5 BYTE ASCII
        POP  DE             ;ADRESSE ZEICHENPUFFER
ZO:     ADD  30H            ;BILDUNG ASCII-ZEICHEN
        LD   (DE),A         ;ZEICHEN IN DEN PUFFER
        DEC  C
        RZ                  ;NACH 5 ZEICHEN FERTIG
        INC  DE
        XOR  A
        LD   B,04H          ;4-BIT-BCD-ZIFFER NACH A
Z1:     ADD  HL,HL
        ADC  A
        DJNZ Z1-#
        JR   ZO-#
        END
```

9.2. Rechenprogramme

Im Befehlsschlüssel des *U880* sind keine Multiplikation und Division enthalten. Diese Operationen müssen durch Programme realisiert werden. Das betreffende Programm hängt natürlich von der Darstellung der Zahlen ab.

Beispiel 7
Multiplikation von 2 ganzen Dualzahlen (16 Bit) ohne Vorzeichen. Das Ergebnis ist 32 Bit lang.
Eingangsparameter: Multiplikand in DE
Multiplikator in BC
Ausgangsparameter: Produkt in HL BC
Verfahren:
Schreiben wir den Multiplikator (MR) in dualer Darstellung

$$MR = b_{15} \cdot 2^{15} + b_{14} \cdot 2^{14} + \ldots b_1 \cdot b^1 + b_0,$$

so erkennen wir aus

$$\begin{aligned} MD \cdot MR &= MD\,(b_{15} \cdot 2^{15} + b_{14} \cdot 2^{14} + \ldots b_1 \cdot 2^1 + b_0) \\ &= 2^{15}b_{15}MD + 2^{14}b_{14}MD + \ldots 2^1 b_1 MD + b_0 MD, \end{aligned}$$

daß wir zur Produktbildung MD mit einer Verschiebung 2^i zum Teilprodukt addieren müssen, wenn die Stelle $b_i = 1$ ist. Statt einer Verschiebung von MD nach links ($\times 2^i$) können wir auch das Produkt nach rechts verschieben. Die Multiplikation läuft damit nach Schema (Bild 9.1) ab.

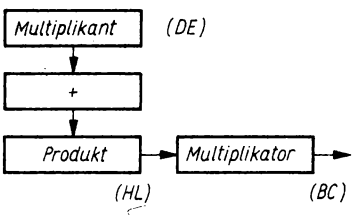
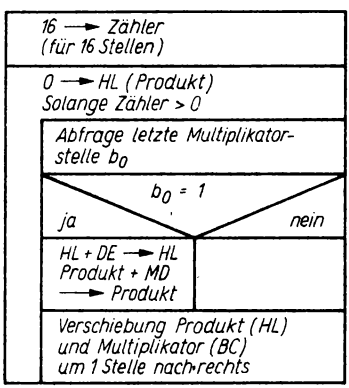


Bild 9.1
Funktionsprinzip für die Multiplikation

Der Ablauf läßt sich wie folgt darstellen:



Da sich die Abfrage der letzten Stelle des Multiplikators b_0 leichter über das CY-Bit realisieren läßt, kann man vor Beginn des Verfahrens den Multiplikator so um eine Stelle nach rechts schieben, daß die letzte Stelle ins CY-Bit kommt. Dadurch entsteht der als Flußbild (Bild 9.2) dargestellte Ablauf.
Zur Multiplikation wird der Multiplikator eine Stelle nach rechts geschoben. Dabei gelangt das niederwertigste Bit in das C-Bit. Ist dieses Bit 1, wird der Inhalt von DE zu HL addiert, ist es 0, erfolgt keine Addition von DE zu HL. Anschließend wird der Inhalt von HL mit BC gemeinsam um eine Stelle nach rechts verschoben. Dabei kommt das nächste Bit des Multiplikators ins CY-Bit. Dieser Vorgang wird insgesamt 16mal wiederholt, da der Multiplikator 16 Stellen umfaßt.
Weil am Anfang einmal BC nach rechts verschoben werden muß, um die 1. Stelle ins CY-Bit zu bringen, wird der Zähler auf 17 gestellt.

```
        PN    B1
;BASISPROGRAMM FUER MULTIPLIKATION
;16 X 16 BIT-MULTIPLIKATION, UNSIGNIERT, ERGEBNIS 32 BIT
;MULTIPLIKAND IN DE
;MULTIPLIKATOR IN BC

MUL2:   LD    HL,0
        LD    A,17            ;ZAEHLER
K3:     RR    B
        RR    C
        DEC   A
        RZ
        JRNC  K7-#
        ADD   HL,DE
K7:     RR    H
        RR    L
        JR    K3-#
        END
```

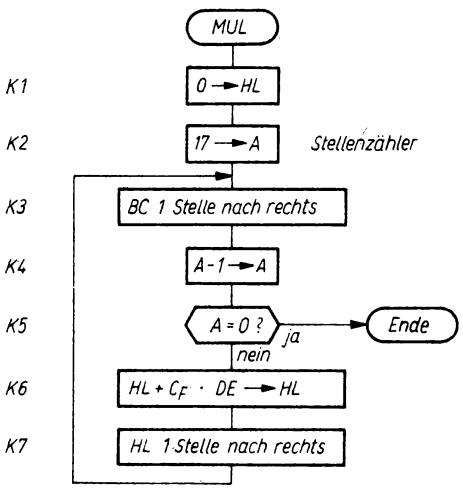


Bild 9.2
Flußdiagramm für die Multiplikation von 2 16stelligen Dualzahlen

- Division

Die Division ist die Umkehrung der Multiplikation. Es soll eine Division mit einem Dividenden (DV) von 32 Bit und einem Divisor (DR) von 16 Bit betrachtet werden. Der Quotient (Q) möge 16 Bit sein.
Dann gilt

$$DV : DR = Q \text{ Rest } R$$

oder

$$\frac{DV}{DR} = Q + \frac{R}{DR}.$$

Bringt man die Gleichung auf die Form

$$R = DV - Q \cdot DR$$

und schreibt den Quotienten Q als Dualzahl

$$Q = Q_{15} \cdot 2^{15} + Q_{14} \cdot 2^{14} + \ldots Q_1 \cdot 2^1 + Q_0,$$

so kann man unser Verfahren aus der Gleichung

$$R = DV - 2^{15}Q_{15}DR - 2^{14}Q_{14}DR - \ldots 2^{1}Q_{1}DR - Q_{0}DR$$

ableiten.
Man probiert, ob $2^{15} \cdot DR$ vom Dividenden DV abzuziehen geht. Wenn »ja«, ist $Q_{15} = 1$, und es werden $2^{15} \cdot DR$ abgezogen; wenn »nein«, ist $Q_{15} = 0$, und $2^{15} \cdot DR$ werden nicht abgezogen. Danach verfährt man ebenso mit $2^{14} \cdot DR$ bis $2^{0} \cdot DR$. Für die Division ergibt sich damit folgendes Blockschema (Bild 9.3).

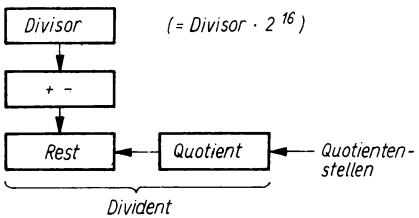


Bild 9.3
Funktionsprinzip für die Division

1. Bildung von Dividend – Divisor
Ist das Ergebnis negativ, so wird der Divisor wieder zum Ergebnis dazugezählt (Rückstellung des Restes) und in die letzte Stelle des Quotienten eine 0 eingetragen.
Ist das Ergebnis positiv, so wird keine Rückstellung des Restes vorgenommen. In die letzte Quotientenstelle trägt man eine 1 ein.
2. Quotient und Dividend werden gemeinsam 1 Stelle nach links verschoben.
3. 1. und 2. werden so oft wiederholt, wie der Quotient Stellen haben soll.
Am Ende steht der Quotient im Quotientenregister und im Dividendenregister der Rest.
Da im Divisorregister der Divisor $\cdot 2^{16}$ steht und wir mit einem 16-Bit-Divisor arbeiten, müssen Punkt 1 und 2 einmal zusätzlich durchlaufen und die nach der ersten Linksverschiebung aus dem Restregister austretende vorderste Stelle in einem weiteren Register aufgefangen werden.

Beispiel 8
Division einer ganzen Zahl (32 Bit) durch eine ganze Zahl (16 Bit). Der Quotient soll 16 Bit Datenbreite haben.
Eingangsparameter: Dividend in HL BC
Divisor in DE
Ausgangsparameter: Quotient in BC

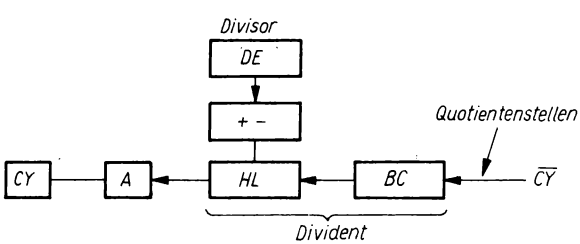


Bild 9.4
Blockschema für Division 32 Bit:
16 Bit → 16 Bit ohne Vorzeichen mit *U880*

Bild 9.4 zeigt das Divisionsschema, wenn wir dem Dividenden, dem Divisor und dem Quotienten bestimmte Register zuweisen.
Als zusätzliches Register zum Auffangen der vordersten Dividendenstelle nehmen wir Register A'. Im CY-Flag entsteht beim Addieren oder Subtrahieren das Vorzeichen des Restes. Ist dieses Vorzeichen 1, ist die Quotientenstelle 0 und umgekehrt. Daher können wir $\overline{CY}$ als Quotientenstelle benutzen. Bild 9.5 zeigt den Ablauf als Struktogramm.

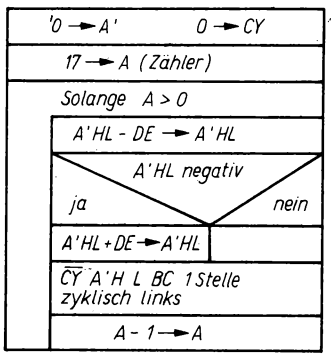


Bild 9.5
Struktogramm für die Division

```
        PN   B2
;32 BIT : 16 BIT-DIVISION, UNSIGNIERT, ERGEBNIS 16 BIT
;DIVIDEND 32 BIT IN HL,BC
;DIVISOR 16 BIT IN DE
;QUOTIENT 16 BIT IN BC

DIV:    XOR  A
        EXAF                    ;A = 0
        LD   A,17               ZAEHLER = 17
K3:     EXAF
        SBC  HL,DE              ;SUBTRAKTION A HL - DE = A HL
        SBC  0
        JRNC K6-#
        ADD  HL,DE              ;ADDITION, WENN ERGEBNIS NEGATIV
        ADC  0
K6:     CCF                     ;CY - CY
        RL   C                  ;VERSCHIEBUNG
        RL   B
        RL   L
        RL   H
        RLA
        EXAF
        DEC  A                  ;A - 1 = A
        JRNZ K3-#
        RET
        END
```

In Bild 9.6 ist der Ablauf als Flußbild gegenübergestellt. Zur Beschreibung von Programmabläufen wird sowohl das Flußbild als auch die Darstellung als Struktogramm verwendet. Die Flußbilddarstellung ist für den Anfänger leichter verständlich. Man sollte jedoch die Struktogrammdarstellung anstreben, da diese schärfer den Algorithmus wiedergibt und außerdem zu übersichtlicheren Beschreibungen und Programmen führt.
Das zusätzliche Register A' wird nicht benötigt, wenn wir statt 16-Bit-Divisor nur einen 15-Bit-Divisor und statt einem 32-Bit-Dividenden einen 30-Bit-Dividenden nehmen.

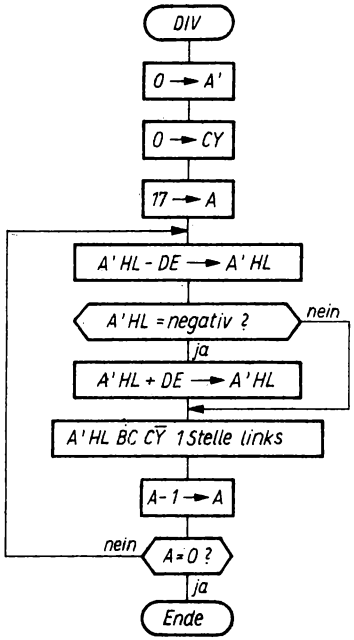


Bild 9.6
Flußdiagramm zur Division

Beispiel 9

Division einer ganzen Zahl 30 Bit durch eine ganze Zahl 15 Bit. Der Quotient ist 15 Bit.
Eingangsparameter: Dividend in HL BC
Divisor in DE
Ausgangsparameter: Quotient in BC
Das Divisionsschema bleibt wie in Bild 9.4.

```
        PN   B3
;DIVISION, GANZE ZAHLEN 30 BIT : 15 BIT GIBT 15 BIT
;DIVIDEND IN HL BC
;DIVISOR IN DE
;QUOTIENT IN BC

DIV2:   LD   A,17           ;ZAEHLER
        OR   A              ;CY = 0
ZYK:    SBC  HL,DE
        JRNC KRUE-#         ;BEI CY = 0 KEINE RUECKSTELLUNG
        ADD  HL,DE          ;RESTRUECKSTELLUNG
KRUE:   CCF
        RL   C              ;VERSCHIEBUNG DIVIDEND
        RL   B
        ADC  HL,HL
        DEC  A
        JRNZ ZYK-#
        RET
        END
```

Da der Divisor gegenüber dem Dividenden um 16 statt um 14 Stellen nach links verschoben ist, muß der Zähler am Anfang auf 17 und nicht auf 15 gestellt werden.
Im anderen Fall müßte vorher der Dividend um 2 Stellen nach links verschoben werden, da sich der Divisor in DE nicht weiter nach rechts schieben läßt.

$$Q = Q_{14} \cdot 2^{14} + \ldots Q_0 .$$

Bis jetzt haben wir die Multiplikation und Division ohne Vorzeichen betrachtet. Bei einer Multiplikation und Division mit Vorzeichen wird meistens Vorzeichen und Betrag getrennt berechnet. Liegt eine negative Zahl als Zweierkomplement vor, muß zunächst der Betrag gebildet werden. Die Betragsbildung kann durch die Operation 0 − Zahl,
wobei die Zahl im Zweierkomplement vorliegt, erfolgen. Die Vorzeichenberechnung ist für Multiplikation und Division gleich, und zwar gilt:

1. Operand	2. Operand	Ergebnis
+	+	+
+	−	−
−	+	−
−	−	+

Da bei negativen Vorzeichen die Vorzeichenstelle »1« und bei positiven Vorzeichen »0« ist, gilt:
Ergebnisvorzeichen = Vorzeichen 1. Operand ⊕ Vorzeichen 2. Operand, wobei ⊕ das Exklusiv-Oder (XOR-Befehl) ist.

Beispiel 10
Multiplikation zweier Zahlen mit Vorzeichen (Zweierkomplement) 15 Bit plus Vorzeichen Ergebnis 30 Bit plus Vorzeichen
Eingangsparameter: Multiplikator in BC
Multiplikand in DE
Ausgangsparameter: Produkt in HL BC

```
        PN   B4
;MULTIPLIKATION MIT VORZEICHEN (ZWEIERKOMPLEMENT)
;DE X BC NACH HLBC

MUL2V: LD   A,D             ;BERECHNUNG ERGEBNISVORZEICHEN
       XOR  B
       AND  80H
       PUSH AF
       BIT  7,D             ;BETRAG MULTIPLIKAND
       JRZ  M3-#
       LD   HL,0
       SBC  HL,DE
       EX   DE,HL
M3:    BIT  7,B             ;BETRAG MULTIPLIKATOR
       JRZ  M4-#
       LD   HL,0
       XOR  A
       SBC  HL,BC
       LD   B,H
       LD   C,L
M4:    CALL MUL2            ;PRODUKT DER BETRAEGE
       POP  AF              ;KOMPLEMENT PRODUKT
       BIT  7,A
       JRZ  M5-#
       SUB  C
       LD   C,A
```

```
        LD    A,0
        SBC   B
        LD    B,A
        LD    A,0
        SBC   L
        LD    L,A
        LD    A,0
        SBC   H
        LD    H,A
M5:     RET
;MULTIPLIKATIONSROUTINE
MUL2:   LD    HL,0
        LD    A,17
K3:     RR    B
        RR    C
        DEC   A
        RZ
        JRNC  K7-#
        ADD   HL,DE
K7:     RR    H
        RR    L
        JR    K3-#
        END
```

Für die Multiplikation der Beträge wurde die Routine von Beispiel 7 genommen.

Beispiel 11

Division zweier Zahlen mit Vorzeichen (Zweierkomplement)
Eingangsparameter: Dividend in HL BC (30 Bit + Vorzeichen)
Divisor in DE (15 Bit + Vorzeichen)
Ausgangsparameter: Quotient in BC (15 Bit + Vorzeichen)

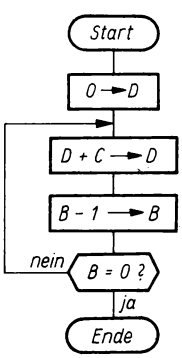
```
        PN    B5
;DIVISIONSROUTINE MIT VORZEICHEN (ZWEIERKOMPLEMENT)
;32 BIT : 16 BIT ERGIBT 16 BIT
;HLBC : DE NACH BC

DIV2U:  LD    A,H             ;BERECHNUNG ERGEBNISVORZEICHEN
        XOR   D
        AND   80H
        PUSH  AF
        BIT   7,D             ;BETRAG DIVISOR
        JRZ   M8-#
        PUSH  HL
        LD    HL,0
        SBC   HL,DE
        EX    DE,HL
        POP   HL
M8:     BIT   7,H             ;BETRAG DIVIDEND
        JRZ   M9-#
        LD    A,0
        SUB   C
        LD    C,A
        LD    A,0
        SBC   B
        LD    B,A
        LD    A,0
        SBC   L
        LD    L,A
        LD    A,0
        SBC   H
        LD    H,A
M9:     CALL  DIV2            ;DIVISION DER BETRAEGE
        POP   AF              ;BETRAG QUOTIENT
        BIT   7,A
        JRZ   M10-#
```

```
        LD    A,0
        SUB   C
        LD    C,A
        LD    A,0
        SBC   B
        LD    B,A
        RET                     ;DIVISIONSROUTINE
DIV2:   LD    A,17
        OR    A
ZYK:    SBC   HL,DE
        JRNC  KRUE-#
        ADD   HL,DE
KRUE:   CCF
        RL    C
        RL    B
        ADC   HL,HL
        DEC   A
        JRNZ  ZYK-#
M10:    RET
        END
```

Für die Division der Beträge wurde die Routine aus Beispiel 9 verwendet.
Für die Arbeit mit BCD-Zahlen soll ein Programm zur Realisierung des kleinen Einmaleins dienen.

Beispiel 12
Multiplikation von 2 ganzen Dezimalzahlen (BCD-Code) mit 2 Stellen. Das Produkt darf maximal ebenfalls 2 Stellen betragen. Da der Rechner sehr schnell arbeitet (pro Befehl etwa 5 µs), führen wir die Multiplikation mit kleinen Zahlen durch einfache Addition durch.
Eingangsparameter: Multiplikand 2 BCD-Stellen in C
Multiplikator 2 BCD-Stellen in B
Ausgangsparameter: Produkt 2 BCD-Stellen in D
Bild 9.7 zeigt das Flußbild.




```
        PN    B6
;MULTIPLIKATION VON 2 BCD-ZAHLEN MIT 2 STELLEN
;MULTIPLIKAND IN REGISTER C
;MULTIPLIKATOR IN REGISTER B
;PRODUKT IN REGISTER D (MAXIMAL 2 STELLEN)

        LD    D,0               ;PRODUKT = 0 SETZEN
M1:     LD    A,D               ;ADDITION MULTIPLIKAND
        ADD   C
        DAA
        LD    D,A
        LD    A,B               ;ZAEHLUNG MULTIPLIKATOR
        SUB   1
        DAA
        LD    B,A
        JRNZ  M1-#
        RET
        END
```

Bild 9.7
Flußbild zur Multiplikation von 2 BCD-Zahlen mit 2 Stellen

Für die Arbeit mit Gleitkommazahlen betrachten wir die Multiplikation. Da eine Gleitkommazahl aus Mantisse und Exponent besteht, gilt:

$$z1 = m_1 \cdot 2^{E_1}$$
$$z2 = m_2 \cdot 2^{E_2}$$

und für das Produkt

$$z = z1 \cdot z2 = m1 \cdot 2^{E_1} \cdot m_2 \cdot 2^{E_2} = m_1 \cdot m_2 \cdot 2^{E_1 + E_2},$$

d. h., wir müssen die Mantissen multiplizieren und die Exponenten addieren. Die Mantisse ist normalisiert. Eine Zahl normalisieren heißt sie so umformen, daß die erste Stelle nach dem Komma 1 wird.

Beispiel 13

```
        PN   B7
;GLEITKOMMAMULTIPLIKATION, 2 BYTE MANTISSE, 1 BYTE EXPONENT
;ZAHLENDARSTELLUNG: MANTISSE NORMALISIERT, BETRAG + VORZEICHEN
;                   (15 + 1 BIT)
;EXPONENT:          ZWEIERKOMPLEMENT (7 + 1 BIT)
;EINGANGSPARAMETER: MANTISSE MULTIPLIKAND IN DE
;                   EXPONENT MULTIPLIKAND IN H
;                   MANTISSE MULTIPLIKATOR IN BC
;                   EXPONENT MULTIPLIKATOR IN L
;AUSGANGSPARAMETER: MANTISSE PRODUKT IN HL
;                   EXPONENT PRODUKT IN A
;BEI UEBERLAUF IST DAS P/V-FLAG GESETZT, SONST RUECKGESETZT

;BERECHNUNG VORZEICHEN
GUM2:   LD   A,D
        XOR  B
        AND  80H            ;VORZEICHEN ERGEBNISMANTISSE AUF BIT 8
        RES  7,B            ;VORZEICHEN MULTIPLIKATOR LOESCHEN
        RES  7,D            ;VORZEICHEN MULTIPLIKAND LOESCHEN
        PUSH HL
        PUSH AF
;MULTIPLIKATION MANTISSEN
MU2:    LD   HL,0
        LD   A,11H
K3:     RR   B
        RR   C
        DEC  A
        JRZ  NRM-#
        JRNC K7-#
        ADD  HL,DE
K7:     RR   H
        RR   L
        JR   K3-#
;NORMALISIERUNG UND STELLENRICHTIGE VERSCHIEBUNG
NRM:    LD   C,1
SH:     RL   B
        RL   L
        RL   H
        DEC  C
        BIT  6,H
        JRZ  SH-#
;RUNDUNG
KN:     RL   B
        LD   DE,0
        ADC  HL,DE
        BIT  7,H
        JRZ  VZ-#
;NORMALISIERUNG NACH DER RUNDUNG
        RR   H
        INC  C
```

```
;VORZEICHEN EINSETZEN
VZ:     POP   AF
        OR    H
        LD    H,A
;BERECHNUNG DES EXPONENTEN
EXP:    POP   DE
        LD    A,E
        ADD   D
        JPPE  END              ;UEBERLAUF
        ADD   C
END:    RET
        END
```

Quadratwurzel

Das folgende Programm soll zeigen, wie höhere Operationen auf einem Mikrorechner programmiert werden können. Wichtig dabei ist, daß man zunächst ein Lösungsverfahren findet, das möglichst an das Dualsystem angepaßt ist. Es gilt:

$$Z = X$$

$$Z = 0, \quad Z_1 Z_2 Z_3 \ldots Z_N \qquad X = 0, \quad X_1 X_2 X_3 \ldots X_N$$

Verfahren:

$$P_0 = Z \qquad X_0 = 0$$

$$P_K = 2\left[P_{K-1} - \operatorname{sgn}(P_{K-1})\left(2^{-K} + \sum_{j=1}^{K-1} X_j \cdot 2^{-j}\right) + 2^{-K-1}\right].$$

$$X_K = \frac{1 + \operatorname{sgn}(P_K)}{2} \qquad \operatorname{sgn}(P) = \begin{cases} +1, \text{ falls P positiv} \\ -1, \text{ falls P negativ} \end{cases}$$

Das Verfahren ist ein Iterationsverfahren. Die Zahl Z, aus der die Wurzel gezogen wird, muß kleiner als 1 sein. Sie liegt als reine Dualzahl mit n Dualstellen Z_1, Z_2 bis Z_n vor. Die Wurzel aus Z hat wieder n Dualstellen X_1, X_2 bis X_N. Im Verfahren geht man von der Zahl $P_0 = Z$ aus. Die Ziffer X_0 (Stelle vor dem Komma) ist 0. Mit P_0 und X_0 wird nach den angegebenen Formeln P_1 und X_1 berechnet. X_1 ist die 1. Stelle der Lösung hinter dem Komma. Mit P_1 und X_1 berechnet man P_2 und X_2 usw. Mit jedem Schritt gewinnt man eine Dualstelle der Wurzel.

Um das Verfahren »rechnergerecht« aufzubereiten, wird die Formel etwas umgestellt. Es sei angenommen, daß $P_0 P_1$ bis P_{K-1} und $X_0 X_1$ bis X_{K-1} bereits ermittelt worden sind und nun P_K und X_K ermittelt werden sollen.

Wenn man P_{K-1} als Zahl schreibt, gilt:

$$P_{K-1} = 0, \qquad S_1 S_2 \ldots S_{K-1} S_K S_{K+1} \ldots S_n,$$

wobei $S_1 S_2 \ldots S_n$ die Dualstellen von P_{K-1} sind.

Weiterhin gilt:

$$\sum_{j=1}^{K-1} X_j \cdot 2^{-1} = 0, \qquad X_1 X_2 \ldots X_{K-1}.$$

Ist P_{K-1} positiv, so ist $\operatorname{sgn}(P_{K-1}) = 1$ und

$$P_K = 2\left[P_{K-1} - \sum_{j=1}^{K-1} X_j 2^{-j} - 2^{-K-1}\right],$$

$$P_K = 2\left[P_{K-1} - \sum_{j=1}^{K-1} X_j 2^{-j}\right] - 2^{-K}.$$

Ist P_{K-1} negativ, so ist $\operatorname{sgn}(P_{K-1}) = -1$ und

$$P_K = 2\left[P_{K-1} + \sum_{j=1}^{K-1} X_j 2^{-j} + 2^{-K} + 2^{-K-1}\right].$$

Wird P_K negativ, so gilt mit P_K negativ $\operatorname{sgn}(P_K) = -1$; $X_K = 0$:

$$P_{K+1} = 2\left[2P_K + \sum_{j=1}^{K} X_j 2^{-j} + 2^{-K-1} + 2^{-K-2}\right] \quad \text{und}$$

$$P_K = 2\left[P_{K-1} - \sum_{j=1}^{K-1} X_j 2^{-j} - 2^{-K-1}\right]. \quad \text{Und damit}$$

$$P_{K+1} = 2\left[2P_{K-1} - 2\sum_{j=1}^{K-1} X_j 2^{-j} - 2^{-K} + \underbrace{\sum_{j=1}^{K-1} X_j 2^{-j}}_{\text{da } X_K = 0} + 2^{-K-1} + 2^{-K-2}\right],$$

$$P_{K+1} = 2\left[2P_{K-1} - \sum_{j=1}^{K-1} X_j 2^{-j} - 2^{-K-2}\right] = 2\left[2P_{K-1} - \sum_{j=1}^{K-1} X_j 2^{-j}\right] - 2^{-K-1}.$$

Man erhält folgendes Verfahren zum Radizieren von Quadratwurzeln:

1. Voraussetzungen	Dualwert
a) Bereitstellen von $P_0 = Z$	$0\ Z_1\ Z_2 \ldots Z_n$
b) $X = 0$, $X_1X_2 \ldots X_n = 0$ setzen	$0\ 0\ 0\ \ldots 0$
c) Bereitstellen einer Konstanten K zum Bit der Ziffer X_K	$0\ 1\ 0\ \ldots 0$

2. Rechenablauf

Ausgangspunkt P_{K-1}; am Anfang P_0

a) Bildung von $2P_{K-1}$

b) Bildung von

$$P_K = 2\left[P_{K-1} - \sum_{j=1}^{K-1} X_j 2^{-j}\right] - 2^{-K}$$

c) Ist P_K positiv, so wird $X_K = 1$ gesetzt (Bildung von $X + K \rightarrow X$), und statt P_{K-1} wird P_K gesetzt.

d) Ist P_K negativ, so wird $X_K = 0$ gesetzt, und statt P_K wird $2P_{K-1}$ gesetzt.

e) K wird um eine Stelle nach rechts verschoben.

f) Fortführung bei a) so lange, bis die Anzahl der Stellen genügt.

Das folgende Beispiel zeigt das Programm für die Quadratwurzel aus einer 15stelligen Dualzahl $|z| < 1$.

Beispiel 15

Quadratwurzel aus einer echtgebrochenen Zahl 15 Bit (Bild 9.8)

Eingangsparameter: Radikand in HL, Komma steht nach Bit 15

Ausgangsparameter: Wurzel in DE
Komma steht nach Bit 15

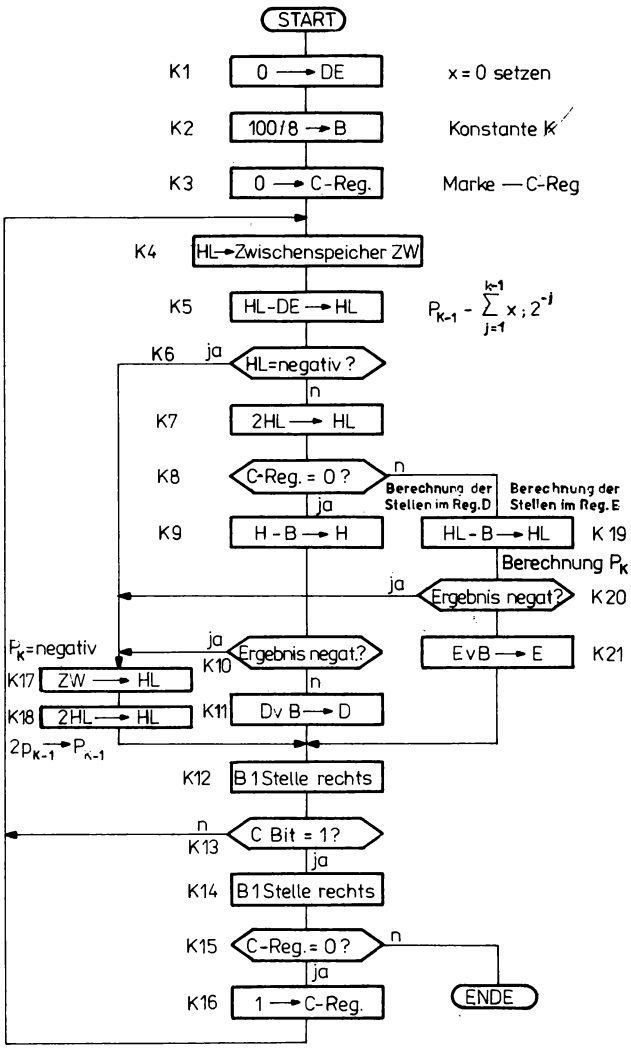


Bild 9.8
Flußdiagramm für das Quadratwurzelziehen aus einer 15stelligen Dualzahl mit $|z| < 1$

```
        PN    W2
;WURZELPROGRAMM
;RADIKAND     0, D1...D15 IN HL
;WURZEL       0, W1...W15 IN DE

WU2R:   LD    BC,4000H
        LD    DE,0
K4:     LD    (ZW),HL
        XOR   A
        SBC   HL,DE
        JRC   K17-#             ;ABFRAGE K6
        RL    L
        RL    H
```

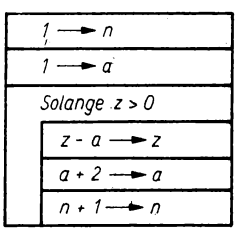
```
        XOR   A
        XOR   C
        JRNZ  K19-#           ;ABFRAGE
        LD    A,H
        SUB   B
        LD    H,A
        JRC   K17-#           ;ABFRAGE K10
        LD    A,D
        OR    B
        LD    D,A
K12:    RR    B
        JRNC  K4-#            ;ABFRAGE K13
        RR    B
        XOR   A
        ADD   C
        RNZ                   ;ABFRAGE K15
        LD    C,1
        JR    K4-#
K17:    LD    HL,(ZW)
        SLA   L
        RL    H
        JR    K12-#
K19:    LD    A,L
        SUB   B
        LD    L,A
        LD    A,H
        SBC   0
        LD    H,A
        JRC   K17-#           ;ABFRAGE K20
        LD    A,E
        OR    B
        LD    E,A
        JR    K12-#
ZW:     DA    0
        END
```

Beispiel 16

Quadratwurzel aus einer ganzen 4stelligen Dezimalzahl. Geht man von der arithmetischen Reihe

$$1 + 3 + 5 + \ldots + (2n - 1) = n^2, \qquad a = 2n - 1$$

aus, so kann man die Wurzel aus einer Zahl z mit folgendem Algorithmus ermitteln:



Während man die Glieder der Reihe a vom Radikanden abzieht, zählt man n aufwärts, bis sich kein nächstes Glied a mehr abziehen läßt. n ist damit der ganze Teil der Wurzel. Bringt man z nach Register HL, a nach Register DE und n nach Register B, so kann man daraus das Flußbild nach Bild 9.9 bilden.

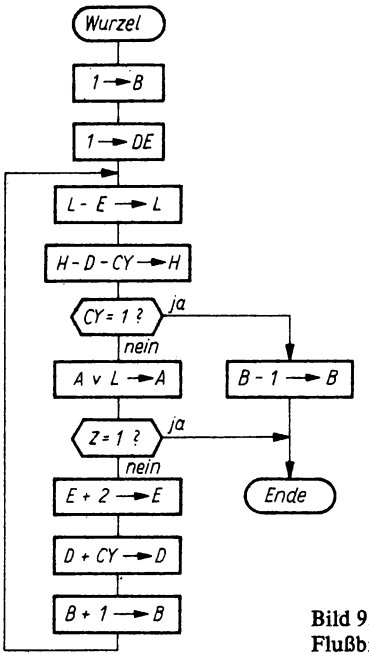


Bild 9.9
Flußbild, Wurzel aus einer ganzen Dezimalzahl

```
        PN   BB
;QUADRATWURZEL AUS EINER GANZEN DEZIMALZAHL MIT 4 BCD-STELLEN
;RADIKAND IN HL
;WURZEL IN B

        LD   B,1
        LD   DE,1
M1:     LD   A,L
        SUB  E
        DAA
        LD   L,A
        LD   A,H
        SBC  D
        DAA
        LD   H,A
        JRC  M3-#
        OR   L
        JRZ  M2-#
        LD   A,E
        ADD  2
        DAA
        LD   E,A
        LD   A,D
        ADC  0
        DAA
        LD   D,A
        LD   A,B
        ADD  1
        DAA
        LD   B,A
        JR   M1-#
M3:     LD   A,B
        SUB  1
        DAA
        LD   B,A
M2:     RET
        END
```

9.3. Anzeige mit LED-Elementen

Lichtemitteranzeigen mit LED dienen dazu, die im Rechner gespeicherten Zahlen anzuzeigen. Dazu werden in den meisten Fällen 7-Segment-Elemente verwendet. Bild 9.10 zeigt den Anschluß von 6 LED-Anzeigen an den Systembus eines Mikrorechners. Der Systembus erhält die Bussignale eines Prozessors (Adreß-, Daten- und Steuerbus). Zu den einzelnen Anzeigen führen Ausgabetore, über die die Information vom Datenbus zur LED-Anzeige gelangt. Die Ausgabe läßt sich im einfachsten Fall über eine Torschaltung realisieren. Besser ist es jedoch, wenn das zur LED-Anzeige erforderliche Bit-Muster in einem Register zwischengespeichert wird. Dazu eignet sich z. B. der Baustein *8212*. Für die Anzeige ist es nicht notwendig, daß die ausgegebenen Ziffern über einen BCD-zu-7-Segment-Decoder umgewandelt werden. Statt dessen werden die zur Ansteuerung eines Anzeigeelements notwendigen Bit-Muster vom Prozessor direkt ausgegeben. Die Umwandlung vom BCD-Code in das notwendige Bit-Muster läßt sich dann durch ein Programm erreichen. Bild 9.11 zeigt, wie den einzelnen Segmenten die Bits eines Bytes zugeordnet werden können. Daraus ergibt sich Tabelle 9.1.

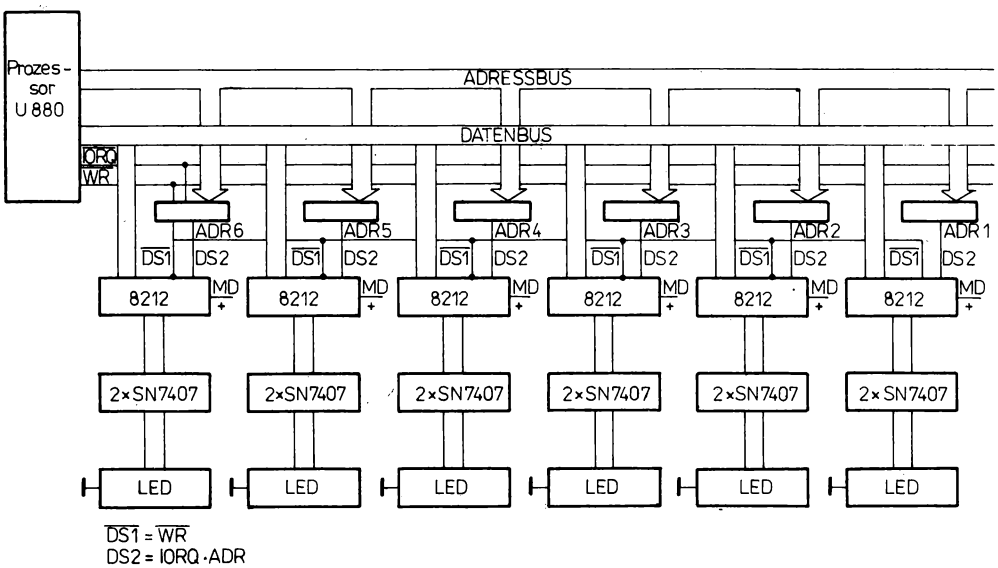


Bild 9.10
Anschluß von 6 LED-Anzeigen an den Systembus eines Mikrorechners

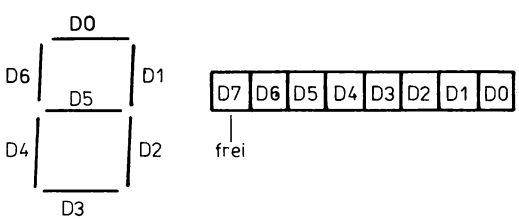


Bild 9.11
Zuordnung der Bit-Stellen eines Bytes zu den Segmenten einer LED-Anzeige

Tabelle 9.1.
Bit-Muster für die Hexadezimalziffern 0 bis F

Zeichen	Bit-Muster (dual)	hexa-dezimal
0	0 1 0 1 1 1 1 1	5F
1	0 0 0 0 0 1 1 0	06
2	0 0 1 1 1 0 1 1	3B
3	0 0 1 0 1 1 1 1	2F
4	0 1 1 0 0 1 1 0	66
5	0 1 1 0 1 1 0 1	6D
6	0 1 1 1 1 1 0 1	7D
7	0 0 0 0 0 1 1 1	07
8	0 1 1 1 1 1 1 1	7F
9	0 1 1 0 1 1 1 1	6F
A	0 1 1 1 0 1 1 1	77
B	0 1 1 1 1 1 0 0	7C
C	0 1 0 1 1 0 0 1	59
D	0 0 1 1 1 1 1 0	3E
E	0 1 1 1 1 0 0 1	79
F	0 1 1 1 0 0 0 1	71

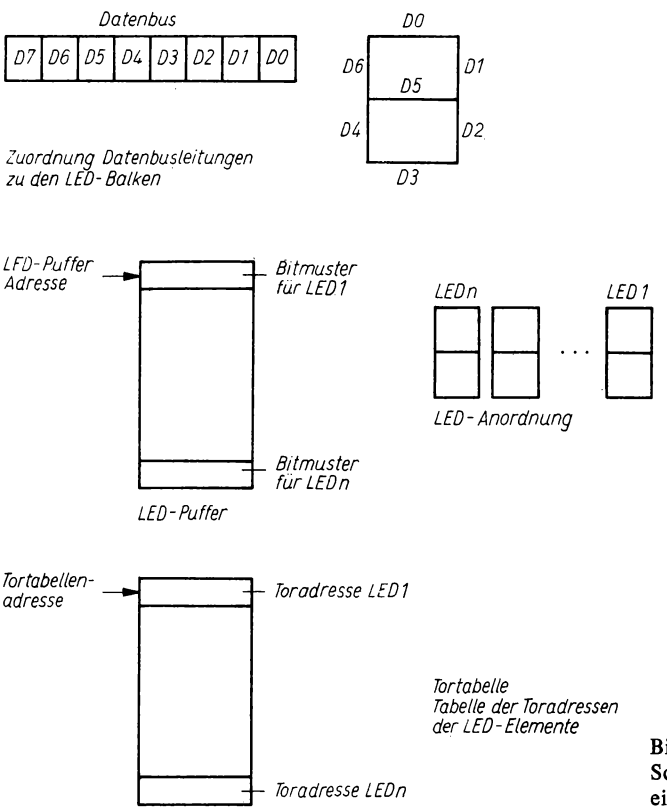
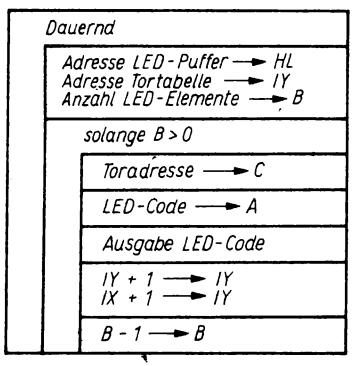


Bild 9.12
Schema zur Programmierung einer LED-Anzeige

Programm zur LED-Anzeige ohne Konvertierung und Zwischenspeicher

Die zur Anzeige kommenden Daten stehen im LED-Puffer. Für jede LED gibt es eine Zelle des LED-Puffers. Jedem Balken einer LED ist ein Bit dieser Zelle (nach Bild 9.12) zugeordnet. Ist dieses Bit »1«, leuchtet der Balken.
Das folgende Programm realisiert die Anzeige des LED-Pufferinhalts auf n-LED-Elementen durch ständige Wiederholung der Datenausgabe (ohne Zwischenpufferung in einem Register).
Dazu sei in HL die LED-Pufferadresse, in IY die Tortabellenadresse und in B die Anzahl der LED-Elemente.
Für das Programm gilt:
a – Struktogramm (im vorletzten Linienrahmen muß es richtig heißen: HL + 1 → HL und IY + 1 → IY)



b – Assembler-Programm

```
        PN   C5
;PROGRAMM LED-ANZEIGE OHNE KONVERTIERUNG UND ZWISCHENSPEICHER
;ADRESSE LED-PUFFER IN HL
;ADRESSE TORTABELLE IN IY
;ANZAHL LED-ELEMENTE IN B

LEDA:   PUSH IY
        PUSH HL
        PUSH BC
ZYK:    LD   C,(IY)
        LD   A,(HL)
        OUT  C
        INC  HL
        INC  IY
        DJNZ ZYK-#
        POP  BC
        POP  HL
        POP  IY
        JR   LEDA-#
        END
```

Um den Inhalt eines Speicherpuffers auf LED-Elementen hexadezimal anzuzeigen, müssen die Daten folgende Schritte durchlaufen (Bild 9.13).

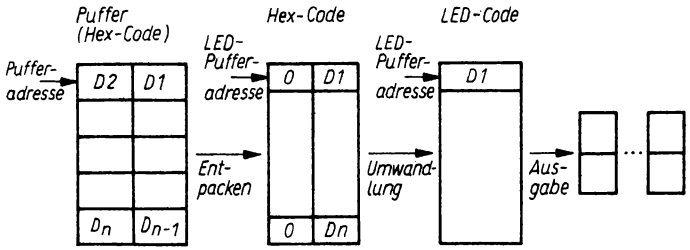
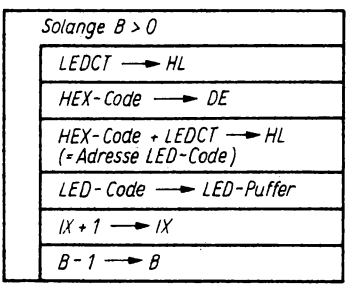


Bild 9.13
Schritte zur hexadezimalen Anzeige eines Puffers auf LED-Elemente

Das Programm zum Entpacken der Daten hat mit
Pufferadresse in HL
LED-Pufferadresse in DE und
Anzahl Pufferzellen in B
folgende Form:
a – Struktogramm



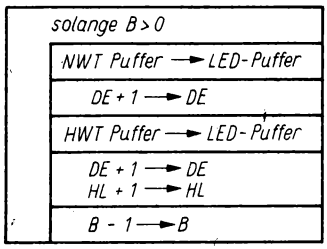
b – Assembler-Programm

```
;ENTPACKEN VON HEX-ZAHLEN
;PUFFERADRESSE IN HL
;LED-PUFFERADRESSE IN DE
;ANZAHL PUFFERZELLEN IN B

ENTP:   XOR   A
ZYK:    RRD                     ;NWT PUFFER --> A
        LD    (DE),A
        INC   DE
        RRD                     ;HWT PUFFER --> A
        LD    (DE),A
        RRD                     ;PUFFERINHALT WIEDERHERSTELLEN
        INC   DE
        INC   HL
        DJNZ  ZYK-#
        RET
```

Die Umwandlung von HEX-Code in den LED-Code kann im LED-Puffer erfolgen. Dazu benötigen wir eine Tabelle, in der der LED-Code für die 16-HEX-Ziffern 0–F der Reihe nach gespeichert ist. Die Anfangsadresse dieser Tabelle habe den Namen LEDCT. Steht die An-

zahl der LED-Elemente in B und die Adresse des LED-Puffers in IX, so gilt:
a – Struktogramm



b – Assembler-Programm

```
        PN   C6
;LED-AUFBEREITUNG FUER HEX-ZAHLEN
;UMWANDLUNG HEX-CODE IN LED-CODE
;ADRESSE DATENPUFFER IN IX
;ANZAHL DER ELEMENTE IN B

UHL:    LD   HL,LEDCT        ;TABELLE LED-CODE
        LD   E,(IX)
        LD   D,0
        ADD  HL,DE
        LD   A,(HL)
        LD   (IX),A
        INC  IX
        DJNZ UHL-#
        RET
LEDCT:  EQU  0               ;ADRESSE FUER LED-CODE-TABELLE.
;                             DIE ADRESSE IST ANZUGEBEN
        END
```

Eine einfache Möglichkeit, eine Dualzahl auf einer LED-Anzeige darzustellen, ergibt sich durch spezielle Auswahl der Balken (in Bild 9.14 stark gezeichnet).

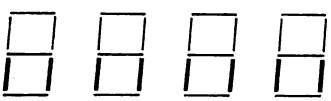
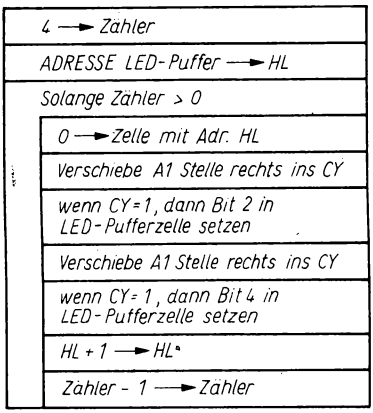


Bild 9.14
Auswahl von Balken einer 4stelligen LED-Anzeige zur Darstellung einer Dualzahl

Soll z. B. der Inhalt von Register A auf dieser LED-Anzeige angezeigt werden, ist der LED-Puffer mit 4 Bit-Mustern so aufzubereiten, daß die Bit-Stellen für die zu leuchtenden Balken »1« gesetzt sind.
Lösung:
a – Struktogramm



b – Assembler-Programm

```
        PN    C7
;LED-AUFBEREITUNG
;BITMUSTER VON REGISTER A WIRD AUF 4 LED-ELEMENTEN ANGEZEIGT
;BITMUSTER IN REGISTER A
;ADRESSE LED-PUFFER IN HL

DUALA:  LD    B,4
ZYK:    LD    (HL),0
        RRA
        JRNC  M1-#
        SET   2,(HL)
M1:     RRA
        JRNC  M2-#
        SET   4,(HL)
M2:     INC   HL
        DJNZ  ZYK-#
        RET
        END
```

Nach Setzen der Bitstellen im LED-Puffer muß zur Anzeige das Programm »LED-Anzeige ohne Konvertierung« aufgerufen werden.

Beispiel 14

Digitaluhr mit Prozessor und einer LED-Anzeige

Zur Realisierung einer Digitaluhr werde die in Bild 9.12 beschriebene LED-Anzeige verwendet. Die Zuordnung der LED-Elemente zur Uhrzeit zeigt die folgende Darstellung:

LED-Element	6	5	4	3	2	1
	Stunden		Minuten		Sekunden	

Das Programm besteht aus folgenden Teilschritten:

1. Schritt – Realisierung des eigentlichen Uhrenprogramms;
2. Schritt – Speichern der Zahlenwerte für Sekunden, Minuten und Stunden in den Datenpuffer für die Anzeige;
3. Schritt – Aufruf des Programms für die LED-Anzeige.

Für den 1. Schritt läßt sich das in Bild 9.15 gezeigte Ablaufdiagramm aufstellen. Im 2. Schritt werden die Zahlenwerte für Sekunden, Minuten und Stunden der Reihe nach aus

den zugehörigen Registern geholt. Nach Trennung der unteren und oberen 4 Bit (BCD-Ziffern) speichert man sie in den Datenpuffer für die LED-Anzeige. Anschließend erfolgt die Umwandlung in den LED-Code und die Ausgabe auf die LED-Elemente.

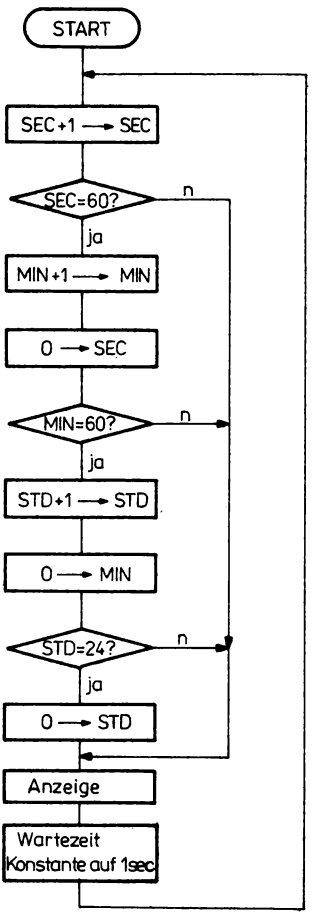


Bild 9.15
Flußdiagramm zur Errechnung von Sekunden, Minuten und Stunden eines Uhrenprogramms

```
        PN   C1
;UHRENPROGRAMM OHNE CTC-SCHALTKREIS
;ZEITZAEHLPROGRAMM

UHR:    LD   A,(U)          ;SEKUNDENZAEHLUNG
        ADD  1
        DAA
        LD   (U),A
        SUB  60H
        JRNZ ANZ-#          ;SPRUNG ZUR ANZEIGE
        LD   (U),A          ;RUECKSETZEN SEKUNDENZAEHLER
        LD   A,(U+1)        ;MINUTENZAEHLER
        ADD  1
        DAA
        LD   (U+1),A
        SUB  60H
        JRNZ ANZ-#          ;SPRUNG ZUR ANZEIGE
```

```
        LD    (U+1),A         ;RUECKSETZEN MINUTENZAEHLER
        LD    A,(U+2)         ;STUNDENZAEHLUNG
        ADD   1
        DAA
        LD    (U+2),A
        SUB   24H
        JRNZ  ANZ-#           ;SPRUNG ZUR ANZEIGE
        LD    (U+1),A         ;RUECKSETZEN STUNDENZAEHLER
ANZ:    CALL  ANZE            ;AUFRUF ANZEIGEPROGRAMM
        JR    UHR-#
;PROGRAMM ZUM ABSPEICHERN VON REGISTER A IN DEN LED-PUFFER
;IM LED-PUFFER STEHEN DIE BEIDEN HEX-ZIFFERN
;IN 2 ZAHLEN RECHTSBUENDIG
;IN HL STEHT DIE LED-PUFFERADRESSE - HL WIRD UM 2 ERHOEHT
ABSP:   LD    B,A
        AND   0FH
        LD    (HL),A          ;UNTERE 4 BIT --> LED-PUFFER
        INC   HL
        RRCA
        RRCA
        RRCA
        RRCA
        AND   0FH
        LD    (HL),A          ;OBERE 4 BIT --> LED-PUFFER
        INC   HL
        RET
;ANZEIGEPROGRAMM
;SPEICHERT DATEN FUER SEKUNDEN, MINUTEN UND STUNDEN IN DEN LED-PUFFER
ANZE:   LD    HL,LP           ;LED-PUFFERADRESSE
        LD    A,(U)
        CALL  ABSP            ;SEK --> LED-PUFFER
        LD    A,(U+1)
        CALL  ABSP            ;MIN --> LED-PUFFER
        LD    A,(U+2)
        CALL  ABSP            ;STD --> LED-PUFFER
        LD    BC,(ZK)         ;ZAEHLKONSTANTE FUER 1 SEKUNDE
ZS:     PUSH  BC              ;ZEITSCHLEIFE FUER 1 SEKUNDE
        CALL  LEDA            ;LED-ANZEIGE
        POP   BC
        DEC   BC              ;ZAEHLUNG FUER 1 SEKUNDE
        LD    A,B
        OR    C
        JRNZ  ZS-#
        RET
U:      DB    0               ;SEKUNDEN-SPEICHERZELLE
        DB    0               ;MINUTEN-SPEICHERZELLE
        DB    0               ;STUNDEN-SPEICHERZELLE
LP:     BER   6               ;6 ZELLEN FUER LED-PUFFER
ZK:     DA    150H            ;ZAEHLKONSTANTE
TOR:    DB    0CH             ;TABELLE DER PERIPHEREN LED-ADRESSEN
        DB    0DH
        DB    0EH
        DB    0FH
        DB    1CH
        DB    1DH
STAB:   DB    5FH             ;SEGMENTTABELLE
        DB    6H
        DB    3BH
        DB    2FH
        DB    66H
        DB    6DH
        DB    7DH
        DB    7H
        DB    7FH
        DB    6FH
        DB    77H
        DB    7CH
        DB    59H
        DB    3EH
        DB    79H
        DB    71H
;AUSGABE AUF LED-ELEMENTE
LEDA:   LD    IY,TOR
        LD    IX,LP
```

```
        LD   B,6
        LD   HL,STAB
ZYK:    LD   C,(IY)          ;ELEMENTADRESSE NACH C
        LD   E,(IX)          ;ZIFFER NACH E
        LD   D,0
        ADD  HL,DE
        LD   A,(HL)          ;SEGMENTCODE NACH A
        OUT  A
        INC  IX
        INC  IY
        DJNZ ZYK-#
        RET
        END
```

9.4. Programme mit peripheren Bausteinen

Beispiel 17

Mit einem CTC-Baustein soll eine Impulsfolge nach Bild 9.16 für eine vorgegebene Frequenz erzeugt werden.

Eine Impulsfolge mit einem Tastverhältnis von 1:1 läßt sich mit dem CTC-Baustein nur dadurch realisieren, daß dem Ausgang ZC/TO ein D-Flip-Flop nachgeschaltet wird (Bild 9.17).

Das ist notwendig, weil der Ausgangsimpuls ZC/TO eine konstante Breite von etwa 0,6 µs hat.

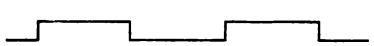


Bild 9.16
Impulsfolge mit einem Tastverhältnis von 1:1

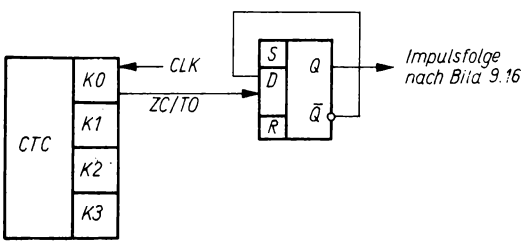


Bild 9.17
CTC-Ausgang mit D-FF

Durch das D-Flip-Flop erfolgt eine Untersetzung 1:2.

Bezeichnet man die Zeitkonstante des CTC-Kanals mit Z und die CTC-Teilerkonstante mit K (16 oder 256), so ist die Gesamtuntersetzung $1:(Z \cdot K \cdot 2)$. Soll z. B. die Frequenz nach Bild 9.16 2,4 kHz betragen und ist die Taktfrequenz 2,4 MHz, muß

$$\frac{2{,}4\,\text{MHz}}{Z \cdot K \cdot 2} = 2{,}4\,\text{kHz} \quad \text{sein,}$$

$$Z \cdot K \cdot 2 = 1000 \quad \text{mit} \quad K = 16$$

$$Z = \frac{1000}{16{,}2} = 31{,}25 \approx 31\,.$$

Das Einstellprogramm für den CTC-Schaltkreis lautet:

```
LD      A, 7     ; Zeitgeber, Teiler 16, Reset
OUT     CTC
LD      A, 31    ; Zeitkonstante 31
OUT     CTC
```

Ein allgemeines Unterprogramm, in dem die Zeitkonstante im Register B und die Adresse des CTC-Kanals im Register C stehen, heißt:

```
CTC:  LD      A, 7  ; Zeitgeber, Teiler 16, Reset
      OUT     A     ; Kanalsteuerwort
      OUT     B     ; Zeitkonstante
      RET
```

Dieses Unterprogramm wird wie folgt aufgerufen:

```
      LD      B, 31
      LD      C, ADR  ; CTC-Kanaladresse
      CALL    CTC
```

Beispiel 18

Digitaluhr mit CTC-Baustein

Der Vorteil dieser Konfiguration ist es, daß das Uhrenprogramm den Rechner nur dann beansprucht, wenn eine Weiterzählung der Zeit (Sekunden) vorgesehen ist. Der Rechner kann damit gleichzeitig für weitere Aufgaben genutzt werden, die durch INTERRUPT angefordert werden müssen.

Bild 9.18 zeigt die Zusammenstellung der Baugruppen zum Aufbau einer Digitaluhr mit

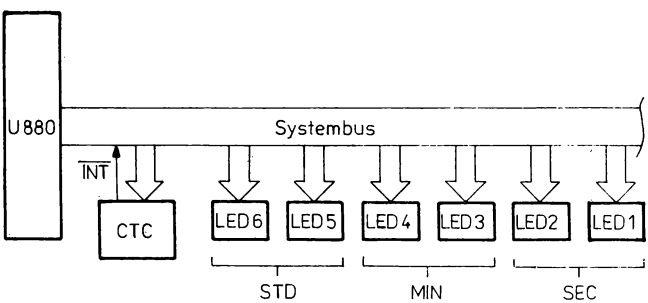


Bild 9.18
Zusammenschaltung der Bausteine *U880* und *U857D* zur Realisierung einer Digitaluhr

CTC-Baustein.

Das Programm besteht aus folgenden Teilabschnitten:

1. Initialisieren des CTC-Bausteins,
2. Uhrenprogramm,
3. Abspeichern der Zahlenwerte für Sekunden, Minuten und Stunden in den Datenpuffer für die Anzeige (LED-Puffer),
4. Aufruf des Programms für die LED-Anzeige.

Der CTC-Baustein zählt die Takte Φ für die Zeiteinheit »1 s«. Dazu setzt man Kanal 0 des Bausteins in die Betriebsart »Zeitgeber«. Wenn man den Vorteiler nutzt, so können mit einem Kanal $256 \times 256 = 65\,536$ Takte gezählt werden. Beträgt die Quarzfrequenz - z. B.

$f = 2{,}4$ MHz, so besteht 1 s aus $2{,}4 \times 10^6$ Takten. Ein Kanal reicht daher für die Zählung einer Sekunde nicht aus. Man zählt deshalb im Kanal 0 die Hundertstelsekunden und über Kanal 1 die Sekunden. Der ZC/TO 0-Impuls wird dabei für die Zählung im Kanal 1 benutzt.
Wird der Vorteiler von Kanal 0 auf $p = 256$ gestellt, so gilt für die Zeitdifferenz der Nulldurchgänge:

$$t_i = t_c \cdot p \cdot TKO = \frac{p \cdot TKO}{f}.$$

Mit $t_i = 0{,}01$ s, $p = 256$, $f = 2{,}457\,6$ MHz $= 2{,}457\,6 \cdot 10^6\ s^{-1}$ gilt:

$$0{,}01\ s = \frac{256 \cdot TKO}{2{,}457\,6 \cdot 10^6 \cdot s^{-1}}.$$

Daraus folgt:

$$TKO = \frac{24\,576}{256} = 96.$$

Im Kanal 0 des CTC-Bausteins wird demzufolge die Zeitkonstante $TKO = 96$ eingestellt. Kanal 1 wird in die Betriebsart »Zähler« gesetzt. Die Zeitkonstante beträgt $TK1 = 100$. Damit durchläuft Kanal 1 jede Sekunde den Zählerstand »0«. Mit diesem Nulldurchgang bildet man eine INTERRUPT-Anforderung an den Prozessor. Für die Initialisierung des CTC-Bausteins werden für das Beispiel folgende Adressen vereinbart:
1. Die Auswahladresse des CTC-Bausteins sei 80H.
2. Der Interruptvektor des CTC-Bausteins sei 90H.
3. Die Taktfrequenz des Rechners sei 2,457 6 MHz.

```
        PN   UHRCTC
;UHRENPROGRAMMM MIT CTC-BAUSTEIN
;ADRESSE CTC0 = 80H, ADRESSE CTC1 = 81H

;INITIALISIERUNG CTC
UCTC:   LD   A,0              ;HOEHERWERTIGER TEIL INTERRUPT-VEKTOR
        LD   I,A
        LD   A,90H            ;NIEDERWERTIGER TEIL INTERRUPT-VEKTOR
        OUT  80H
        LD   IX,ABED          ;ADRESSE BEDIENPROGRAMM CTC
        LD   (92H),IX         ;INTERRUPT FUER KANAL 1
        LD   A,37H            ;KANALSTEUERWORT FUER KANAL 0,
                               BETRIEBSART ZEITGEBER, INTERRUPT-VERBOT,
                               VORTEILER 256, ZEITKONSTANTE LADEN
        OUT  80H
        LD   A,(TK0)          ;ZEITKONSTANTE FUER KANAL 0
        OUT  80H
        LD   A,0C7H           ;KANALSTEUERWORT FUER KANAL 1,
                               BETRIEBSART ZAEHLER, INTERRUPT ERLAUBT,
                               ZEITKONSTANTE LADEN
        OUT  81H
        LD   A,(TK1)          ;ZEITKONSTANTE FUER KANAL 1

        OUT  81H
        IM   2                ;INTERRUPT-MODE 2
        EI                    ;INTERRUPT-FREIGABE
;ANZEIGEPROGRAMM
ANZ:    LD   HL,LEDB          ;ADRESSE LED-PUFFER NACH HL
        LD   A,(UHR)          ;SEKUNDEN NACH LED-PUFFER
        CALL ABSP
        LD   A,(UHR+1)        ;MINUTEN NACH LED-PUFFER
```

```
        CALL ABSP
        LD   A,(UHR+2)        ;STUNDEN NACH LED-PUFFER
        CALL ABSP
        CALL LED              ,SEC, MIN UND STD IN ANZEIGEREGISTER
        JR   ANZ-#
;INTERRUPT-BEDIENPROGRAMM (ZEITZAEHLPROGRAMM)
ABED:   PUSH HL
        PUSH AF
SEC:    LD   HL,UHR           ;SEKUNDENZAEHLERADRESSE NACH HL
        LD   A,(HL)
        ADD  1                ;SEC + 1 --> SEC
        DAA
        LD   (HL),A
        CMP  60H              ;SEC = 60?
        JRZ MIN-#
        JR   EN-#
MIN:    LD   (HL),0           0 NACH SEC
        INC  HL               ;MINUTENZAEHLERADRESSE NACH HL
        LD   A,(HL)
        ADD  1                ;MIN + 1 --> MIN
        DAA
        LD   (HL),A
        CMP  60H              ;MIN = 60?
        JRZ  STD-#
        JR   EN-#
STD:    LD   (HL),0           ;0 NACH MIN
        INC  HL               ;STD-ZAEHLER NACH HL
        LD   A,(HL)
        ADD  1
        DAA                   ;STD + 1 --> STD
        LD   (HL),A
        CMP  24H              ;STD = 24?
        JRZ  RES-#
        JR   EN-#
RES:    LD   (HL),0           ;0 NACH STD
EN:     POP  AF
        POP  HL
        EI                    ;INTERRUPT-FREIGABE
        RETI
;ABSPEICHERPROGRAMM FUER 2 BCD-ZIFFERN AUS REGISTER A
;IN EINEN LED-PUFFER, DESSEN ADRESSE IN HL STEHT
ABSP:   PUSH AF
        AND  0FH
        LD   (HL),A           ;UNTERE 4 BIT NACH LED-PUFFER
        INC  HL
        POP  AF
        RRCA
        RRCA
        RRCA
        RRCA
        AND  0FH
        LD   (HL),A           ;OBERE 4 BIT NACH LED-PUFFER
        INC  HL
        RET
;AUSGABEPROGRAMM DER ZIFFERN VOM LED-PUFFER IN DIE LED-ELEMENTE
LED:    LD   B,6              ;ZAEHLER FUER 6 ZIFFERN
        LD   IX,LEDB          ;ADRESSE LED-PUFFER NACH IX
        LD   IY,ANR           ;ADRESSE ANZEIGEREGISTERTABELLE NACH IY
AUS:    LD   C,(IY)           ;ADRESSE ANZEIGEREGISTER NACH C
        LD   HL,SEGM          ;ADRESSE DER SEGMENTTABELLE NACH HL
        LD   E,(IX)           ;ZIFFER NACH DE
        LD   D,0
        ADD   HL,DE           ;IN HL STEHT ADRESSE DES SEGMENTCODES
                               DER ZIFFER
        LD   A,(HL)
        OUT  A                ;ZIFFER AUSGEBEN
        INC  IX
        INC  IY
        DJNZ AUS-#            ;NAECHSTE ZIFFER
        RET
LEDB:   BER  6                ;6 ZELLEN FUER LED-PUFFER
ANR:    DB   0CH              ;TABELLEN DER ADRESSEN DER LED-ELEMENTE
        DB   0DH
        DB   0EH
```

```
        DB    0FH
        DB    1CH
        DB    1DH
SEGM:   DB    05FH          ;SEGMENTTABELLE
        DB    6H
        DB    3BH
        DB    2FH
        DB    66H
        DB    6DH
        DB    7DH
        DB    7H
        DB    7FH
        DB    6FH
UHR:    BER   3             ;SPEICHER FUER SEC-, MIN-, STD-WERTE
TK0:    DB    60H           ;ZEITKONSTANTE FUER CTC-KANAL 0
TK1:    DB    64H           ;ZEITKONSTANTE FUER CTC-KANAL 1
        END
```

Programm zur Ausgabe eines Zeichens über PIO-Schaltkreis-Realisierung eines SIF-1000-Anschlußspiegels

Der Interfacespiegel SIF 1000 zur Datenausgabe besteht aus folgenden Leitungen:

- 8 Leitungen zur Datenausgabe DATA 1 bis DATA 8
- 3 Kommandoleitungen KOMA 1 bis KOMA 3
- 3 Statusleitungen STATA 1 bis STATA 3
- der RUFA-Leitung
- der ENDA-Leitung

Zunächst werden Daten und Kommando ausgegeben. Sind Daten und Kommando auf den Leitungen stabil, wird dies durch RUFA den peripheren Geräten mitgeteilt. Nachdem die Elektronik des peripheren Gerätes das Datenzeichen und das Kommando abgenommen hat, gibt es den STATUS STATA und etwas später das Signal ENDA. ENDA sagt dem Rechner, daß der STATUS vorhanden ist und Daten, Kommando und RUFA abgeschaltet werden können. Nachdem der Rechner den STATUS übernommen hat, schaltet er RUFA ab. Danach schaltet die Elektronik des peripheren Gerätes ENDA und STATA ab (Bild 9.19).

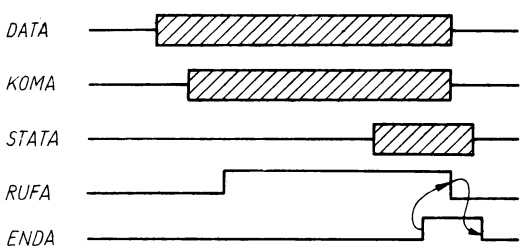


Bild 9.19
Taktdiagramm zur Datenausgabe über SIF-1000-Spiegel

Zum Anschluß eines Ausgabegerätes über SIF-1000-Interface-Spiegel benötigen wir 16 Leitungen. Diese 16 Leitungen lassen sich mit einem PIO-Schaltkreis realisieren. Kanal A nehmen wir für die Daten und Kanal B für die restlichen Signale. Bild 9.20 zeigt das Anschlußbild.

Die programmtechnische Realisierung der Ausgabe eines Zeichens läßt sich entweder nach der Betriebsart »Polling« oder »Interrupt« realisieren.

Bei der Betriebsart Polling fragt das Programm nach RUFA ständig das Signal ENDA ab.

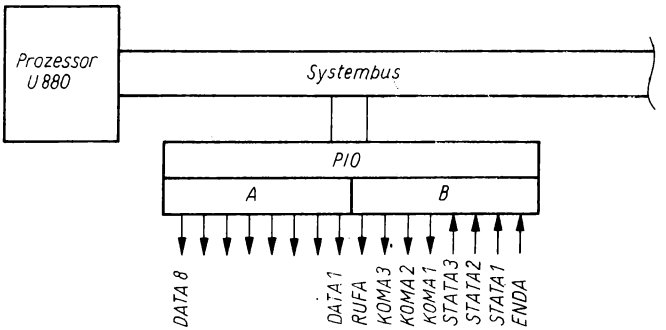


Bild 9.20
Realisierung der SIF-1 000-Signale mit PIO-Schaltkreis

Bei der Betriebsart INTERRUPT erzeugt das Signal ENDA einen INTERRUPT. Der Rechner kann zwischen Ausgabe eines Zeichens und dem folgenden INTERRUPT durch ENDA eine 2. Aufgabe (sogenannte Hintergrundaufgabe) bearbeiten.
Der PIO-Schaltkreis muß vor der eigentlichen Ausgabe auf die Betriebsart eingestellt (initialisiert) werden.

Beispiel 19
Ausgabe eines Zeichens in Polling-MODE aus dem Register A mit SIF-1000-Interface über PIO nach Bild 9.20. Das Programm besteht aus 2 Teilen, dem Initialisierungsprogramm und dem Ausgabeprogramm. Für die Ausgabe wird Kanal A und B in Bit-MODE initialisiert. Das Ausgabeprogramm für ein Zeichen läßt sich wiederum zur Ausgabe eines Datensatzes verwenden.
Initialisierung

```
    .  PN   C2
;PIO-ADRESSEN
;DIE ADRESSEN SIND ENTSPRECHEND DER HARDWARE EINZUSETZEN

SPIOA: EQU  0               ;STEUERWORT PIO KANAL A
SPIOB: EQU  0               ;STEUERWORT PIO KANAL B
;INITIALISIERUNGSPROGRAMM
INIT:  LD   HL,STAB         ;ADRESSE STEUERWORTTABELLE
       LD   C,SPIOA         ;ADRESSE STEUERWORT PIO KANAL A
       LD   B,3             ;ANZAHL DER STEUERWORTE
       OTIR
       LD   C,SPIOB         ;ADRESSE STEUERWORT PIO KANAL B
       LD   B,3             ;ANZAHL DER STEUERWORTE
       OTIR
       RET
;INITIALISIERUNGSDATEN
;KANAL A
STAB:  DB   OFFH            ;BETRIEBSART BITMODE
       DB   0               ;BIT 0 - 7 AUSGABE
       DB   7H              ;INTERRUPT-VERBOT
;KANAL B
       DB   OFFH            ;BETRIEBSART BITMODE
       DB   OFH             ;BIT 0 - 3 EINGABE, BIT       AUSGABE
       DB   7H              ;INTERRUPT-VERBOT
       END
```

Ausgabe eines Zeichens im Polling-MODE

```
        PN   C3
;PROGRAMM ZUR AUSGABE EINES ZEICHENS AUS DEM REGISTER A
;PIO-ADRESSE TOR A IN REGISTER C
;PIO-ADRESSE TOR B IN REGISTER D
;SIF-1000-KOMMANDO IN REGISTER E
;SIF-1000-STATUS WIRD INS REGISTER B EINGELESEN
;DATENAUSGABE UEBER TOR A
;KOM, STAT, RUF UND END UEBER TOR B

AUS:    OUT  A              ;DATENBYTE AUS
        LD   C,D
        LD   A,E            ;KOM UEBER TOR B AUS
        OUT  A
        OR   80H            ;RUF EINSCHALTEN
        OUT  A
WS1:    IN   A              ;WARTEN AUF H-PEGEL
        BIT  0,A            ;VON END
        JRY  WS1-#
        AND  0EH            ;STAT AUSBLENDEN
        LD   B,A            ;UND INS REGISTER B
        XOR  A              ;ABSCHALTEN RUF UND KOM
        OUT  A
WS2:    IN   A              ;WARTEN AUF L-PEGEL
        BIT  0,A            ;VON END
        JRNZ WS2-#
        RET
        END
```

Ausgabe eines Datensatzes aus dem Speicher

```
        PN   C4
;AUSGABE EINES DATENSATZES AUS DEM ARBEITSSPEICHER
;AN SIF-1000-GERAET UEBER PIO
;ADRESSE AUSGABEPUFFER IN HL
;VEREINBARTES STEUERZEICHEN FUER TEXT- ODER
;UEBERTRAGUNGSENDE IM REGISTER B
;PIO-ADRESSE TOR A IN REGISTER C
;PIO-ADRESSE TOR B IN REGISTER D
;SIF-1000-KOMMANDO STELLENRICHTIG IN REGISTER E
;STATUS WIRD NICHT AUSGEWERTET

AUS:    EQU  0              ;ADRESSE FUER UP AUS IST SPAETER EINZUSETZEN
ASA:    LD   A,(HL)         ;ZEICHEN NACH A
        CMP  B              ;ENDE ERREICHT?
        RZ                  ;RET, WENN JA
        PUSH HL
        PUSH BC
        CALL AUS            ;ZEICHEN AUSGEBEN
        POP  BC
        POP  HL
        INC  HL             ;NAECHSTE ADRESSE
        JR   ASA-#
        END
```

10. Arbeitstabellen

10.1. Befehlsübersicht *U880*

Abkürzungen
n 8-Bit-Zahl,
nn 16-Bit-Zahl,
N Register A, B, C, D, E, H, L,
M Speicherzelle ADR = ⟨HL⟩, ⟨IX + d⟩, ⟨IY + d⟩.

Befehlsgruppen (Übersicht)

0	Adreßoperationen (steht in Verbindung mit Befehlen, die zum Speicher zugreifen)	
1	Transportoperationen	1 Wort
		Doppelwort
		Blocktransfer
2	Rechen- und logische Operationen mit 1 Operanden	
	Akkumulatorbefehle	logische Operationen
	1-Wort-Befehle	Verschiebeoperationen
	Doppelwortbefehle	Bit-Operation
		Rechenoperationen
3	Rechen- und logische Operationen mit 2 Operanden	1-Wort-Befehle
		Doppelwortbefehle
4	Rechen- und logische Operationen mit Feldern	
5	Sprungbefehle	
6	Unterprogramm-Befehle	Aufruf des Unterprogramms
		Rückkehr zum Hauptprogramm
7	Ein-/Ausgabebefehle	1-Wort-Befehle
		Blocktransfer
8	Steuerbefehle	

Tabelle der *U880*-Befehle

Einzelworttransfer

LD	r, s	s → r	Bei allen Transportbefehlen außer LD A, I und LD A, R							
LD	d, r	r → d		C	Z	P/V	S	N	H	
s = n, N, M[12]		r = N								
r = n, N		d = N, M								
LD	A, s	s → A								

[12] N = Register A, B, C, D, E, H, L.
M = Speicherzelle ADR = ⟨HL⟩, ⟨IX + d⟩, ⟨IY + D⟩.

LD	d, A	A → d	Bei LD A, I und LD A, R							
s = ⟨BC⟩, ⟨DE⟩, ⟨nn⟩, I, R				C	Z	P/V	S	N	H	
d = ⟨BC⟩, ⟨DE⟩, ⟨nn⟩, I, R					↕	IFF	↕	0	0	

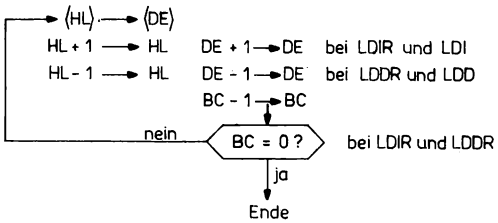
Doppelworttransfer

				C	Z	P/V	N	H	
LD	dd, nn	nn → dd	} dd = BC, DE, HL, SP, IX, IY						
LD	dd, ⟨nn⟩	⟨nn⟩ → dd							
LD	⟨nn⟩, ss	ss → ⟨nn⟩,	ss = BC, DE, HL, SP, IX, IY						
LD	SP, ss	ss → SP	ss = HL, IX, IY						
PUSH	ss	ss → STACK	} ss = BC, DE, HL, AF, IX, IY						
POP	ss	STACK → ss							

Registeraustausch

			C	Z	P/V	N	H	
EX	DE, HL	DE ↔ HL						
EX	AF, AF′	AF ↔ A′F′						
EXX		BC ↔ B′C′ DE ↔ D′E′ HL ↔ H′L′						
EX	⟨SP⟩, ss	ss ↔ ⟨SP + 1⟩, ⟨SP⟩ ss = HL, IX, IY						

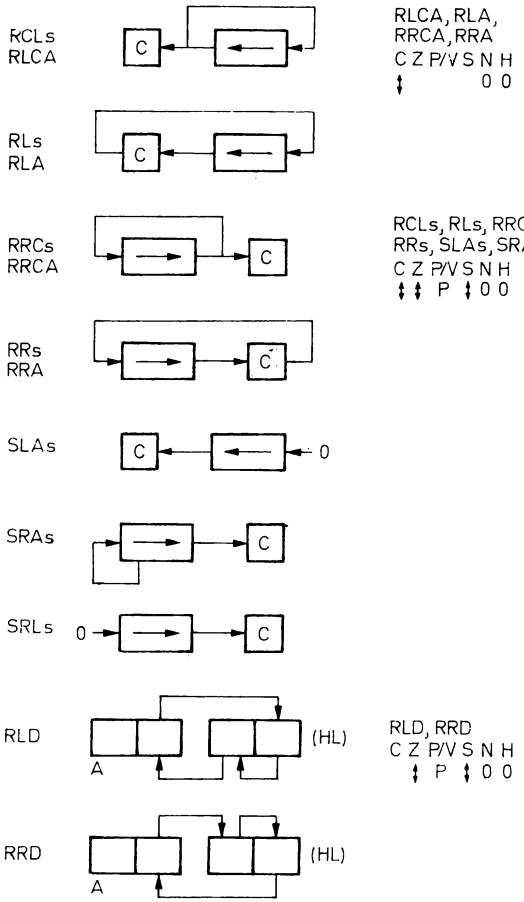
Blocktransfer



	C	Z	P/V	S	N	H
LDIR		X	0	X	0	0
LDDR		X	0	X	0	0
LDI		X	↕	X	0	0
LDD		X	↕	X	0	0

Rechenoperationen und logische Operationen mit 1 Operanden

				C	Z	P/V	S	N	H
CPL		$\overline{A}$	→ A	·	·		·	1	1
NEG		$\overline{A}$	+ 1 → A	↕	↕	∨	↕	1	↕
CCF		$\overline{C}$	→ C	↕	·		·	0	X
SCF		1	→ C	1	·	·	·	0	0
DAA		BCD⟨A⟩	→ A	↕	↕	P	↕	·	↕
DEC	d	d	− 1 → d		↕	∨	↕	1	↕
INC	d	d	+ 1 → d	} d = N, M	↕	∨	↕	0	↕
BIT	b, s	$\overline{s_b}$	→ Z		↕	X	X	0	1
SET	b, s	1	→ s_b	} s = N, M					
RES	b, s	0	→ s_b						
INC	dd	dd	+ 1 → dd	} dd = BC, DE, HL, SP, IX, IY					
DEC	dd	dd	− 1 → dd						



Rechenoperationen mit 2 Operanden

8-Bit-Operationen

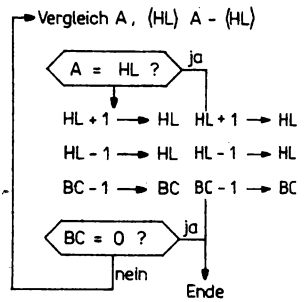
				C	Z	P/V	S	N	H
ADD	s	A + s → A	s = N, n, M[12]	↕	↕	∨	↕	0	↕
ADC	s	A + s + C → A		↕	↕	∨	↕	0	↕
SUB	s	A − s → A		↕	↕	∨	↕	1	↕
SBC	s	A − s − C → A		↕	↕	∨	↕	1	↕
AND	s	A ∧ s → A		0	↕	P	↕	0	1
OR	s	A ∨ s → A		0	0	P	↕	0	1
XOR	s	A⊕s → A		0	0	P	↕	0	1
CP	s	Vergleich A, s	A − s stellt Flags	↕	↕	∨	↕	1	↕

16-Bit-Operationen

				C	Z	P/V	S	N	H
ADD	HL, ss	HL + ss → HL	ss = BC, DE, HL, SP	↕	·		·	0	X
ADC	HL, ss	HL + ss + C → HL		↕	↕	v	↕	0	↕
SBC	HL, ss	HL − ss − C → HL		↕	↕	v	↕	1	↕
ADD	IX, ss	IX + ss → IX	ss = BC, DE, IX, SP	↕			·		↕
ADD	IY, ss	IY + ss → IY	ss = BC, DE, IY, SP	↕					↕

Rechenoperationen auf Feldern
Suchoperationen

CPIR
CPDR
CPI
CPD



A − ⟨HL⟩ stellt Flags

bei CPIR und CPI

bei CPDR und CPD

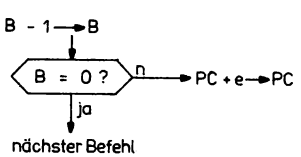
bei CPIR und CPDR

	C	Z	P/V	S	N	H
	0	↕	↕	X	1	X
	0	↕	↕	X	1	X
	0	↕	↕	X	1	X
	0	↕	↕	X	1	X

Sprungbefehle

			C	Z	P/V	S	N	H
JP	nn	nn → PC						
JP	cc, nn	nn → PC, wenn cc erfüllt cc = NZ, Z, NC, C, PO, PE, P, M						
JR	e	PC + e → PC						
JR	kk, e	PC + e → PC, wenn kk erfüllt kk = NZ, Z, NC, C						
JP	⟨ss⟩	ss → PC ss = HL, IX, IY						

DJNZ e



Unterprogrammbefehle

			C	Z	P/V	S	N	H
CALL	nn	PC → STACK nn → PC						
CALL	cc, nn	CALL nn, wenn erfüllt cc = NZ, Z, NC, C, PO, PE, P, M						
RST	z	CALL z mit z = OH, 8H, 10H, 18H, ..., 38H						
RET		STACK → PC						
RET	cc	STACK → PC, wenn cc erfüllt						
RETN		Rücksprung aus nichtmaskiertem INTERRUPT						
RETI		Rücksprung aus maskiertem INTERRUPT						

Ein-/Ausgabebefehle

		Adreßbus $A_{15} \dots A_8$	Adreßbus $A_7 \dots A_0$
IN	A, ⟨n⟩ ⟨n⟩ → A		
IN	r, ⟨C⟩ ⟨C⟩ → r r = N[13]	A	n
OUT	⟨n⟩, A A → ⟨n⟩	B	C
OUT	⟨C⟩, r r → ⟨C⟩ r = N[13]	A	n
		B	C

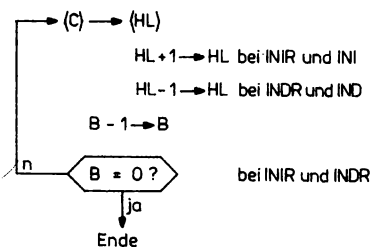
bei Adresse n

C	Z	P/V	S	N	H

bei Adresse C

C	Z	P/V	S	N	H
		P		0	0
		P		0	0
		P		0	0
		P		0	0

Blockeingabe



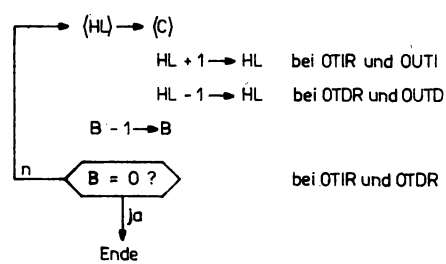
Adreßbus

$A_{15} \dots A_8$	$A_7 \dots A_0$
B	C

INI, IND, OUTI, OUTD

C	Z	P/V	S	N	H
	↕	X	X	1	X

Blockausgabe



INIR, INDR, OTIR, OTDR

C	Z	P/V	S	N	H
	1	X	X	1	X

Steuerbefehle

			C	Z	P/V	S	N	H
NOP	keine Operation							
HALT	Haltbefehl							
DI	0 → IFF	INTERRUPT gesperrt						
EI	1 → IFF	INTERRUPT frei						
IMO	MODE 0	Befehl vom Bus						
IM1	MODE 1	CALL 38H						
IM2	MODE 2	CALL ⟨I, IV⟩						

[13] N ist Register A, B, C, D, E, H, L.

10.2. Befehlssatzerweiterung *U880*

8-Bit-Ladebefehle

s / d	A	B	C	D	E	H	L	IXH	IYL	IYH	IYL	Direkt-op. n
A	7F	78	79	7A	7B	7C	7D	DD7C	DD7D	FD7C	FD7D	3E
B	47	40	41	42	43	44	45	DD44	DD45	FD44	FD45	06
C	4F	48	49	4A	4B	4C	4D	DD4C	DD4D	FD4C	FD4D	0E
D	57	50	51	52	53	54	55	DD54	DD55	FD54	FD55	16
E	5F	58	59	5A	5B	5C	5D	DD5C	DD5D	FD5D	FD5D	1E
H	67	60	61	62	63	64	65	-		-	-	26
L	6F	68	69	6A	6B	6C	6D	-	-	-	-	2E
IXH	DD67	DD60	DD61	DD62	DD63	-	-	DD64	DD65	-	-	DD26
IXL	DD6F	DD68	DD69	DD6A	DD6B	-	-	DD6C	DD6D	-		DD2E
IYH	FD67	FD60	FD61	FD62	FD63	-	-	-	-	FD64	FD65	FD26
IYL	FD6F	FD68	FD69	FD6A	FD6B	-	-	-	-	FD6C	FD6D	FD2E

Arithmetikbefehle

	A	B	C	D	E	H	L	IXH	IXL	IYH	IYL
ADD	87	80	81	82	83	84	85	DD84	DD85	FD84	FD85
ADC	8F	88	89	8A	8B	8C	8D	DD8C	DD8D	FD8C	FD8D
SUB	97	90	91	92	93	94	95	DD94	DD95	FD94	FD95
SBC	9F	98	99	9A	9B	9C	9D	DD9C	DD9D	FD9C	FD9D
AND	A7	A0	A1	A2	A3	A4	A5	DDA4	DDA5	FDA4	FDA5
XOR	AF	A8	A9	AA	AB	AC	AD	DDAC	DDAD	FDAC	FDAD
OR	B7	B0	B1	B2	B3	B4	B5	DDB4	DDB5	FDB4	FDB5
CP	BF	B8	B9	BA	BB	BC	BD	DDBC	DDBD	FDBC	FDBD
INC	3C	04	0C	14	1C	24	2C	DD24	DD2C	FD24	FD2C
DEC	3D	05	0D	15	1D	25	2D	DD25	DD2D	FD25	FD2D

SLI-Befehle

Wie SLA-Befehle, nur wird eine 1 statt einer 0 nachgeschoben.

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
									DD	FD
SLA	CB	CB	CB	CB	CB	CB	CB	CB	CB	CB
	27	20	21	22	23	24	25	26	d	d
									26	26
									DD	FD
SLI	CB	CB	CB	CB	CB	CB	CB	CB	CB	CB
	37	30	31	32	33	34	35	36	d	d
									36	36

Ausführungszeiten

Befehl	Bytezahl	Zyklenzahl	Takte
SLI	wie SLA	wie SLA	wie SLA
LD r_i,n	3	3	11
LD r_i,r	2	2	8
LD r,r_i	2	2	8
LD r_i,r_i	2	2	8
ADD A,r_i	2	2	8
•			
•			
•			
DEC r_i	2	2	8

r = Register A, B, C, D, E

r_i = Register IXH, IXL, IYH, IYL

n = 1-Byte-Konstante

10.3. Flagbelegungstabelle *U880*

Befehle bei denen die Flags nicht geändert werden:

LD	r, s	LD	(nn), A	EX	DE, HL	JRC	e	RETN	
LD	r, n	LD	I, A	EXAF		JRNC	e	RST	Z
LD	M, r	LD	R, A	EXX		JRZ	e	IN	n
LD	(IX + e), r	LD	dd, nn	EX	(SP), HL	JRNZ	e	OUT	n
LD	(IY + e), r	LD	dd, (nn)	EX	(SP), IX	JMP	(HL)	OUT	r
LD	M, n	LD	(nn), dd	EX	(SP), IY	JMP	(IX)	NOP	

LD	(IX + e), n	LD	SP, HL	INC	dd	JMP	(IY)	HALT
LD	(IY + e), n	LD	SP, IX	DEC	dd	DJNZ	e	DI
LD	A, (BC)	LD	SP, IY	SET	b, s	CALL	nn	EI
LD	A, (DE)	PUSH	dd	RES	b, s	CAcc	nn	IM0
LD	A, (nn)	PUSH	AF	JMP	nn	RET		IM1
LD	(BC), A	POP	dd	JPcc	nn	Rkk		IM2
LD	(DE), A	POP	AF	JR	e	RETI		

Befehle mit Flagänderung

LD	A, I S	1, falls Inhalt von I bzw. R negativ, sonst 0
LD	A, R Z	1, falls Inhalt von I bzw. R gleich Null, sonst 0
	H	0
	P/V	Enthält den Inhalt von IFF2
	N	0
	C	Nicht beeinflußt
LDI	S	Nicht beeinflußt
LDD	Z	Nicht beeinflußt
	H	0
	P/V	1, falls BC − 1 ≠ 0, sonst 0
	N	0
	C	Nicht beeinflußt
LDIR	S	Nicht beeinflußt
	Z	Nicht beeinflußt
	H	0
	P/V	0
	N	0
	C	Nicht beeinflußt
CPI	S	1 bei negativem Ergebnis, sonst 0
CPIR	Z	1 bei A = (HL), sonst 0
CPD	H	1 bei negativem Übertrag von Bit 4, sonst 0
CPDR	P/V	1 bei BC − 1 = 0, sonst 0
	N	1
	C	Nicht beeinflußt
ADD s	S	1 bei negativem Ergebnis, sonst 0
ADC s	Z	1 bei Ergebnis Null, sonst 0
ADD n	H	1 bei Übertrag von Bit 3, sonst 0
ADC n	P/V	1 bei Überlauf, sonst 0
	N	0
	C	1 wenn Übertrag von Bit 7, sonst 0
SUB s	S	1 bei negativem Ergebnis, sonst 0
SBC s	Z	1 bei Ergebnis gleich Null, sonst 0
SUB n	H	1 bei negativem Übertrag von Bit 4, sonst 0
SBC n	P/V	1 bei Überlauf, sonst 0
	N	1
	C	1 bei negativem Übertrag, sonst 0

AND	s	S	1 bei negativem Ergebnis, sonst 0
XOR	s	Z	1 bei Ergebnis Null, sonst 0
AND	n	H	1
XOR	n	P/V	1 bei gerader Parität, sonst 0
		N	0
		C	0
OR	s	S	1 bei negativem Ergebnis, sonst 0
OR	n	Z	1 bei Ergebnis Null, sonst 0
		H	1
		PV	1 bei gerader Parität, sonst 0
		N	1
		C	1
CMP	s	S	1 bei negativem Ergebnis, sonst 0
CMP	n	Z	1 bei Ergebnis Null, sonst 0
		H	1 bei negativem Übertrag von Bit 4, sonst 0
		P/V	1 bei Überlauf, sonst 0
		N	1
		C	1 wenn negativer Übertrag, sonst 0
INC	s	S	1 bei negativem Ergebnis, sonst 0
		Z	1 bei Ergebnis gleich Null, sonst 0
		H	1 bei Übertrag von Bit 3, sonst 0
		P/V	1, falls s = 7FH vor Befehlsausführung war, sonst 0
		N	0
		C	Nicht beeinflußt
DEC	s	S	1 bei negativem Ergebnis, sonst 0
		Z	1 bei Ergebnis gleich Null, sonst 0
		H	1 bei negativem Übertrag von Bit 4, sonst 0
		P/V	1 falls Operand zuvor gleich 80H war, sonst 0
		N	1
		C	Nicht beeinflußt
CPL		S	Nicht beeinflußt
		Z	Nicht beeinflußt
		H	1
		P/V	Nicht beeinflußt
		N	1
		C	Nicht beeinflußt
NEG		S	1 bei negativem Ergebnis, sonst 0
		Z	1 bei Ergebnis gleich Null, sonst 0
		H	1 bei negativem Übertrag von Bit 4, sonst 0
		P/V	1, wenn Inhalt des Akkumulators vor dem Befehl = 80H war, sonst 0
		N	1
		C	1, falls Akkumulator vor dem Befehl, sonst 0
DAA		S	1, falls höchstwertiges Bit von A nach Befehl = 1 ist, sonst 0

Z	1 bei Akkumulator = 0 nach Befehl, sonst 0
H	Siehe folgende Tabelle
P/V	1 bei geradzahligem Ergebnis, sonst 0
N	Nicht beeinflußt
C	Siehe folgende Tabelle

C-Flagstellung bei DAA

Operation	CY vor DAA	HEX-Wert im höherwertigen Halbbyte (Bit 7–4)	H vor DAA	HEX-Wert im niederwertigen Halbbyte (Bit 3–0)	Zum BYTE addierte Zahl	CY nach DAA
	0	0–9	0	0–9	00	0
	0	0–8	0	A–F	06	0
	0	0–9	1	0–3	06	0
ADD	0	A–F	0	0–9	60	1
ADC	0	9–F	0	A–F	66	1
INC	0	A–F	1	0–3	66	1
	1	0–2	0	0–9	60	1
	1	0–2	0	A–F	66	1
	1	0–3	1	0–3	66	1
SUB	0	0–9	0	0–9	00	0
SBC	0	0–8	1	6–F	FA	0
DEC	1	7–F	0	0–9	A0	1
NEG	1	6–F	1	6–F	9A	1

ADD	HL, ss	S	–
ADD	IX, pp	Z	–
ADD	IY, rr	H	1, wenn Übertrag aus Bit 11, sonst 0
		P/V	–
		N	0
		C	1 bei Übertrag aus Bit 15, sonst 0
ADC	HL, ss	S	1, wenn Ergebnis negativ, sonst 0
		Z	1, wenn Ergebnis Null, sonst 0
		H	1 bei Übertrag aus Bit 11, sonst 0
		P/V	1 bei Überlauf, sonst 0
		N	0
		C	1 bei Übertrag aus Bit 15, sonst 0
SBC	HL, ss	S	1, wenn Ergebnis negativ, sonst 0
		Z	1, wenn Ergebnis Null, sonst 0
		H	1, wenn Übertrag von Bit 12, sonst 0
		P/V	1 bei Überlauf, sonst 0
		N	1

	C	1 bei Übertrag, sonst 0
RLCA	S	–
RLA	Z	–
	H	0
	P/V	–
	N	0
	C	Inhalt des Bit 7 von A
RRCA	S	–
RRA	Z	–
	H	0
	P/V	–
	N	0
	C	Inhalt von Bit 0 von A
RLC s	S	1, wenn Ergebnis negativ, sonst 0
RL s	Z	1, wenn Ergebnis Null, sonst 0
	H	0
	P/V	1 bei gerader Parität, sonst 0
	N	0
	C	Inhalt des Bit 7
RRC s	S	1, wenn Ergebnis negativ, sonst 0
RR s	Z	1, wenn Ergebnis Null, sonst 0
SRA s	H	0
SRL s	P/V	1 bei gerader Parität, sonst 0
	N	0
	C	Inhalt von Bit 0
RLD	S	1, wenn Ergebnis negativ, sonst 0
RRD	Z	1, wenn Ergebnis Null, sonst 0
	H	0
	P/V	1 bei gerader Parität, sonst 0
	N	0
	C	–
BIT b, s	S	Undefiniert
	Z	1, wenn Bit b = 0, sonst 0
	H	1
	P/V	Undefiniert
	N	0
	C	–
CCF	S	–
	Z	–
	H	Bisheriger Inhalt des Übertrags-(Carry)bits wird kopiert
	P/V	–
	N	0
	C	1, falls CY vorher Null war, sonst 0

SCF	S	–
	Z	–
	H	0
	P/V	–
	N	0
	C	1
IN r	S	1, wenn Eingabe negativ, sonst 0
	Z	1, wenn Eingabe Null, sonst 0
	H	0
	P/V	1 bei gerader Parität der Eingabe
	N	0
	C	–
INI	S	Unbestimmt
IND	Z	1, wenn B − 1 = 0, sonst 0
OUTI	H	Unbestimmt
OUTD	P/V	Unbestimmt
	N	1
	C	–
INIR	S	Unbestimmt
INDR	Z	1
OTIR	H	Unbestimmt
OTDR	P/V	Unbestimmt
	N	1
	C	–

n, e = 8-Bit-Zahl nn = 16-Bit-Zahl
s = Register A, B, C, D, E, H, L oder Speicherzelle Adresse in HL, IX + e, IY + e
r = Register A, B, C, D, E, H, L
dd = Registerpaar BC, DE, HL, IX, IY
cc = Bedingung Z, NZ, C, NC, PE, PO, M, P
kk = Bedingung Z, NZ, C, NC

10.4. Codierungstabelle *U880*

Opcode	U880	Opcode	U880	Opcode	U880
00	NOP	09	ADD HL, BC	11	LD DE, nn
01	LD BC, nn	0A	LD A, (BC)	12	LD (DE), A
02	LD (BC), A	0B	DEC BC	13	INC DE
03	INC BC	0C	INC C	14	INC D
04	INC B	0D	DEC C	15	DEC D
05	DEC B	0E	LD C, n	16	LD D, n
06	LD B, n	0F	RRCA	17	RLA
07	RLCA			18	JR e
08	EXAF	10	DJNZ e	19	ADD HL, DE

Opcode	U880	Opcode	U880	Opcode	U880
1A	LD A, (DE)	43	LD B, E	6D	LD L, L
1B	DEC DE	44	LD B, H	6E	LD L, M
1C	INC E	45	LD B, L	6F	LD L, A
1D	DEC E	46	LD B, M		
1E	LD E, n	47	LD B, A	70	LD M, B
1F	RRA	48	LD C, B	71	LD M, C
		49	LD C, C	72	LD M, D
20	JRNZ e	4A	LD C, D	73	LD M, E
21	LD HL, nn	4B	LD C, E	74	LD M, H
22	LD (adr), HL	4C	LD C, H	75	LD M, L
23	INC HL	4D	LD C, L	76	HALT
24	INC H	4E	LD C, M	77	LD M, A
25	DEC H	4F	LD C, A	78	LD A, B
26	LD H, n			79	LD A, C
27	DAA	50	LD D, B	7A	LD A, D
28	JRZ e	51	LD D, C	7B	LD A, E
29	ADD HL, HL	52	LD D, D	7C	LD A, H
2A	LD HL, (adr)	53	LD D, E	7D	LD A, L
2B	DEC HL	54	LD D, H	7E	LD A, M
2C	INC L	55	LD D, L	7F	LD A, A
2D	DEC L	56	LD D, M		
2E	LD L, n	57	LD D, A	80	ADD B
2F	CPL	58	LD E, B	81	ADD C
		59	LD E, C	82	ADD D
30	JRNC e	5A	LD E, D	83	ADD E
31	LD SP, nn	5B	LD E, E	84	ADD H
32	LD (adr), A	5C	LD E, H	85	ADD L
33	INC SP	5D	LD E, L	86	ADD M
34	INC M	5E	LD E, M	87	ADD A
35	DEC M	5F	LD E, A	88	ADC B
36	LD M, n			89	ADC C
37	SCF	60	LD H, B	8A	ADC D
38	JRC e	61	LD H, C	8B	ADC E
39	ADD HL, SP	62	LD H, D	8C	ADC H
3A	LD A, (adr)	63	LD H, E	8D	ADC L
3B	DEC SP	64	LD H, H	8E	ADC M
3C	INC A	65	LD H, L	8F	ADC A
3D	DEC A	66	LD H, M		
3E	LD A, n	67	LD H, A	90	SUB B
3F	CCF	68	LD L, B	91	SUB C
		69	LD L, C	92	SUB D
40	LD B, B	6A	LD L, D	93	SUB E
41	LD B, C	6B	LD L, E	94	SUB H
42	LD B, D	6C	LD L, H	95	SUB L

Opcode	U880	Opcode	U880	Opcode	U880
96	SUB M	BA	CMP D	DD	IX-Befehle
97	SUB A	BB	CMP E	DE	SBC n
98	SBC B	BC	CMP H	DF	RST 18H
99	SBC C	BD	CMP L		
9A	SBC D	BE	CMP M	E0	RPO
9B	SBC E	BF	CMP A	E1	POP HL
9C	SBC H			E2	JPPO nn
9D	SBC L	C0	RNZ	E3	EX (SP), HL
9E	SBC M	C1	POP BC	E4	CAPO nn
9F	SBC A	C2	JPNZ nn	E5	PUSH HL
		C3	JMP nn	E6	AND n
A0	AND B	C4	CANZ nn	E7	RST 2OH
A1	AND C	C5	PUSH BC	E8	RPE
A2	AND D	C6	ADD n	E9	JMP M
A3	AND E	C7	RST OH	EA	JPPE nn
A4	AND H	C8	RZ	EB	EX DE, HL
A5	AND L	C9	RET	EC	CAPE nn
A6	AND M	CA	JPZ nn	ED	Sondertrans-
A7	AND A	CB	Verschiebe-		portbefehle
A8	XOR B		und Bitbefehle	EE	XOR n
A9	XOR C	CC	CAZ nn	EF	RST 28H
AA	XOR D	CD	CALL nn		
AB	XOR E	CE	ADC n	F0	RP
AC	XOR H	CF	RST 8H	F1	POP AF
AD	XOR L			F2	JPP NN
AE	XOR M	D0	RNC	F3	DI
AF	XOR A	D1	POP DE	F4	CAP nn
		D2	JPNC nn	F5	PUSH AF
B0	OR B	D3	OUT n	F6	OR n
B1	OR C	D4	CANC nn	F7	RST 3OH
B2	OR D	D5	PUSH DE	F8	RM
B3	OR E	D6	SUB n	F9	LD SP, HL
B4	OR H	D7	RST 10H	FA	JPM nn
B5	OR L	D8	RC	FB	EI
B6	OR M	D9	EXX	FC	CAM nn
B7	OR A	DA	JPC nn	FD	IY-Befehle
B8	CMP B	DB	IN n	FE	CMP n
B9	CMP C	DC	CAC nn	FF	RST 38H

Verschiebe- und Bitbefehle

1. Byte CB

2. Byte	7 6	5 4 3	2 1 0
			REG
	0 0	RLC	0 B
	0 1	RRC	1 C
	0 2	RL	2 D
	0 3	RR	3 E
	0 4	SLA	4 H
	0 5	SRA	5 L
	0 6	–	6 M
	0 7	SRL	7 A
	1 0	BIT 0, r	
	1 1	BIT 1, r	
	1 2	BIT 2, r	
	1 3	BIT 3, r	
	1 4	BIT 4, r	
	1 5	BIT 5, r	
	1 6	BIT 6, r	
	1 7	BIT 7, r	
	2 0	RES 0, r	
	2 1	RES 1, r	
	2 2	RES 2, r	
	2 3	RES 3, r	
	2 4	RES 4, r	
	2 5	RES 5, r	
	2 6	RES 6, r	
	2 7	RES 7, r	
	3 0	SET 0, r	
	3 1	SET 1, r	
	3 2	SET 2, r	
	3 3	SET 3, r	
	3 4	SET 4, r	
	3 5	SET 5, r	
	3 6	SET 6, r	
	3 7	SET 7, r	

Indexbefehle mit IX

(bei Indexbefehlen mit IY ist das 1. Byte FD)

DD09	ADD	IX, BC
DD19	ADD	IX, DE
DD21	LD	IX, nn
DD22	LD	(adr), IX
DD23	INC	IX
DD29	ADD	IX, IX
DD2A	LD	IX, (adr)
DD2B	DEC	IX
DD34	INC	(IX + offset)
DD35	DEC	(IX + offset)
DD36	LD	(IX + offset), n
DD39	ADD	IX, SP
DD46	LD	B, (IX + offset)
DD4E	LD	C, (IX + offset)
DD56	LD	D, (IX + offset)
DD5E	LD	E, (IX + offset)
DD66	LD	H, (IX + offset)
DD6E	LD	L, (IX + offset)
DD70	LD	(IX + offset), B
DD71	LD	(IX + offset), C
DD72	LD	(IX + offset), D
DD73	LD	(IX + offset), E
DD74	LD	(IX + offset), H
DD75	LD	(IX + offset), L
DD77	LD	(IX + offset), A
DD7E	LD	A, (IX + offset)
DD86	ADD	A, (IX + offset)
DD8E	ADC	A, (IX + offset)
DD96	SUB	(IX + offset)
DD9E	SBC	A, (IX + offset)
DDA6	AND	(IX + offset)
DDAE	XOR	(IX + offset)
DDB6	OR	(IX + offset)
DDBE	CP	(IX + offset)
DDCB of 06	RLC	(IX + offset)
DDCB of 0E	RRC	(IX + offset)
DDCB of 16	RL	(IX + offset)
DDCB of 1E	RR	(IX + offset)
DDCB of 26	SIA	(IX + offset)
DDCB of 2E	SRA	(IX + offset)
DDCB of 3E	SRL	(IX + offset)
DDCB of 46	BIT	0, (IX + offset)

DDCB of 4E	BIT	1, (IX + offset)
DDCB of 56	BIT	2, (IX + offset)
DDCB of 5E	BIT	3, (IX + offset)
DDCB of 66	BIT	4, (IX + offset)
DDCB of 6E	BIT	5, (IX + offset)
DDCB of 76	BIT	6, (IX + offset)
DDCB of 7E	BIT	7, (IX + offset)
DDCB of 86	RES	0, (IX + offset)
DDCB of 8E	RES	1, (IX + offset)
DDCB of 96	RES	2, (IX + offset)
DDCB of 9E	RES	3, (IX + offset)
DDCB of A6	RES	4, (IX + offset)
DDCB of AE	RES	5, (IX + offset)
DDCB of B6	RES	6, (IX + offset)
DDCB of BE	RES	7, (IX + offset)
DDCB of C6	SET	0, (IX + offset)
DDCB of CE	SET	1, (IX + offset)
DDCB of D6	SET	2, (IX + offset)
DDCB of DE	SET	3, (IX + offset)
DDCB of E6	SET	4, (IX + offset)
DDCB of EE	SET	5, (IX + offset)
DDCB of F6	SET	6, (IX + offset)
DDCB of FE	SET	7, (IX + offset)
DDE1	POP	IX
DDE3	EX	(SP), IX
DDE5	PUSH	IX
DDE9	JP	(IX)
DDF9	LD	SP, IX

Sondertransportbefehle

ED40	IN	B
ED41	OUT	B
ED42	SBC	HL, BC
ED43	LD	(adr), BC
ED44	NEG	
ED45	RETN	
ED46	IM	0
ED47	LD	I, A
ED48	IN	C
ED49	OUT	C
ED4A	ADC	HL, BC
ED4B	LD	BC, (adr)
ED4D	RETI	
ED4F	LD	R, A
ED50	IN	D
ED51	OUT	D
ED52	SBC	HL, DE
ED53	LD	(adr), DE
ED56	IM	1
ED57	LD	A, I
ED58	IN	E
ED59	OUT	E
ED5A	ADC	HL, DE
ED5B	LD	DE, (adr)
ED5E	IM	2
ED5F	LD	A, R
ED60	IN	H
ED61	OUT	H
ED62	SBC	HL, HL
ED67	RRD	
ED68	IN	L
ED69	OUT	L
ED6A	ADC	HL, HL
ED6F	RLD	
ED70	IN	nur Flags
ED72	SBC	HL, SP
ED73	LD	(adr), SP
ED78	IN	A
ED79	OUT	A
ED7A	ADC	HL, SP
ED7B	LD	SP, (adr)
EDAO	LDI	
EDA1	CPI	
EDA2	INI	
EDA3	OUTI	
EDA8	LDD	
EDA9	CPD	
EDAA	IND	
EDAB	OUTD	
EDBO	LDIR	
EDB1	CPIR	
EDB2	INIR	
EDB3	OTIR	
EDB6	LDDR	
EDB9	CPDR	
EDBA	INDR	
EDBB	OTDR	

10.5. Befehlsübersicht *8080*

Einzelworttransfer

				C	Z	P	S	H
MOV	r, s	s → r	s, r = N, m					
MVI	r, n	n → r	[N ist Register A, B, C, C, B, H, L, M ist Speicherzelle, ADR = (HL)]					
STAX	ss	A → ⟨ss⟩	ss = BC, DE					
LDAX	ss	⟨ss⟩ → A	ss = BC, DE					
STA	nn	A → ⟨nn⟩						
LDA	nn	⟨nn⟩ → A						
SPHL		HL → SP						

Doppelworttransfer

			C	Z	P	S	H
LXI	dd, nn	nn → dd	dd = BC, DE, HL				
SHLD	nn	HL → ⟨nn⟩					
LHLD	nn	⟨nn⟩ → HL					
PUSH	ss	ss → STACK	ss = BC, DE, HL, AF				
POP	ss	STACK → ss	ss = BC, DE, HL, AF				

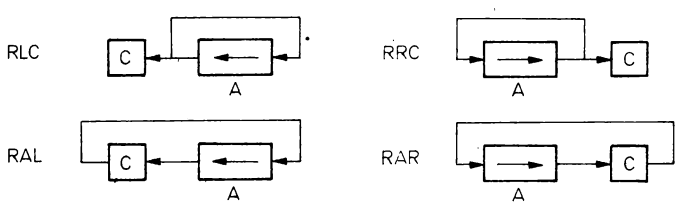
Registeraustausch

		C	Z	P	S	H
XCHG	HL ↔ DE					
XTHL	HL ↔ ⟨SP + 1⟩, ⟨SP⟩					

Rechenoperationen und logische Operationen mit 1 Operanden

			C	Z	P	S	H
CMC		$\overline{C} \rightarrow C$	↕				X
STC		1 → C	1				0
CMA		$\overline{A} \rightarrow A$	·	·	·	·	1
DAA		BCD ⟨A⟩ → A	↕	↕	↕	↕	↕
INR d		d + 1 → d d = N, M		↕	↕	↕	↕
DCR d		d − 1 → d d = N, M		↕	↕	↕	↕
INX	dd	dd + 1 → dd	dd = BC, DE, HL, SP				
DCX	dd	dd − 1 → dd					

Verschiebebefehle



Rechenoperationen mit 2 Operanden

8-Bit-Operationen

				C	Z	P	S	H
ADD	r	A + r → A		↕	↕	↕	↕	↕
ADC	r	A + r + C → A		↕	↕	↕	↕	↕
SUB	r	A − r → A		↕	↕	↕	↕	↕
SBB	r	A − r − C → A		↕	↕	↕	↕	↕
ANA	r	A ∧ r → A	r = N, M	↕	↕	↕	↕	↕
ORA	r	A ∨ r → A		↕	↕	↕	↕	↕
XRA	r	A ⊕ r → A		↕	↕	↕	↕	↕
CMP	r	Vergleich A, r		↕	↕	↕	↕	↕
		A − r setzt die Flags		↕	↕	↕	↕	↕
ADI	n	A + n → A		↕	↕	↕	↕	↕
ACI	n	A + n + C → A		↕	↕	↕	↕	↕
SUI	n	A − n → A		↕	↕	↕	↕	↕
SBI	n	A − n − C → A		↕	↕	↕	↕	↕
ANI	n	A ∧ n → A		↕	↕	↕	↕	↕
ORI	n	A ∨ n → A		↕	↕	↕	↕	↕
XRI	n	A ⊕ n → A		↕	↕	↕	↕	↕
CPI	n	Vergleich A, n		↕	↕	↕	↕	↕
		A − n setzt die Flags		↕	↕	↕	↕	↕

16-Bit-Operationen

				C	Z	P	S	H
DAD	ss	HL + ss → HL	ss = BC, DE, HL, SP	↕				X

Sprungbefehle

			C	Z	P	S	H
PCHL		HL → PC					
JMP	nn	nn → PC					
Jcc	nn	nn → PC, wenn erfüllt cc = NZ, Z, NC, C, PO, PE, P, M					

Unterprogrammbefehle

			C	Z	P	S	H
CALL	nn	PC → STACK, nn → PC					
Ccc	nn	CALL nn, wenn cc erfüllt, sonst Leerbefehl cc = NZ, Z, NC, C, PO, PE, P, M					
RST	Z	wie CALL Z Z = OH, 8H, 10H, ..., 38H					
RET		STACK → PC					
Rcc		RET, wenn cc erfüllt, sonst Leerbefehl cc = NZ, Z, NC, C, PO, PE, P, M					

Ein-/Ausgabebefehle

			Adreßbus $A_{15} \dots A_8$	Adreßbus $A_7 \dots A_0$	C	Z	P	S	H
IN	n	⟨n⟩ → A	A	n					
OUT	n	A → ⟨n⟩	A	n					

C Z P S H

EI INTERRUPT frei
DI INTERRUPT gesperrt
HLT HALT
NOP keine Operation

10.6. Codierungstabelle *8080*

Opcode	8080	Opcode	8080	Opcode	8080
00	NOP	20	–	40	MOV B, B
01	LXI B, dddd	21	LXI H, dddd	41	MOV B, C
02	STAX B	22	SHLD adr	42	MOV B, D
03	INX B	23	INX H	43	MOV B, E
04	INR B	24	INR H	44	MOV B, H
05	DCR B	25	DCR H	45	MOV B, L
06	MVI B, dd	26	MVI H, dd	46	MOV B, M
07	RLC	27	DAA	47	MOV B, A
08		28	–	48	MOV C, B
09	DAD B	29	DAD H	49	MOV C, C
0A	LDAX B	2A	LHLD adr	4A	MOV C, D
0B	DCX B	2B	DCX H	4B	MOV C, E
0C	INR C	2C	INR L	4C	MOV C, H
0D	DCR C	2D	DCR L	4D	MOV C, L
0E	MVI C, dd	2E	MVI L, dd	4E	MOV C, M
0F	RRC	2F	CMA	4F	MOV C, A
10	–	30	–	50	MOV D, B
11	LXI D, dddd	31	LXI SP, dddd	51	MOV D, C
12	STAX D	32	STA adr	52	MOV D, D
13	INX D	33	INX SP	53	MOV D, E
14	INR D	34	INR M	54	MOV D, H
15	DCR D	35	DCR M	55	MOV D, L
16	MVI D, dd	36	MVI M, dd	56	MOV D, M
17	RAL	37	STC	57	MOV D, A
18	–	38	–	58	MOV E, B
19	DAD D	39	DAD SP	59	MOV E, C
1A	LDAX D	3A	LDA adr	5A	MOV E, D
1B	DCX D	3B	DCX SP	5B	MOV E, E
1C	INR E	3C	INR A	5C	MOV E, H
1D	DCR E	3D	DCR A	5D	MOV E, L
1E	MVI E, dd	3E	MVI A, dd	5E	MOV E, M
1F	RAR	3F	CMC	5F	MOV E, A

Opcode	8080
60	MOV H, B
61	MOV H, C
62	MOV H, D
63	MOV H, E
64	MOV H, H
65	MOV H, L
66	MOV H, M
67	MOV H, A
68	MOV L, B
69	MOV L, C
6A	MOV L, D
6B	MOV L, E
6C	MOV L, H
6D	MOV L, L
6E	MOV L, M
6F	MOV L, A
70	MOV M, B
71	MOV M, C
72	MOV M, D
73	MOV M, E
74	MOV M, H
75	MOV M, L
76	HLT
77	MOV M, A
78	MOV A, B
79	MOV A, C
7A	MOV A, D
7B	MOV A, E
7C	MOV A, H
7D	MOV A, L
7E	MOV A, M
7F	MOV A, A
80	ADD B
81	ADD C
82	ADD D
83	ADD E
84	ADD H
85	ADD L
86	ADD M
87	ADD A
88	ADC B
89	ADC C

Opcode	8080
8A	ADC D
8B	ADC E
8C	ADC H
8D	ADC L
8E	ADC M
8F	ADC A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
96	SUB M
97	SUB A
98	SBB B
99	SBB C
9A	SBB D
9B	SBB E
9C	SBB H
9D	SBB L
9E	SBB M
9F	SBB A
A0	ANA B
A1	ANA C
A2	ANA D
A3	ANA E
A4	ANA H
A5	ANA L
A6	ANA M
A7	ANA A
A8	XRA B
A9	XRA C
AA	XRA D
AB	XRA E
AC	XRA H
AD	XRA L
AE	XRA M
AF	XRA A
B0	ORA B
B1	ORA C
B2	ORA D
B3	ORA E

Opcode	8080
B4	ORA H
B5	ORA L
B6	ORA M
B7	ORA A
B8	CMP B
B9	CMP C
BA	CMP D
BB	CMP E
BC	CMP H
BD	CMP L
BE	CMP M
BF	CMP A
C0	RNZ
C1	POP B
C2	JNZ adr
C3	JMP adr
C4	CNZ adr
C5	PUSH B
C6	ADI dd
C7	RST O
C8	RZ
C9	RET
CA	JZ adr
CB	–
CC	CZ adr
CD	CALL adr
CE	ACI dd
CF	RST 1
D0	RNC
D1	POP D
D2	JNC adr
D3	OUT port
D4	CNC adr
D5	PUSH D
D6	SUI dd
D7	RST 2
D8	RC
D9	–
DA	JC adr
DB	IN port
DC	CC adr
DD	–

Opcode	8080	Opcode	8080	Opcode	8080
DE	SBI dd	E9	PCHL	F4	CP adr
DF	RST 3	EA	JPE adr	F5	PUSH PSW
		EB	XCHG	F6	ORI dd
E0	RPO	EC	CPE adr	F7	RST 6
E1	POP H	ED	–	F8	RM
E2	JPO adr	EE	XRI dd	F9	SPHL
E3	XTHL	EF	RST 5	FA	JM adr
E4	CPO adr			FB	EI
E5	PUSH H	F0	RP	FC	CM adr
E6	ANI dd	F1	POP PSW	FD	–
E7	RST 4	F2	JP adr	FE	CPI dd
E8	RPE	F3	DI	FF	RST 7

10.7. ASCII-Code

Der American Standard Code for Informations Interchange (Tabelle 10.1.) ist der am meisten verwendete Code für alphanumerische Zeichen. Mit 7 Bit werden die wichtigsten Zeichen codiert. Das 8. Bit benutzt man als Prüfbit oder für einen 2. Zeichensatz für Drucker oder Bildschirm.

Tabelle 10.1.
ASCII-Code

Dez	Oktal	Hex	Zeichen	Bemerkungen
0	000	00	NUL	Null, leer
1	001	01	SOH	Anfang des Kopfes (Start of heading)
2	002	02	STX	Anfang des Textes (Start of text)
3	003	03	ETX	Ende des Textes (End of text)
4	004	04	EOT	Ende der Übertragung (End of transmission)
5	005	05	ENQ	Anfrage (Enquiry); Wer ist dort?
6	006	06	ACK	Bestätigung (Acknowledge)
7	007	07	BEL	Akustisches Signal (Bell)
8	010	08	BS	Rücktaste, Rücksetzen um ein Zeichen (Backspace)
9	011	09	HT	Horizontaler Tabulator (Horizontal tab)
10	012	0A	LF	Zeilenvorschub (Line feed)
11	013	0B	VT	Vertikaler Tabulator (Vertical tab)
12	014	0C	FF	Formatsteuerung (Form feed)
13	015	0D	CR	Wagenrücklauf (CARRIAGE return)
14	016	0E	SO	Umschalten auf Großbuchstaben (Shift out)
15	017	0F	SI	Umschalten auf Kleinbuchstaben (SHIFT in), Umschalten auf schwarzes Farbband

Dez	Oktal	Hex	Zeichen	Bemerkungen
16	020	10	DLE	Umschaltung der Datenübertragung (Data link escape)
17	021	11	DC 1	Gerätesteuerzeichen 1
18	022	12	DC 2	Gerätesteuerzeichen 2
19	023	13	DC 3	Gerätesteuerzeichen 3
20	024	14	DC 4	Gerätesteuerzeichen 4
21	025	15	NAK	Negative Bestätigung (Negativ acknowledge), Fehler
22	026	16	SYN	Synchronisieren
23	027	17	ETB	Ende des übertragenen Blockes (End of the transmission block) auch LEM (logisches Ende des Mediums)
24	030	18	CAN	Annullierung (CANCEL)
25	031	19	EM	Ende des Mediums
26	032	1A	SUB	Substitution
27	033	1B	ESC	Escape
28	034	1C	FS	File separator
29	035	1D	GS	Group separator
30	036	1E	RS	(Record separator) Wagenrücklauf mit Zeilenvorschub
31	037	1F	US	Unit separator
32	040	20	SP	Leerzeichen (Space)
33	041	21	!	Ausrufezeichen (Exclamation point)
34	042	22	"	Anführungsstriche
35	043	23	#	Nummernzeichen
36	044	24	$	Währungszeichen
37	045	25	%	Prozentzeichen
38	046	26	&	Ampersand
39	047	27	'	Hochkomma
40	050	28	(	
41	051	29	)	
42	052	2A	*	
43	053	2B	+	
44	054	2C	,	
45	055	2D	-	
46	056	2E	.	
47	057	2F	/	
48	060	30	0	
49	061	31	1	
50	062	32	2	
51	063	33	3	
52	064	34	4	
53	065	35	5	
54	066	36	6	
55	067	37	7	
56	070	38	8	
57	071	39	9	
58	072	3A	:	
59	073	3B	;	
60	074	3C	<	
61	075	3D	=	
62	076	3E	>	
63	077	3F	?	

Dez	Oktal	Hex	Zeichen	Bemerkungen
64	100	40	@	
65	101	41	A	
66	102	42	B	
67	103	43	C	
68	104	44	D	
69	105	45	E	
70	106	46	F	
71	107	47	G	
72	110	48	H	
73	111	49	I	
74	112	4A	J	
75	113	4B	K	
76	114	4C	L	
77	115	4D	M	
78	116	4E	N	
79	117	4F	O	
80	120	50	P	
81	121	51	Q	
82	122	52	R	
83	123	53	S	
84	124	54	T	
85	125	55	U	
86	126	56	V	
87	127	57	W	
88	130	58	X	
89	131	59	Y	
90	132	5A	Z	
91	133	5B	[	
92	134	5C	\	
93	135	5D	]	
94	136	5E	→	
95	137	5F	–	
96	140	60	/	
97	141	61	a	
98	142	62	b	
99	143	63	c	
100	144	64	d	
101	145	65	e	
102	146	66	f	
103	147	67	g	
104	150	68	h	
105	151	69	i	
106	152	6A	j	
107	153	6B	k	
108	154	6C	l	
109	155	6D	m	
110	156	6E	n	
111	157	6F	o	
112	160	70	p	
113	161	71	q	
114	162	72	r	
115	163	73	s	
116	164	74	t	

Dez	Oktal	Hex	Zeichen	Bemerkungen
117	165	75	u	
118	166	76	v	
119	167	77	w	
120	170	78	x	
121	171	79	y	
122	172	7A	z	
123	173	7B	{	
124	174	7C	|	
125	175	7D	}	
126	176	7E	~	
127	177	7F	DEL	

10.8. EBCDIC-Code

Der Extended Binary Coded-Decimal Informations-Code (Tabelle 10.2.) ist ein erweiterter BCD-Code zum Informationsaustausch. Dieser 8-Bit-Code von *IBM* benutzt alle Codierungen von 00 bis FF. Im wesentlichen werden mit ihm die gleichen Zeichen dargestellt wie im ASCII-Code. Er hat jedoch eine Reihe Erweiterungsmöglichkeiten für Spezialzeichen.

Tabelle 10.2.
EBCDIC-Code

Hex	Zeichen	Hex	Zeichen	Hex	Zeichen	Hex	Zeichen	Hex	Zeichen	Hex	Zeichen
0		30		60		90		C0		F0	0
1		31		61		91	j	C1	A	F1	1
2		32		62		92	k	C2	B	F2	2
3		33		63		93	L	C3	C	F3	3
4		34		64		94	m	C4	D	F4	4
5		35		65		95	n	C5	E	F5	5
6		36		66		96	o	C6	F	F6	6
7		37		67		97	p	C7	G	F7	7
8		38		68		98	q	C8	H	F8	8
9		39		69		99	r	C9	I	F9	9
A		3A		6A		9A		CA		FA	
B		3B		6B		9B		CB		FB	
C		3C		6C	%	9C		CC		FC	
D		3D		6D	–	9D		CD		FD	
E		3E		6E		9E		CE		FE	
F		3F		6F	?	9F		CF		FF	
10		40	Blank	70		A0		D0			
11		41		71		A1		D1	J		
12		42		72		A2	s	D2	K		
13		43		73		A3	t	D3	L		
14		44		74		A4	u	D4	M		
15		45		75		A5	v	D5	N		
16		46		76		A6	w	D6	O		
17		47		77		A7	x	D7	P		
18		48		78		A8	y	D8	Q		

19		49		79		A9	z	D9	R		
1A		4A		7A	:	AA		DA			
1B		4B		7B	#	AB		DB			
1C		4C		7C		AC		DC			
1D		4D	(	7D	'	AD		DD			
1E		4E	+	7E	=	AE		DE			
1F		4F		7F	"	AF		DF			
20		50	&	80		B0		E0			
21		51		81	a	B1		E1			
22		52		82	b	B2		E2	S		
23		53		83	c	B3		E3	T		
24		54		84	d	B4		E4	U		
25		55		85	e	B5		E5	V		
26		56		86	f	B6		E6	W		
27		57		87	g	B7		E7	X		
28		58		88	h	B8		E8	Y		
29		59		89	i	B9		E9	Z		
2A		5A		8A		BA		EA			
2B		5B	$	8B		BB		EB			
2C		5C		8C		BC		EC			
2D		5D	)	8D		BD		ED			
2E		5E	;	8E		BE		EE			
2F		5F		8F		BF		EF			

11. Literatur

[1] *W. Schwarz/G. Meyer/D. Eckhard*, Mikrorechner – Wirkungsweise, Programmierung, Applikation. Berlin 1984

[2] *L. Claßen*, Programmierung des Mikroprozessorsystems *U880 – K1520*. Berlin 1985

[3] *J. Matschke*, Von der einfachen Logikschaltung zum Mikrorechner. Berlin 1986

[4] *D. Werner*, Programmierung von Mikrorechnern. Berlin 1986

[5] *TGL 26176* Unipolarer Mikroprozessorschaltkreis *U880D*

[6] *TGL 35001* Unipolarer serieller Ein-/Ausgabe-Schaltkreis *U856D*

[7] *TGL 35837* Unipolarer Parallel-Ein-/Ausgabe-Schaltkreis *U855D*

[8] *TGL 35002* Unipolarer Zähler-/Zeitgeber-Schaltkreis *U857D*

[9] *TGL 37003* Unipolarer Schaltkreis für direkten Speicherzugriff der Serie *U858D*

Die Veröffentlichung enthält eine zusammenfassende Darstellung der Grundlagen zur 8-Bit-Mikrorechentechnik.
Inhaltliche Schwerpunkte sind:

- Die Arbeitsweise und Programmierung
  - der Prozessorfamilie *U 880* mit den peripheren Bausteinen PIO, SIO, CTC, DMA sowie
  - der Prozessorfamilie *8080* mit den peripheren Bausteinen *8251, 8255* und *8212.*
- Die Beschreibung von Busschaltkreisen, Codier- und Decodierschaltkreisen.
- Die Assemblersprache für die Prozessoren *U 880* und *8080* mit Programmierbeispielen. und mathematischen Grundprogrammen.
- Die Software der 8-Bit-Technik.
- Zahlenumrechnungstabellen – Übersichtstabellen für Befehls- und Zeichencodierungen.

ISBN 3-327-00668-7
00900