veb mikroelektronik "wilhelm pieck" mühlhausen





# I n h a l t s v e r z e i c h n i s

# 1. E i n l e i t u n g

Die Programme ASSMON1 und ASSMON2 stellen ein komplettes U880-Assemblerentwicklungssystem dar. Das Programm ASSMON1 enthält einen Editor und einen 2-Pass-Assembler. Im Programm ASSMON2 sind ein Testmonitor und ein Disassembler zusammengefaßt. Die Lage der Programme im Speicher muß vom Anwender vor dem Laden festgelegt werden, ist also nicht vorgegeben. Die Syntax des verwendeten Assemblers entspricht der UDOS-Sprachversion.
Für eine effektive Programmierung des KC compact in Maschinensprache ist das zum Computer gehörige Systemhandbuch unbedingt erforderlich, welches im Fachhandel erhältlich ist. Darin sind unter anderem alle wichtigen Systemunterprogramme und Arbeitszellen beschrieben.
Neben dem Assembler im Programm ASSMON1 wird dem Programmierer mit dem Programm ASSMON2 ein Debugger und Disassembler bereitgestellt. Damit wird ein Rückübersetzen in Befehlsmnemonik, ein schrittweises Abarbeiten oder ein Abarbeiten in Echtzeit von bereits vorhandenem Maschinencode möglich.
In der nachfolgenden Beschreibung werden gleichzeitig zu betätigende Tasten mit "-" (z.B. ^CTRL_-^A_) und nacheinander zu betätigende Tasten mit "+" (z.B. ^X_+^Y_) zwischen den Tasten dargestellt.
Die beiden Programme ASSMON1 und ASSMON2 sind jeweils mit ihren Ladeprogrammen hintereinander auf der Kassette CC1001 abgespeichert. Soll ein Programm geladen werden, wird die Kassette an den Programmanfang des jeweiligen Ladeprogrammes gespult und

RUN" ^RETURN_+^weitere Taste_

in den Computer eingegeben. Nach dem Einlesen des Ladeprogramms ist gegebenenfalls der Kassettenrecorder zu stoppen, wenn keine Motorschaltung vorhanden ist. Das Ladeprogramm startet selbständig und fordert vom Bediener die Eingabe der eigentlichen Ladeadresse von ASSMON1 bzw. ASSMON2. Es können Zahlen zwischen 1000 und 30000 eingegeben werden. ASSMON1 sollte sinnvollerweise im unteren RAM-Bereich (Eingabe von 1000) geladen werden, da das Quellprogramm, die Adreßtabelle und die Maschinenphase im Normalfall hinter ASSMON1 abgelegt werden. ASSMON2 muß jeweils in den RAM-Bereich gelegt werden, der nicht vom zu testenden Programm belegt wird. Sollen beide Programme gleichzeitig im Speicher gehalten werden, wird ASSMON1 unten (Adresse 1000) und ASSMON2 oben (Adresse 30000) in den RAM geladen. In diesem Fall muß ASSMON2 zuerst geladen werden.

Die Programme sind auf der Kassette wie folgt angeordnet:

| Programm | Zählerstand | |
|---|---|---|
| | Geracord | LCR-C |
| ASSMON1 | 010 | 018 |
| ASSMON2 | 060 | 090 |

Auf der A-Seite sind die Programme mit 1000 Baud und auf der B-Seite mit 2000 Baud Übertragungsrate abgespeichert. Dabei kann die B-Seite mit 2000 Baud nur bei optimal eingestelltem Kassettenrecorder eingelesen werden.

# 2. E d i t o r

## 2.1. Einführung

Der Editor, der in ASSMON1 enthalten ist, ist ein einfacher, zeilenorientierter Aufbereiter. Die Quellprogramme können mit seiner Hilfe eingegeben und geändert werden. Ähnlich wie in BASIC erfolgt hier die Bearbeitung über einen Puffer, dessen Inhalt erst mit dem Zeilenabschluß (^ENTER_) in die Textdatei übernommen wird. In der Textdatei werden Zwischenräume komprimiert, um Speicherplatz zu sparen. Die Zeile wird deshalb bei der Übernahme vom Puffer in die Textdatei auf Zwischenräume kontrolliert. Gefundene Zwischenräume werden durch ein Tabulatorzeichen ersetzt. Das erleichtert nachher auch die formatierte Ausgabe auf dem Bildschirm oder dem Drucker. Im Kommentar werden Zwischenräume nicht komprimiert.
Nach dem Einladen von ASSMON1 wird unter anderem folgendes Hilfsmenü auf dem Bildschirm angezeigt:

```
   Assemble                 Bye
   Current+ State           Delete
   Edit                     Find
   Get text                 Help
   Insert                   CTRL/J -> ASSMON2
   List                     Move
   reNumber                 Object
   Put text                 Q put ASCII
   Run                      Separator
   Tape speed               Upper line
   Verify                   Width
   teXt info                Y print length
   Z print text
```

Danach erscheint das Promptzeichen '>' und der Editor wartet auf Kommandos. Als Kommandos werden nur einzelne Buchstaben eingegeben. Bei den angezeigten Kommandos des Hilfsmenüs ist jeweils ein Buchstabe groß geschrieben. Dieser Buchstabe muß eingegeben werden, um das jeweilige Kommando zu starten.
Kommandozeilen haben folgendes allgemeingültiges Format:

```
>K a,b,c,d ^RETURN_

   K ...   Kommando, welches ausgeführt werden soll
 a,b ...   Zahl von 1 - 32767
 c,d ...   Zeichenkette maximal 20 Stellen
```

Die Anzahl der verlangten Argumente ist vom Kommando abhängig. Der Editor speichert die eingegebenen Argumente. Diese können dann bei nachfolgenden Kommandos weiterverwendet werden. Beim

Programmstart sind a und b auf den Wert 10 gesetzt und die Zeichenketten sind leer. Die Kommandobuchstaben können in Klein- oder Großbuchstaben eingegeben werden.
Bei fehlerhaft eingegebener Kommandozeile erscheint die Fehlermeldung 'PARDON?'.
In den folgenden Abschnitten werden die einzelnen Kommandos beschrieben. Falls ein Argument in den Zeichen '^_' eingeschlossen ist, muß dieses Argument unbedingt angegeben werden, um das Kommando durchführen zu können.

## 2.2. Editor-Kommandos

Texteingabe
---

Zur Texteingabe gibt es zwei Möglichkeiten, genau wie in BASIC. Zum einen kann man Zeile für Zeile durch Eingabe der Zeilennummer und des zugehörigen Textes eingeben. Zum zweiten besteht die Möglichkeit der automatischen Zeilennummernbereitstellung (in BASIC: AUTO-Kommando).

Kommando: I n,m (Insert)

n ... 1.Zeile
m ... Schrittweite Zeilennummerierung

Wird eine bereits vorhandene Zeilennummer eingegeben, wird die zugehörige Zeile in der Textdatei mit der neuen Zeile überschrieben bzw. gelöscht, wenn nur die Zeilennummer und ^RETURN_ eingegeben wurde. Es sind Zeilennummern von 1 bis 32767 zulässig.
Bei der Texteingabe sind folgende Kontroll-Funktionen möglich:

| | | |
|---|---|---|
| ^RETURN_ | ... | Abschluß der Eingabe einer Zeile |
| ^TAB_ | ... | Sprung zur nächsten Tabulatorposition |
| ^CTRL_-^X_ | ... | Löschen der Zeile bis zum Beginn |
| ^ESC_ | ... | Rückkehr in den Kommandomodus |
| ^DEL_ | ... | Löschen des Zeichens vor der Cursorposition |

Nähert sich die Textdatei dem RAM-Ende, gibt der Editor die Meldung 'Bad Memory' aus. Es kann an dieser Stelle kein weiterer Text eingegeben werden, und die Datei sollte auf Band gesichert werden.

## Textausgabe

Kommando: L n,m (List)

n ... 1. auszugebende Zeile
m ... letzte auszugebende Zeile

Ohne Argumentangabe werden immer die Standardwerte 1 und 32767 angenommen. Die Auflistung erfolgt in formatierter Form. Falls Zeilennummer m noch nicht erreicht ist, werden immer 24 Zeilen ausgegeben. Danach kann man mit ^ESC_ in den Kommandomodus zurückkehren oder mit Betätigung jeder anderen Taste weiterlisten.

## Textbearbeitung

Die nachfolgend beschriebenen Kommandos sind die eigentlichen Editierkommandos. Sie unterstützen das Verändern der gesamten Textdatei bzw. einer einzelnen Zeile.

Kommando: E n (Edit)

n ... Zeilennummer

Die Zeile mit der Nummer n wird zur Bearbeitung in den Eingabepuffer kopiert. Alle Bearbeitungen finden im Puffer statt und nicht in der Textdatei. Bis zur Übernahme des Pufferinhaltes in die Datei mit ^RETURN_ kann jederzeit wieder auf die Originalzeile zurückgegriffen werden. Nach Aufruf von 'E n' wird die Zeile auf den Bildschirm geschrieben und die gleiche Zeilennummer mit nachfolgendem Cursor darunter ausgegeben. Der Cursor bildet einen gedanklichen Zeiger, der über die Zeile bewegt werden kann. Folgende Unterkommandos stehen zur Verfügung:

| | |
|---|---|
| ^SPACE_ | erhöht Zeiger um 1 |
| ^DEL_ | vermindert Zeiger um 1 |
| ^RETURN_ | Beenden der Aufbereitung, Pufferinhalt wird in die Textdatei übernommen |
| ^Q_ | Verzichten auf Veränderung der Zeile, Pufferinhalt wird nicht in die Textdatei übernommen |
| ^R_ | erneutes Laden der Textdatei aus der Datei in den Puffer |
| ^L_ | Rest der Zeile ausgeben, Zeiger steht danach wieder am Zeilenanfang |
| ^K_ | Lösche Zeichen, auf dem der Zeiger steht |
| ^Z_ | Lösche alle Zeichen von Zeigerposition bis Zeilenende |
| ^F_ | Suche nächstes Vorkommen der Zeichenkette f (siehe Kommando F) in der Textdatei, bei keinem weiteren Vorkommen wird das Kommando abgebrochen |
| ^S_ | Ersetze gefundene Zeichenkette f mit der vorher |

definierten Zeichenkette s und suche nächstes Vorkommen der Zeichenkette f

^I_ Einfügen von Zeichen an der Zeigerposition, der Einfüge-Cursor wird als "*" dargestellt, verlassen wird der Einfügemodus mit ^RETURN_

^X_ setzt Zeiger auf Zeilenende und ruft Unterkommando I auf

^C_ Aufruf der Änderungsfunktion innerhalb des Editier-Modus, Zeichen auf Zeigerposition werden durch Tasteneingabe überschrieben und der Zeiger um 1 erhöht, verlassen wird dieses Unterkommando durch ^RETURN_ oder automatisch, wenn das Zeilenende erreicht wird, während des "Änderungsmodus" wird der Cursor als "+" dargestellt.

Kommando: D ^n,m_ (Delete)

n ... 1.Zeile
m ... letzte Zeile

Die Zeilen von n bis m werden in der Textdatei gelöscht. Für eine Einzelzeile muß n=m sein. Eine Einzelzeile kann auch durch Eingabe von Zeilennummer und ^RETURN_ gelöscht werden.

Kommando: M n,m (Move)

n ... Quellzeile
m ... Zielzeile

Zeile n wird in Zeile m kopiert. Dabei wird m, wenn vorhanden, überschrieben und Zeile n gelöscht.

Kommando: N ^n,m_ (Renumber)

Es wird eine Neunumerierung der Zeilen ab Zeile n mit der Schrittweite m durchgeführt. Sollte durch Renumber eine Zeilennummer größer 32767 gebildet werden, wird Renumber nicht ausgeführt und die alte Numerierung bleibt erhalten.

Kommando: F n,m,f,s

Die Textdatei wird von der Zeile n bis zur Zeile m nach der Zeichenkette f durchsucht. Wird die Kette gefunden, wird in das Kommando E verzweigt. Dort können dann die dort möglichen Unterkommandos benutzt werden. Wurde der Zeilenbereich und die Zeichenkette bereits bei einem vorhergehenden Kommando eingegeben, braucht zur Suche lediglich noch 'F' eingegeben zu werden.

## 2.3. Lade- und Rette-Kommandos

Die nachfolgend beschriebenen Kommandos dienen allgemein dem Datenaustausch zwischen dem RAM des Computers und einem externen Datenträger (Kassette oder Diskette). Es können Textdateien oder Objektcode (Maschinenprogramme) gerettet bzw. geladen werden.

Kommando: P n,m,s (Put text)

n ... 1.Zeile
m ... letzte Zeile
s ... Name der Datei

Die Textdatei wird von der Zeile n bis zur Zeile m unter dem Dateinamen s auf Band gerettet. Wurden n,m und s durch vorhergehende Kommandos bereits gesetzt, werden diese alten Werte verwendet. Vor der Eingabe des Kommandos P muß der Kassettenrecorder auf Aufnahme gestellt werden.

Kommando: Q n,m,s (put ASCII)

n ... 1.Zeile
m ... letzte Zeile
s ... Dateiname

Die Textdatei kann auch als reine ASCII-Datei ohne Zeilennummerierung auf Band abgelegt werden. Es gelten die gleichen Bedingungen wie bei 'P'.

Kommando: G,,s (Get text)

s ... Dateiname

Die Textdatei mit dem Namen s soll vom Magnetband geladen werden. Ist s ein leerer String, wird die erste gefundene Textdatei geladen. Nach Kommandoaufruf muß der Recorder auf Wiedergabe gestellt werden. Es beginnt jetzt die Suche nach einer Datei mit dem angegebenen Namen, bzw. bei Leerstring wird die zuerst gefundene Textdatei geladen. Wenn die Datei gefunden wird, wird die Meldung 'LOADING filename' ausgegeben, ansonsten wird 'FOUND filename' angezeigt und die Suche wird fortgesetzt. Befindet sich bereits eine Textdatei im Speicher, wird die neu geladene Datei hinten angefügt, und es erfolgt eine neue Durchnumerierung der Zeilen beginnend mit 1 und der Schrittweite 1.

Kommando: V,,s (Verify)

s ... Dateiname

Mit diesem Kommando kann eine mit 'P' gerettete Textdatei auf Übereinstimmung mit der momentan im Speicher vorhandenen Datei überprüft werden (Probe, ob sich Datei später auch laden läßt).

Als Ergebnis wird 'VERIFIED' (Übereinstimmung) oder 'FAILED' (Fehler) ausgegeben.

Kommando: T n (Tape speed)

n ... Zahl >0

Mit 'T' kann die SAVE-Geschwindigkeit zwischen 1000 und 2000 Baud gewechselt werden. Fehlt das Argument n, wird auf 1000 Baud eingestellt. Wird n mit >0 angegeben, wird auf 2000 Baud eingestellt.

Kommando: O,,s (Objekt)

s ... Dateiname

Mit 'O' wird der bei der letzten Assemblierung entstandene Objektcode auf Band gerettet. Vor der Eingabe von 'O' muß der Recorder auf Aufnahme gestellt werden. Der gerettete Objektcode kann nicht mehr mit ASSMON1 geladen werden. Um diese Datei zu laden, muß man sich in ASSMON2 oder BASIC befinden.

## 2.4. Weitere Kommandos

Kommando: H (Help)

Mit 'H' können die in ASSMON1 möglichen Kommandos auf dem Bildschirm angezeigt werden.

Kommando: Z n,m (print text)

n ... 1.Zeile
m ... letzte Zeile

Mit 'Z' kann die Textdatei von der Zeile n bis zur Zeile m auf einem Drucker ausgegeben werden. Fehlen die Argumente n und m, wird die gesamte Datei ausgedruckt. Sollte kein Drucker angeschlossen sein, wird 'No Printer' auf dem Bildschirm ausgegeben. Mit beliebigem Tastendruck wird das Drucken gestoppt und mit ^ESC_ abgebrochen. Nach dem Druckstopp kann mit beliebiger Taste (außer ^ESC_) fortgesetzt werden.

Kommando: B (Bye)

Mit diesem Kommado wird zu BASIC zurückgesprungen. Um in ASSMON1 zurückzukehren, muß mit CALL die Ladeadresse von ASSMON1+4 für Warmstart oder Ladeadresse + 2 für Kaltstart aufgerufen werden.

Kommando: S,,d (Separator)

d ... Trennzeichen

Zum Trennen der Argumente in der Kommadozeile wird ',' verwendet. Mit 'S' kann aber auch jedes andere Zeichen dafür definiert werden. Dazu ist der String d einzugeben. Ist d länger als ein Zeichen, wird jeweils das erste Zeichen genommen. ^SPACE_ ist als Trennzeichen nicht zulässig.

Kommando: C (Current State)

Das Kommado 'C' bewirkt die Ausgabe der zuletzt definierten bzw. Standardargumente a und b für die zwei Zeilennummern und c und d für die Zeichenketten. Sinnvoll ist dieses Kommando immer dann, wenn die Argumente bei späteren Kommandos verwendet werden sollen, um deren Inhalt zu prüfen.

Kommando: X (text info)

Mit 'X' wird die Start- und Endadresse der Textdatei dezimal angezeigt. Das ist zur weiteren Verwendung des Textes von BASIC aus oder zur Überprüfung des noch freien Speichers nützlich.

Kommando: W (Width)

Das Kommando 'W' schaltet zwischen den Bildschirmmodi 1 und 2 um, so daß wahlweise 40 oder 80 Zeichen pro Zeile angezeigt werden.

Kommando: Y (print lenght)

'Y' legt die Anzahl der Zeilen pro Seite bei Druckerausgabe fest.

Kommando: U (Upper line)

Soll Quelltext gerettet oder gelöscht werden, interessiert oft die letzte verwendete Zeilennummer. Mit 'U' wird diese auf dem Bildschirm ausgegeben.

Kommando: ^CTRL_-^J_ (JUMP to ASSMON2)

Mit ^CTRL_-^J_ wird in das Programm 'ASSMON2' gesprungen, falls es geladen ist.

Kommando: A (Assemble)

Mit 'A' wird der im Speicher befindliche Quelltext assembliert. Dieses Kommando wird ausführlich im Abschnitt 3.8. beschrieben.

Kommando: R (Run)

'R' startet ein assembliertes Programm auf der Adresse, die im Quelltext mit 'ENT' festgelegt wurde (siehe Abschn. 3.6.6.).

# 3. A s s e m b l e r

Der Assembler ist der wichtigste Programmteil in 'ASSMON1'. Er übersetzt den Quelltext in den gewünschten Maschinencode, der vom U880 direkt verarbeitet werden kann. Verwendet wurde hierfür ein 2-Phasen-Assembler, der alle Standard-U880-Befehle, verschiedene Pseudo-Befehle und Assembler-Kommandos verarbeiten kann. Weitere Möglichkeiten bietet das bedingte Assemblieren, d.h. Teile des Quellprogramms werden nur unter bestimmten Bedingungen in Maschinencode umgesetzt.

Es sei darauf hingewiesen, daß es mehrere verschiedene Assemblersprachen für die U880 bzw. Z80 CPU gibt. Der hier beschriebene Assembler verarbeitet Quellprogramme in ZILOG-Mnemonik (wie z. B. die Assembler der Bürocomputer A5120/30 unter den Betriebssystemen UDOS und SCPx).

## 3.1. Die Arbeitsweise des Prozessors

Mit dem U880-Prozessor stehen dem Programmierer über 600 Operationscodes für arithmetische, logische, Programmorganisations-, Datentransfer- sowie Ein-/Ausgabe-Befehle zur Verfügung. Sämtliche Befehle eines abzuarbeitenden Maschinenprogramms stehen in externen Speicherbausteinen (dem sog. Hauptspeicher) des Rechners. Die Befehlsabläufe ("Befehlszyklen") sehen alle im Prinzip gleich aus (s. Abb. 3.1).

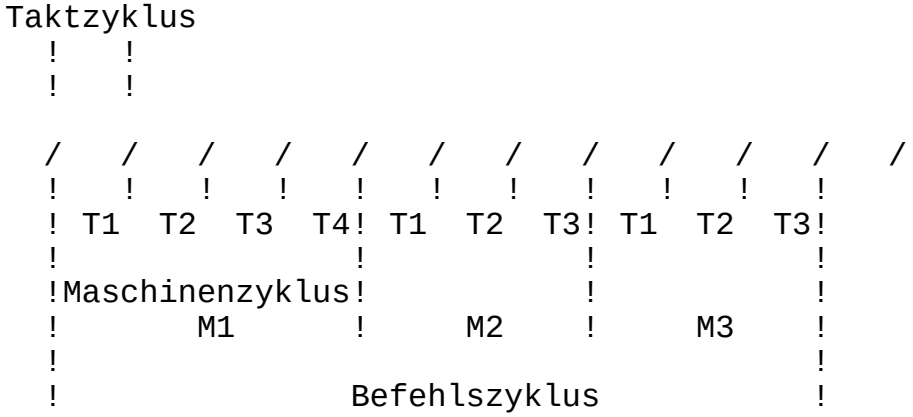



Abb. 3.1: Beispiel eines Befehlszyklus

Man ersieht aus Abb. 3.1, daß folgende Hierarchie besteht:
Die Befehlszyklen bestehen aus mehreren "Maschinenzyklen". Die Anzahl der Maschinenzyklen pro Befehl ist unterschiedlich (zwischen 1 und 6). Jeder Maschinenzyklus besteht seinerseits aus

mehreren "Taktzyklen" (3 bis 6 Taktzyklen). Die Dauer eines Taktzyklus ist durch die Frequenz des Taktgenerators der CPU gegeben. Beim KC compact beträgt die Taktfrequenz 4 MHz, d.h. die Dauer eines Taktzyklus beträgt 250 ns. Im Abschnitt 3.7 ist für die einzelnen Befehle der CPU die Gesamtanzahl der Taktzyklen jeweils angegeben. Man kann hieraus die Ausführungszeit eines Befehls ermitteln, was z. B. für Programme mit Zeitschleifen notwendig ist.

### 3.2. Die Registerstruktur des U880

Der U880-Prozessor hat die in Abb. 3.2 dargestellte Registerstruktur.


Abb. 3.2: Registerstruktur

Die einzelnen Register bestehen aus 16-Bit-Speichern, die entsprechend der in der Abbildung dargestellten Aufteilung wahlweise als 8-Bit-Register oder als 16-Bit-Registerpaare benutzt werden können. Der Zahlenbereich der 8-Bit-Register geht von 0 bis 255 (bzw. -128 bis +127 bei vorzeichenbehafteten Zahlen) und der Zahlenbereich der 16-Bit-Register von 0 bis 65535 (bzw.
-32768 bis +32767).
Nach dem Einschalten des Rechners wird immer der Hauptregistersatz angesprochen. Das Umschalten auf die Hintergrundregister geschieht durch 2 Austauschbefehle (s. Abschn. 3.7.2.) getrennt für Akkumulator/Flag und Allgemeinregister. Danach beziehen sich alle Befehle bis zum erneuten Umschalten auf die Hintergrundregister. Beim KC compact wird der Hintergrundregistersatz im System für die Interruptbehandlung verwendet und steht dem Nutzer deshalb nicht zur freien Verfügung.

## Akkumulator

Das 8-Bit-Register A dient bei arithmetischen und logischen Befehlen zur Aufnahme eines Operanden. Der andere Operand kommt aus einem anderen Register oder aus dem Speicher. Das Ergebnis der Operation wird wieder im Akkumulator abgelegt, und steht dort für eine weitere Verarbeitung zur Verfügung.

## Allgemeine Register

Diese können als 8-Bit-Register (B, C, D, E, H, L) oder als 16-Bit-Registerpaare (BC, DE, HL) benutzt werden. Die Register kann man frei als Zwischenspeicher verwenden, jedoch beziehen sich bestimmte Befehle auf einzelne Register. So dient das HL-Registerpaar der indirekten Speicheradressierung. Das DE-Registerpaar kann mit dem Registeraustauschbefehl über HL dem selben Zweck dienen. Bei einigen Befehlen werden beide Registerpaare als Adressenspeicher für Quell- und Zieladressen benutzt (z. B. LDIR).
Das Register B bzw. BC wird vorwiegend als Zählregister verwendet.

## Befehlszähler

Das 16-Bit-Register PC enthält den aktuellen Befehlszählerstand. Beim Einschalten des Rechners wird der Befehlszähler auf Null gesetzt. Bei Sprung- und Unterprogrammbefehlen wird er mit einem neuen Wert geladen, sonst wird er automatisch jeweils um die Befehlslänge erhöht.

## Stackpointer

Der Stackpointer SP enthält die 16-Bit-Adresse der aktuellen Spitze des Kellerspeichers der CPU. Der Kellerspeicher arbeitet nach dem Prinzip, daß die zuletzt gespeicherten Daten wieder als erste ausgegeben werden ("last in - first out"). Er dient vorwiegend zur Aufnahme der Rückkehradressen bei Unterprogramm-Aufrufen und Interruptroutinen. Außerdem kann er zum Ablegen (PUSH) und Wiedereinlesen (POP) von 16-Bit-Daten aus den Registern verwendet werden. Durch Setzen des Stackpointers im Initialisierungsprogramm des Rechners wird die Lage des für den Kellerspeicher zur Verfügung stehenden Teils des Operativ-Speichers (RAM-Bereich) festgelegt. Die Größe ist zunächst nicht begrenzt. Beim Programmerstellen ist aber zu beachten, daß für das jeweilige Programm ausreichend Kellerspeicherplätze zur Verfügung stehen müssen.
Der Stackpointer wird beim Abspeichern im Keller um 2 Byte verkleinert und beim Einlesen um 2 Byte erhöht. Er zeigt immer auf den zuletzt eingespeicherten Wert.
Ein Unterprogrammaufruf kellert den Befehlszählerstand ein, ein Rückkehrbefehl kellert den PC wieder aus. Außerdem können in den Unterprogrammen Daten mit PUSH und POP zwischengespeichert werden. Damit nicht versehentlich Befehlszähler und Daten verwechselt werden, ist bei der Verwendung von PUSH und POP größte Sorgfalt auf ein symmetrisches Ein- und Auskellern zu legen.

## Indexregister

Die Indexregister IX und IY werden zur indizierten Adressierung (s. Abschn. 3.5.) verwendet oder können als 16-Bit-Datenregister verwendet werden.
Bei der indizierten Adressierung kann auf einen Speicherbereich in einer Umgebung von -128 bis +127 um den im Register gespeicherten Adressenwert direkt zugegriffen werden.

## Interruptvektorregister

Dieses Register beinhaltet den höherwertigen Adreßteil der Tabelle für die Interruptroutinen (s. dazu Abschn. 3.4.).

## Refreshregister

Dieses 7-Bit-Register wird bei jeder Befehlsverarbeitung um 1 erhöht. Es dient zum Auffrischen der Inhalte der dynamischen RAM-Speicher und ist für den Programmierer kaum von Bedeutung.

### 3.3. Flag-Bits

Die CPU verfügt über zwei Status-(Flag-)Register (s. Abb. 3.2). Durch Veränderung einzelner Bits wird über die Art des Ergebnisses der letzten Prozessoroperation Auskunft gegeben. Diese Auskunft wird hauptsächlich dazu benutzt, bedingte Sprünge vorzunehmen, d. h. je nach Ergebnis der Prüfung eines dieser Bedingungsbits die eine oder aber eine andere Aktion durchzuführen. Die Stellung der einzelnen Bits innerhalb des Flag-Registers F zeigt folgende Tabelle:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | Z | X | H | X | P/V | N | C |

Hierbei bedeuten:

- C - Übertragsbit (Carry-Flag)
- N - Additions-/Subtraktionsbit (Add/Subtract-Flag)
- P/V - Paritäts-/Überlaufbit (Parity/Overflow-Flag)
- H - Halb-Byte-Überlaufbit (Half-Carry-Flag)
- Z - Nullbit (Zero-Flag)
- S - Vorzeichenbit (Sign-Flag)
- X - nicht verwendet

Das Carry-Flag wird gesetzt (=1), wenn bei der Addition zweier Operanden ein Übertrag von Bit 7 entsteht, sowie wenn bei der Subtraktion ein Bit geborgt werden muß (das Ergebnis negativ wird). Darüber hinaus fungiert das Carry-Flag als Bit-Speicher bei Verschiebe- und Rotationsbefehlen.
Das Zero-Flag wird gesetzt, wenn das Ergebnis einer Operation den Wert Null ergibt. Bei Einzelbitbefehlen dient es zur Übergabe ausgelesener Bits.
Die Funktion des P/V-Flags hängt von der verwendeten Operation ab. Bei logischen und Verschiebebefehlen wird die Parität des Ergebnisses angezeigt (gerade Parität: P/V = 1; ungerade Parität P/V = 0). Bei arithmetischen Befehlen wird das P/V-Flag als Vorzeichen-Überlaufkennzeichnung benutzt; es wird z. B. gesetzt, wenn das Ergebnis zweier Vorzeichenzahlen außerhalb des zulässigen Bereiches von -128 bis +127 liegt.
Das Sign-Flag zeigt nach Additionen und Subtraktionen an, ob das Ergebnis positiv ist (S = 0) oder negativ (S = 1).
Das Half-Carry-Flag wirkt wie das Carry-Flag, jedoch wird der Übertrag von Bit 3 auf Bit 4 angezeigt.
Mit dem Add/Subtract-Flag wird gekennzeichnet, ob als letzter Befehl eine Addition (N = 0) oder eine Subtraktion (N = 1) durchgeführt wurde.
Die genaue Reaktion der Flags auf die einzelnen Befehle kann der Befehlsliste (Anhang A) entnommen werden.

### 3.4. Interruptsystem

Soll der Rechner auf externe Ereignisse reagieren, so hat man die Möglichkeit, entweder den betreffenden Eingabekanal ständig abzufragen oder das laufende Programm mittels Interrupt zu unterbrechen, und nach Reaktion auf das Eingabesignal (Interruptprogramm) das ursprüngliche Programm fortzusetzen. Die Tastaturabfrage im KC compact wird z.B. über Interrupt realisiert.
Die U880-CPU hat zwei getrennte Signaleingänge zur Auslösung von Interrupts:

NMI - nichtmaskierbarer Interrupt höchster Priorität
INT - maskierbarer Interrupt (kann in 3 verschiedenen Interrupt-Modi betrieben werden)

#### 3.4.1. Nichtmaskierbarer Interrupt (NMI)

Die NMI-Signalzuführung kann nicht gesperrt werden. Ein NMI-Signal führt also in jedem Fall zu einer Unterbrechung des laufenden Programms und zu einem erzwungenen Unterprogrammsprung zur Speicheradresse 0066H. An dieser Stelle muß die Interruptbehandlungsroutine vom Programmierer eingetragen sein. Die Interruptroutine muß am Ende mit einem Rücksprung ins unterbrochene Programm (mit dem Befehl RETN) abgeschlossen werden.

#### 3.4.2. Maskierbarer Interrupt (INT)

Die INT-Signalzuführung kann mit Hilfe der Befehle

EI - eingeschaltet (enable interrupt) und mittels
DI - ausgeschaltet (disable interrupt)

werden (maskieren).

Ist das Interruptsystem nicht freigegeben (DI), so werden INT-Anforderungen ignoriert.
Die Steuerung des INT-Eingangs der CPU erfolgt über zwei Merker IFF1 und IFF2 (Interruptflipflops). Der Befehl EI setzt beide auf 1 und DI beide auf 0. IFF2 dient als Merker der Stellung von IFF1 bei der NMI-Behandlung (während der NMI-Behandlung, d. h. bis RETN, ist kein INT möglich).
Wird nun ein anstehendes INT-Signal erkannt, so führt die CPU hardwaremäßig eine DI-Operation aus, d. h. IFF1 und IFF2 werden auf 0 gesetzt. Sollen weiterhin Interrupts zugelassen werden, so ist spätestens vor Verlassen der Interruptbehandlungsroutine mit RETI der Befehl EI zu programmieren.
Der maskierbare Interrupt INT kann in 3 Arbeitsweisen betrieben werden, die mit den Befehlen IM 0, IM 1 bzw. IM 2 eingeschaltet werden. Der KC compact arbeitet im Interrupt-Mode IM 1.

Interrupt-Mode IM 0
---
Wird ein INT-Signal akzeptiert, so wird gleichzeitig ein auf dem Datenbus vom Interruptauslöser (peripherer Baustein) bereitzustellender 1-Byte-Befehl eingelesen und anschließend ausgeführt. Im Normalfall werden dazu die Restart-Befehle

RST n (n = 0, 8, #10, #18, #20, #28, #30, #38)

verwendet, die einen Unterprogrammaufruf zur Speicheradresse n bewirken, an der die Interruptbehandlungsroutine beginnen muß.

Interrupt-Mode IM 1
---
Beim Akzeptieren des INT-Signals wird unabhängig von den anderen Eingängen ein Unterprogrammsprung zur Adresse #38 durchgeführt (ähnlich wie NMI). Der KC compact arbeitet in diesem Modus. Genauere Aussagen zum Interruptverhalten des KC compact lesen Sie bitte in /1/ nach.

Interrupt-Mode IM 2
---
Diese Betriebsart der CPU ist die leistungsfähigste und gestattet die individuelle Behandlung unterschiedlicher peripherer Bausteine. Die Interruptbehandlung läuft nach folgendem Schema ab:
In der CPU wird aus dem Wert des Interruptregisters I und dem vom peripheren Baustein auf dem Datenbus bereitzustellenden Interruptvektor IV eine 16-Bit-Adresse gebildet. Das Interruptregister I bildet dabei den höherwertigen Adressteil und der Interruptvektor IV den niederwertigen. Von dieser und der folgenden Speicheradresse wird die eigentliche Startadresse der Interruptbehandlungsroutine ausgelesen und ein Unterprogrammsprung dorthin durchgeführt. Die Interruptbehandlungsroutine muß mit RETI beendet werden, wobei ggf. zuvor das Interruptsystem mit EI einzuschalten ist.
Mit dem Interruptregister I wird also die Lage der Tabelle der Startadressen für die Interruptbehandlungsroutinen im Speicher festgelegt. Durch den vom peripheren Baustein bereitgestellten Interruptvektor IV, der geradzahlig sein muß, wird eine der 128 möglichen Startadressen ausgewählt, an der die Interruptroutine beginnt.

## 3.5. Adressierungsarten

Der Befehlssatz des Prozessors beinhaltet 6 verschiedene Adressierungsarten zur Bereitstellung von Register-, Speicher- oder Ein/Ausgabe-Adressen für zu spezifizierende Datenwörter:

Direkte Adressierung
---
Der Operationscode beinhaltet vollständig die entsprechende Adresse.

```
z.B.:   LD      A,B
        LD      (#0200),HL
```

Implizite Adressierung
---
Der Operationscode bezieht sich fest auf bestimmte Speicherplätze oder Register.

```
z.B.:   EXX
        SCF
```

Unmittelbare Adressierung
---
Dem Operationscode folgt unmittelbar eine 8- oder 16-Bit-Konstante im Speicher.

```
z.B.:   LD      A,6
        XOR     #20
```

Indirekte Adressierung
---
Die 16-Bit-Adresse befindet sich in einem Registerpaar der CPU. Der Befehl bezieht sich indirekt auf diese Adresse.

```
z.B.:   LD      A,(HL)
        LDIR
```

Indizierte Adressierung
---
Der Operationscode beinhaltet ein Datenbyte (zwischen -128 und +127), das zum Inhalt des Doppelregisters IX oder IY addiert die vollständige Adresse ergibt.

```
z.B.:   LD      A,(IX+6)
```

### 3.6. Syntax der Assemblersprache

Der hier beschriebene Assembler verarbeitet die ZILOG-Mnemonik (spezielle U880-Assembler-Mnemonik). Dabei gelten allerdings einige Einschränkungen und Besonderheiten, auf die in diesem Abschnitt an entsprechender Stelle eingegangen wird.
Ein Assemblerprogramm (Quellprogramm) besteht, wie schon erwähnt,

aus einer Folge von Anweisungen (sog. Statements), die zusammen das Anwenderprogramm ergeben. Jede der Quellprogrammzeilen ist aus einem Markenfeld, einem Operationscodefeld (mnemonische Ausdrücke), einem Operandenfeld und einem Kommentarfeld aufgebaut.

Beispiel für eine Quellprogrammzeile:

| Zeilen-Nr. | Markenfeld | Operations-codefeld | Operanden-feld | Kommentar-feld |
|---|---|---|---|---|
| 10 | START: | LD | A,6 | ;Akku laden |

Je nach Art der Befehle können oder müssen einzelne dieser Felder wegfallen. Die einzelnen Felder müssen durch eine beliebige Anzahl von Leerzeichen oder Tabulatoren voneinander getrennt werden.
Im Quellprogramm wird außer in Zeichenketten nicht zwischen Groß- und Kleinbuchstaben unterschieden.

Die Zeile wird vom Assembler wie folgt verarbeitet:
Als erstes wird immer das 1. Zeichen der Zeile untersucht.

Folgende Fälle sind möglich:

| 1. Zeichen | Bedeutung |
|---|---|
| ; | die ganze Zeile ist Kommentar |
| * | das (die) folgende(n) Zeichen wird (werden) als Assembler-Kommando erwartet |
| Zeilenende-markierung | Leerzeile --> wird ignoriert |
| Leerzeichen | wird ignoriert, das nächste Zeichen wird untersucht |
| Tabulator | wird ignoriert, das nächste Zeichen wird untersucht |
| ASCII | Die Zeile wird nach einem nachfolgenden Leer-,Tabulatorzeichen oder ':' untersucht. Bei ':' wird die Zeichenkette als Marke erkannt, bei Leer- oder Tabulatorzeichen versucht der Assembler, die Zeichenkette als Assemblerbefehl zu deuten. |

Nach der Verarbeitung der ersten Zeichenkette versucht der Assembler, die nächste Zeichenkette als Operation, falls vorher eine Marke erkannt wurde, oder als Operanden, falls vorher eine Operation erkannt wurde, die Operanden benötigt, zu deuten. Nach den Operanden wird dann ein Kommentar oder eine Zeilenendemarkierung erwartet.

### 3.6.1. Zeilennummern

Vor jeder Quellprogrammzeile steht ähnlich wie in BASIC eine Zeilennummer. Diese dienen der eindeutigen Identifizierung von Quelltextzeilen, da ja der Editor zeilenorientiert arbeitet. Für die Zeilennummern ist der Zahlenbereich von 1 bis 32767 zulässig.

### 3.6.2. Marken

Marken sind symbolische Bezugspunkte innerhalb des Programms. Sie werden verwendet, um in einer anderen Anweisung auf den momentanen Befehlszählerstand, auf eine andere Marke oder auf eine Konstante Bezug nehmen zu können. Eine Marke muß in der ersten Spalte der Programmzeile beginnen und kann bis zu 16 Zeichen lang sein, wobei nur die ersten 6 Zeichen signifikant sind. Das erste Zeichen muß ein Buchstabe sein, als weitere sind Buchstaben, Ziffern, und die Zeichen ^, _, -, ., ' zulässig. Marken müssen mit Doppelpunkt abgeschlossen werden.
Ein Markenname (Bezeichner) darf selbst kein reserviertes Wort sein (s. Anhang E). Zulässige Bezeichner sind z.B.:

```
LOOP:
loop:
Q^14_:
LDIR:
EINS-ZWEI-DREI:
J':
```

### 3.6.3. Operationscodes

Im Operationscodefeld steht eine der ZILOG Maschinenbefehlsmnemoniken oder eine der Assembler-Pseudoanweisungen.

### 3.6.4. Operanden

Je nach Art des Operationscodes muß das Operandenfeld entweder leer sein, oder es enthält einen oder zwei (durch ein Komma getrennte) Operanden, die eine Adresse (Speicher, Register oder Ein/Ausgabe-Kanal), eine Konstante oder eine Flag-Bedingung repräsentieren. Die Operanden ergänzen die jeweiligen Anweisungen durch eine Information darüber, mit welchen Parametern die Operation durchzuführen ist.
Es sind folgende Schlüsselwörter für die Operandenfelder reserviert

- Die Namen der internen Register der CPU, die jeweils den 8-Bit-Inhalt eines dieser Register ansprechen. Die Registernamen sind: A, B, C, D, E, F, H, L, I und R.
- Die Namen der Doppelregister und Registerpaare. Doppelregister

sind die 16-Bit-Register IX, IY, SP und PC. Über die Registerpaar-Bezeichnungen AF, BC, DE und HL lassen sich die oben bezeichneten 8-Bit-Register der CPU paarweise (als 16- Bit-Wörter) vom Assemblerbefehl ansprechen.

- Die in der CPU integrierten Zweitregister werden in der Assemblersprache mit einem Hochkomma gekennzeichnet. Die Namen sind AF', BC', DE' und HL'. Von diesen Namen ist jedoch lediglich die Kombination AF' als Operand zulässig (in der Anweisung EX AF,AF').
- Der Zustand der 4 vom Assemblerprogramm testbaren Bedingungs-Bits wird in der Flag-Bedingung wie folgt notiert:

| Bedingungs-Bit-Bezeichnung | Bedingung erfüllt (Flag-Bit = 1) | Bedingung nicht erfüllt (Flag-Bit = 0) |
|---|---|---|
| Übertrags-Bit (Carry) | C | NC |
| Null-Bit (Zero) | Z | NZ |
| Vorzeichen-Bit (Sign) | M (negativ) | P (positiv) |
| Parität-/Überlauf-Bit (Parity/Overflow) | PE (gerade-even) | PO (ungerade-odd) |

(vgl. Abschn. 3.3.)

Operanden können auch Ausdrücke sein. Sie bestehen aus einem oder mehreren Termen (Formelteilen), welche durch einen Operator getrennt sind.
Für Terme sind folgende Schreibweisen zulässig:

| | |
|---|---|
| Dezimale Konstante | z.B. 12102 |
| Hexadezimale Konstante | z.B. #FA02 |
| Binäre Konstante | z.B. %10101101 |
| Zeichenkonstante | z.B. "H" |
| Namen | z.B. LOOP23 |

Zusätzlich darf "$" für den momentanen Befehlszählerstand verwendet werden.

Folgende Operatoren sind zulässig:

| | |
|---|---|
| + | Addition |
| - | Subtraktion |
| & | logisches UND |

|  |  |
|---|---|
|  | logisches ODER |
| ! | logisches XOR |
| * | Multiplikation |
| / | Division |
| ? | MOD-Funktion (a?b = a-(a/b)*b), d.h. Rest einer Division |

### 3.6.5. Kommentare

Kommentare dienen Dokumentationszwecken und der Erhöhung der Übersichtlichkeit von Quellprogrammen. Sie sind kein funktioneller Bestandteil des Programms und werden beim Assemblervorgang übersprungen.
Ein Kommentar darf in jeder Spalte der Programmzeile beginnen und er endet mit dem Zeilenende. Das erste Zeichen eines jeden Kommentars muß ein Semikolon ';' sein.

### 3.6.6. Pseudo-Befehle

Pseudooperationen sind Anweisungen an den Assembler zur Steuerung der Übersetzung des Quellprogramms. Es gibt daher zu Pseudooanweisungen keinen U880-Maschinencode. Sie sind aber wie ausführbare Anweisungen aufgebaut; können also mit einer Marke versehen und mit einem Kommentar beendet werden.
Normalerweise beginnt ein Assemblerprogramm mit einer ORG-Pseudoanweisung. Sie legt fest, auf welche Adresse der Beginn des Maschinenprogramms gelegt wird. Wird sie weggelassen, so legt der Assembler den Anfang auf den ersten freien Speicherplatz hinter dem Quelltext und der Markentabelle fest. Als Operand kann eine Marke, eine hexadezimale Zahl oder eine Verknüpfung stehen.
Der Assembler verarbeitet folgende Pseudoanweisungen:

```
     ORG  Ausdruck
```

Setzt den Adreßzähler auf den Wert des Ausdrucks. Üblicherweise wird damit der Speicherbeginn eines Maschinenprogramms definiert. Würde beim Assemblieren durch die ORG-Anweisung 'ASSMON1', die Quelldatei oder die Markentabelle überschrieben, wird die Assemblierung abgebrochen und die Fehlermeldung 'Bad Org!' ausgegeben (siehe auch Abschnitt 3.8.1.).

```
Marke     EQU  Ausdruck
```

Weist der Marke den Wert des Ausdrucks zu. Damit kann im Assemblerprogramm mit symbolischen Bezeichnungen anstelle von Konstanten gearbeitet werden. Der Ausdruck darf kein Bezeichner sein, dem nicht nicht vorher schon ein Wert zugewiesen wurde.

```
     DEFB Ausdruck
```

"Definiere Byte" - legt den durch den Ausdruck festgelegten Byte-

Wert auf die nächste Speicherstelle (z.B. DEFB 25).

DEFW Ausdruck

"Definiere Wort" - legt den durch den Ausdruck festgelegten 2-Byte-Wert (Wort) auf die nächsten beiden Speicherstellen. Dabei wird zunächst das niederwertige Byte und danach das höherwertige Byte abgelegt.

DEFM "Text"

Legt die ASCII-Werte der durch 'Text' definierten Zeichenkette im Speicher ab. Die Zeichenkette muß in Hochkommas eingeschlossen sein. Z.B.

DEFM "HALLO"

legt die Bytefolge #48, #41, #4C, #4C, #4F in den Speicher.

DEFS Ausdruck

Reserviert einen Speicherbereich, dessen Länge durch den Ausdruck festgelegt wird, bzw. erhöht den Befehlszähler durch den Wert des Ausdrucks. Der reservierte Bereich wird mit Nullen gelöscht.

ENT Ausdruck

Legt die Startadresse des Maschinenprogramms fest. Diese Anweisung ist notwendig, um das übersetzte Programm zu Testzwecken von 'ASSMON1' aus mit 'R' zu starten. Fehlt die ENT-Anweisung, wird kein Ersatzwert eingetragen.

### 3.6.7. Bedingte Pseudo-Befehle

Bedingte Pseudo-Befehle ermöglichen es, Teile des Quellprogramms nur unter bestimmten Bedingungen assemblieren zu lassen, bzw. eine Verzweigung zu erreichen. Folgende Befehle sind möglich:

IF Ausdruck

Ergibt der Ausdruck den Wert Null, wird der Quelltext vom Assembler bis zu 'ELSE' oder 'END' ignoriert. Ist der Wert des Ausdrucks ungleich Null, wird die Assemblierung normal fortgesetzt.

ELSE

Durch 'ELSE' wird die Assemblierung fortgesetzt oder aufgehoben, je nachdem, ob der Ausdruck hinter 'IF' den Wert Null ergab oder nicht (siehe IF THEN ELSE in BASIC).

END

END schaltet das Ignorieren von Quelltext ab.

Es ist zu beachten, daß 'IF ELSE END'-Konstruktionen nicht geschachtelt werden dürfen.

### 3.6.8. Assemblerkommandos

Genau wie Pseudo-Befehle werden Assemblerkommandos nicht in Maschinencode übersetzt. Sie beeinflussen den Ausdruck der Assemblerliste, gestatten es, Quelltext vom Band zu assemblieren oder den assemblierten Maschinencode gleich während der Assemblierung auf Band auszugeben. Das ist bei langen Quelltexten nützlich, oder wenn mehrere Quellen zusammen assembliert werden sollen. Assemblerkommandos werden durch ein vorangestelltes '*' in der Zeile gekennzeichnet. Der Buchstabe nach dem '*' bestimmt die Funktion und muß groß geschrieben werden. Einige Kommandos erwarten nach dem Buchstaben noch bestimmte Parameter, z.B. '+', '-' oder Zeichenketten. Außer '*F' werden alle Assemblerkommandos erst im 2. Assembler-Pass ausgewertet.

Folgende Kommandos sind möglich:

*L-

Beginnend mit dieser Zeile wird die Auflistung unterdrückt.

*L+

Beginnend mit dieser Zeile erfolgt wieder eine Auflistung.

*D-

Die Ausgabe des Befehlszählers erfolgt ab dieser Zeile hexadezimal.

*D+

Die Ausgabe des Befehlszählers erfolgt ab dieser Zeile dezimal.

*E

Veranlaßt die Ausgabe von drei Leerzeilen auf dem Bildschirm, bzw. Blattvorschub bei Druckerausgabe.

*Hs

Der String s wird als Kopfzeile definiert und wird nach jedem

'*E' ausgegeben.
'*H' führt automatisch '*E' aus.

*S

Die Assemblierung (Auflistung) wird an dieser Stelle gestoppt. Durch Drücken einer beliebigen Taste wird die Assemblierung fortgesetzt.
'*S' unterbricht das Assemblieren nicht, wenn '*L-' aktiv ist.

*T+s

Dieses Kommando bewirkt, daß der Objektcode während der Assemblierung auf Band ausgegeben wird. Mit s wird der Name der abzulegenden Maschinencode-Datei festgelegt.
Durch '*T-', ORG-Anweisung oder Abschluß der Assemblierung wird die Bandausgabe beendet.

*T-

Beendet die Bandausgabe, die mit '*T+' begonnen wurde.

*Fs

Mit diesem wirkungsvollen Kommando kann Quelltext vom Band assembliert werden. Es besteht also die Möglichkeit, weit größere Quellprogramme (zerteilt in mehrere Stücke) zu assemblieren, als es die Größe des Quelltextspeichers (RAM) zuläßt. Der Name s der Datei, die eingefügt werden soll, darf bis zu acht Zeichen lang sein. Fehlt die Namensangabe, wird die erste auf Band gefundene Datei eingefügt. Bei jedem gefundenen '*F' wird der Bediener zum Einschalten des Recorders und dem Betätigen einer Taste aufgefordert. Da die Datei bei beiden Assembler-Pässen eingelesen werden muß, muß nach dem ersten Pass zurückgespult werden oder die Datei muß sich zweimal hintereinander auf dem Band befinden. Wird ein Recorder ohne Motorschaltung verwendet, muß gegebenenfalls nach einem gelesenen Block einer Datei die Pausentaste am Recorder gedrückt werden, bis der gelesene Block assembliert wurde. Die so verwendeten Dateien sind mit dem Kommando 'P' des Editors auf Band zu speichern.

### 3.6.9. Markentabelle

Im ersten Assemblerlauf wird eine sogenannte Symbol- oder Markentabelle erstellt. Beim ersten Auftreten eines Namens (Marke) wird er zusammen mit dem Befehlszählerstand oder dem Wort in der 'EQU'-Anweisung und zwei Informationsbytes in der Tabelle abgelegt. Die beiden Informationsbytes dienen der alphabetischen Einordnung.
Die Länge des Eintrags in der Tabelle hängt von der Namenslänge

ab (8 bis 13 Bytes). Ist die Assemblierung beendet, wird die Größe der benötigten Markentabelle ausgegeben.
Wurde ein Name nicht definiert oder wurde ihm kein Wert zugewiesen, wird am Ende die Meldung '*WARNING* symbol absent' ausgeben.

### 3.7. Befehlssatz des Assemblers

In diesem Abschnitt wird der syntaktische Aufbau und die Wirkungsweise der einzelnen Assemblerbefehle beschrieben. Der U880-Prozessor verfügt über einen sehr umfangreichen Befehlssatz, der nicht nur 8-Bit-, sondern auch 16-Bit-Befehle umfaßt. Mit der großen Befehlsliste stehen dem Programmierer meist verschiedene Möglichkeiten zur Verfügung, ein und dasselbe Problem mehr oder weniger elegant zu lösen. Viel hängt dabei von der Übung und den Erfahrungen des Programmierers ab.
Besonders zu beachten ist, daß z. B. verschiedene Assembler-befehle nur für spezielle Operanden gelten. Die Gültigkeit des verwendeten Befehls ist also anhand der in diesem Abschnitt angegebenen Übersicht über alle erlaubten Befehle sorgfältig zu überprüfen.

Der Befehlssatz läßt sich in folgende Gruppen einteilen:

- Lade- und Austauschbefehle
- Blocktransfer- und Blocksuchbefehle
- Arithmetik- und Logikbefehle
- Programmverzweigungsbefehle
- Unterprogrammbefehle
- Rotations- und Verschiebebefehle
- Einzelbitbefehle
- Steuerbefehle
- Ein- und Ausgabebefehle

Die Ladebefehle transportieren Daten zwischen den Registern oder zwischen Registern und dem Speicher. Registeraustauschbefehle erschließen dem Programmierer die Hintergrundregister der CPU, Blocktransferbefehle transportieren ganze Datenblöcke zwischen verschiedenen Speicherbereichen. Mit einem Blocksuchbefehl kann man einen Speicherbereich nach einem Datenbyte durchmustern.
Die Arithmetik- und Logikbefehle arbeiten mit einem Akkumulator (A, HL, IX oder IY) als ersten Operanden und einem Register, Speicherstelle oder Konstanten als zweiten Operanden. Das Ergebnis der Operation (z. B. einer Addition) steht wieder im Akkumulator und es werden die entsprechenden Flag-Bits gesetzt.
Die Programmverzweigungsbefehle gestatten es, in Abhängigkeit von den Flag-Bits Sprünge (relative oder absolute) zu anderen Programmteilen durchzuführen.
Im Unterschied zu den Sprungbefehlen bieten die Unterprogramm-

befehle die Möglichkeit, nach Abarbeitung des aufgerufenen Programmstücks wieder das ursprüngliche Programm fortzusetzen.
Die Rotations-, Verschiebe- und Einzelbitbefehle verwendet man, um den Inhalt einzelner Bits der Register oder von Speicherstellen abzufragen oder zu verändern.
Die Steuerbefehle dienen zur Beeinflussung des Interruptsystems der CPU. Mit den Ein- und Ausgabebefehlen kann man spezielle Toradressen zur Kommunikation mit externen Bausteinen (PIO, SIO, CTC) ansprechen und Daten ein- oder ausgeben.
Der U880-Prozessor hat variable Befehlswortlänge (1 bis 4 Byte). Die Byteanzahl und die Kodierung der einzelnen Befehle kann man der Befehlscode-Tabelle im Anhang A entnehmen. Der Befehlscode erscheint auch im Listing des Assemblers.
In den Abschnitten 3.7.1 bis 3.7.12 werden alle Assemblerbefehle und deren Wirkungsweise erläutert. Dabei werden die folgenden Abkürzungen verwendet:

Abkürzungsverzeichnis:
---

| | |
|---|---|
| r, r' | - Einfachregister A, B, C, D, E, H, L |
| dd | - Doppelregister BC, DE, HL, SP |
| qq | - Doppelregister AF, BC, DE, HL |
| pp | - Doppelregister BC, DE, SP |
| n | - 8-Bit-Konstante |
| nn | - 16-Bit-Konstante, Adresse |
| d | - Verschiebung bei Adressierung über Indexregister, im Bereich -128 = d = +127 |
| b | - Bit, das in den Einzelbitbefehlen behandelt werden soll, 0 = b = 7 |
| (HL) | - Inhalt des durch HL adressierten Speicherplatzes. Register L beinhaltet dabei die niederwertigen 8 Bits und Register H die höherwertigen 8 Bits der Adresse des Speicherplatzes. |
| p | - Einer der Werte #00,#08,#10,#18,#20,#28,#30,#38 |
| CY | - Carry-Flag |
| T | - Anzahl der Taktzyklen des Befehls |

Bei Befehlen, die Flag-Bedingungen für Programmsprünge auswerten sind 2 Taktzyklen angegeben. Dabei bezieht sich die zweite Zahl auf den Fall, daß kein Sprung durchgeführt und mit dem nächsten im Speicher stehenden Befehl weitergearbeitet wird.
T*250 Nanosekunden ist die Befehlsausführungszeit auf dem KC compact.

Flag-Beeinflussung der Befehle (s. 3.3.):
---
In der letzten Spalte der folgenden Befehlsübersicht ist für die einzelnen Flag-Bits C, N, P/V, H, Z, S deren Wert nach der Ausführung des Befehls angegeben. Dabei bedeutet:

1 - gesetzt (=1)
0 - zurückgesetzt (=0)
- - unverändert
* - entsprechend dem Ergebnis der Operation,d. h.:
gesetzt wenn erfüllt, zurückgesetzt wenn nicht erfüllt
x - unbestimmt
V - Overflow-Funktion
P - Parity-Funktion
F - Inhalt des Interrupt-Flip-Flops IFF2 (vgl. Abschn 3.4.2)

### 3.7.1. 8-Bit-Ladebefehle

Die Ladebefehle transportieren 8-Bit-Daten intern zwischen Registern oder zwischen Registern und dem Speicher. Dabei steht im Operandenfeld als erstes der Zielspeicherplatz und danach (durch Komma getrennt) der Quellspeicherplatz. Der Inhalt des Quellspeicherplatzes wird bei diesen Befehlen nicht verändert.

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| LD r,r' | 4 | Inhalt des Registers r' wird in das Register r umgespeichert | ------ |
| LD r,n | 7 | Die 8-Bit-Konstante n wird in das Register r geladen | ------ |
| LD r,(HL) | 7 | Inhalt des durch Registerpaar HL adressierten Speicherplatzes wird in das Register r geladen. | ------ |
| LD r,(IX+d) | 19 | Inhalt des durch Register IX (bzw. IY) plus Verschiebung d adressierten Speicherplatzes wird in Register r geladen | ------ |
| LD r,(IY+d) | 19 | | ------ |
| LD (HL),r | 7 | Transportiert ein Byte aus Register r auf den durch Registerpaar HL adressierten Speicherplatz. | ------ |
| LD (IX+d),r | 19 | Bringt Daten aus dem Register r an den Speicherplatz, dessen Adresse durch den Inhalt des IX (bzw. IY)-Registers plus Verschiebung d spezifiziert ist | ------ |
| LD (IY+d),r | 19 | | ------ |
| LD (HL),n | 10 | Bewirkt den Transport der Konstanten n an den durch Registerpaar HL adressierten Speicherplatz. | ------ |

| | | | | |
|---|---|---|---|---|
| LD | (IX+d),n | 19 | Bewirkt den Transport der Konstanten n an den Speicherplatz, dessen Adresse durch den Inhalt des IX (bzw. IY)-Register plus Verschiebung d spezifiziert ist | ------ |
| LD | (IY+d),n | 19 | | ------ |
| LD | A,(BC) | 7 | Der Inhalt des durch Registerpaar BC adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen | ------ |
| LD | A,(DE) | 7 | Der Inhalt des durch Registerpaar DE adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen | ------ |
| LD | A,(nn) | 13 | Der Inhalt des Speicherplatzes nn wird in den Akkumulator (A-Register) geladen | ------ |

| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|---|
| LD | (nn),A | 13 | Der Inhalt des Akkumulators (A-Register) wird auf den Speicherplatz nn geladen | ------ |
| LD | (BC),A | 7 | Der Inhalt des Akkumulators (A-Register) wird auf den Speicherplatz geladen, dessen Adresse im Registerpaar BC steht | ------ |
| LD | (DE),A | 7 | Der Inhalt des Akkumulators (A-Register) wird auf den Speicherplatz geladen, dessen Adresse im Registerpaar DE steht | ------ |
| LD | A,I | 9 | Der Inhalt des Interruptregisters I wird in den Akkumulator (A-Register) geladen | **0F0- |
| LD | A,R | 9 | Der Inhalt des Refresh-Registers I wird in den Akkumulator (A-Register) geladen | **0F0- |
| LD | I,A | 9 | Der Inhalt des Akkumulators (A-Register) wird in das Interruptregister I geladen | ------ |
| LD | R,A | 9 | Der Inhalt des Akkumulators (A-Register) wird in das Refresh-Register R geladen | ------ |

### 3.7.2. 16-Bit-Ladebefehle

Die 16-Bit-Ladebefehle transportieren 16-Bit-Daten intern zwischen den Registern oder zwischen Registern und dem Speicher. Der Inhalt des Quellspeichers wird dabei nicht verändert.

Spezielle 16-Bit-Befehle sind die PUSH- und POP-Befehle. Mit Ihnen werden 16-Bit-Daten aus Doppelregistern in den Kellerspeicher gebracht bzw. zurück in die Doppelregister geholt. Man verwendet sie häufig zum Retten von Registerinhalten z. B. in Unterprogrammen:

```
UP:     PUSH    HL
        PUSH    DE
        PUSH    BC
          ...       ) Unterprogramm-
          ...       ) Befehle
          ...       )
        POP     BC
        POP     DE
        POP     HL
        RET
```

Nach Beendigung des Unterprogramms besitzen die Register BC, DE, HL die gleichen Inhalte wie vor dem Aufruf.

Zu beachten ist, daß 16-Bit-Daten im Speicher auf 2 Byte mit aufeinanderfolgenden Adressen nn und nn+1 abgespeichert werden. Die unteren 8 Bit (niederwertiger Teil) stehen auf der Adresse nn und die oberen 8 Bit (höherwertiger Teil) auf der Adresse nn+1. Alle 16-Bit-Wörter (auch in Maschinencodes) werden vom Prozessor grundsätzlich auf diese Weise (erst niederwertiger, dann höherwertiger Teil) abgespeichert.

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| LD dd,nn | 10 | Die 16-Bit-Konstante nn wird in das Doppelregister dd geladen | |
| LD IX,nn<br>LD IY,nn | 14 | Die 16-Bit-Konstante nn wird in das Indexregister IX (bzw. IY) geladen | |
| LD HL,(nn) | 16 | Inhalt der Speicherplätze nn und nn+1 wird in das Doppelregister HL geladen<br>Inhalt von nn+1 --> Register H<br>Inhalt von nn --> Register L | |
| LD pp,(nn) | 20 | Inhalt der Speicherplätze nn und nn+1 wird in das Doppelregister pp geladen<br>Inhalt von nn+1 --> höherwertiges Reg<br>Inhalt von nn --> niederwertiges Reg | |
| LD IX,(nn)<br>LD IY,(nn) | 20 | Inhalt der Speicherplätze nn und nn+1 wird in das Indexregister IX (bzw. IY) geladen | |

```
            !  ! Inhalt von nn+1  -->  höherwert. Teil   !
            !  ! Inhalt von nn    -->  niederwert. Teil  !
            !  !                                         !
LD   (nn),HL!16! Inhalt des Doppelregisters HL wird auf  !------
            !  ! die Adressen nn und nn+1 transportiert  !
            !  ! Register H       -->  Inhalt von nn+1   !
            !  ! Register L       -->  Inhalt von nn     !
            !  !                                         !
LD   (nn),pp!20! Inhalt des Doppelregisters pp wird auf  !------
            !  ! die Adressen nn und nn+1 transportiert  !
            !  ! höherwertiges Reg -->  Inhalt von nn+1  !
            !  ! niederwertiges Reg -->  Inhalt von nn   !
            !  !                                         !
LD   (nn),IX!20! Inhalt des Indexregisters IX (bzw. IY)  !------
LD   (nn),IY!20! wird auf die Adressen nn und nn+1 trans-!------
            !  ! portiert                                !
            !  ! höherwert. Teil   -->  Inhalt von nn+1  !
            !  ! niederwert. Teil   -->  Inhalt von nn   !
            !  !                                         !
LD   SP,HL  ! 6! Inhalt des Doppelregisters HL wird in   !------
            !  ! den Stackpointer (Kellerzeiger) trans-  !
            !  ! portiert                                !
            !  !                                         !
```

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |

```
            !  !                                         !
LD   SP,IX  !10! Inhalt des Indexregisters IX (bzw. IY)  !------
LD   SP,IY  !10! wird in den Stackpointer (Kellerzeiger) !------
            !  ! transportiert                           !
            !  !                                         !
PUSH   qq   !11! Inhalt des Doppelregisters qq wird in   !------
            !  ! Kellerspeicher (Stack) transportiert:   !
            !  ! erniedrigen von SP                      !
            !  ! höherwertiges Reg --> Inhalt von SP-Adr !
            !  ! erniedrigen von SP                      !
            !  ! niederwertiges Reg --> Inhalt von SP-Adr!
            !  !                                         !
PUSH   IX   !15! Inhalt des Indexregisters IX (bzw. IY)  !------
PUSH   IY   !15! wird in den Kellerspeicher (Stack)      !------
            !  ! transportiert:                          !
            !  ! erniedrigen von SP                      !
            !  ! höherwert. Teil  -->  Inhalt von SP-Adr !
            !  ! erniedrigen von SP                      !
            !  ! niederwert. Teil  -->  Inhalt von SP-Adr!
            !  !                                         !
POP   qq    !10! Inhalt der von Stackpointer SP adres-   !------
            !  ! sierten 2 Speicherstellen wird in das   !
            !  ! Doppelregister qq übertragen:           !
            !  ! Inhalt von SP-Adr --> niederwertiges Reg!
```

```
            !  ! erhöhen von SP                              !
            !  ! Inhalt von SP-Adr --> höherwertiges Reg     !
            !  ! erhöhen von SP                              !
            !  !                                             !
POP   IX    !14! Inhalt der von Stackpointer SP adres-       !------
POP   IY    !14! sierten 2 Speicherstellen wird in das       !------
            !  ! Indexregister IX (bzw. IY) übertragen:      !
            !  ! Inhalt von SP-Adr --> niederwertig. Teil    !
            !  ! erhöhen von SP                              !
            !  ! Inhalt von SP-Adr --> höherwertig. Teil     !
            !  ! erhöhen von SP                              !
            !  !                                             !
```

### 3.7.3. Registeraustauschbefehle

Diese Befehle dienen dem schnellen Austausch von Doppelregister-inhalten und erschließen dem Programmierer die Hintergrund-register.

Zum Beispiel

```
UP:     EXX
        ...  ) Unterprogramm-
        ...  ) Befehle
        ...  )
        EXX
        RET
```

rettet die Registerinhalte BC, DE, HL für das Hauptprogramm. Im Unterschied zur Verwendung von PUSH und POP gehen die Inhalte der Hintergrundregister ggf. im Unterprogramm verloren.
Durch den Befehl EX DE,HL kann für in DE enthaltene Adressen auch die indirekte Adressierung (über HL) verwendet werden.

```
  Mnemonik  ! T!        Wirkungsweise des Befehls        !SZHPNC
            !  !                                         !   V
============================================================
            !  !                                         !
EX   DE,HL  ! 4! Die 16-Bit-Inhalte der Registerpaare DE !------
            !  ! und HL werden ausgetauscht              !
            !  ! DE   --->  HL                           !
            !  !                                         !
EX   AF,AF' ! 4! Die 16-Bit-Inhalte der Registerpaare AF !------
            !  ! und AF' werden ausgetauscht             !
            !  ! AF   --->  AF'                          !
            !  !                                         !
EXX         ! 4! Die 16-Bit-Inhalte der nachstehenden    !------
            !  ! Registerpaare werden ausgetauscht       !
            !  ! BC   --->  BC'                          !
```

```
            !  ! DE   --->  DE'                          !
            !  ! HL   --->  HL'                          !
            !  !                                         !
EX   (SP),HL!19! Der Inhalt des Registers L wird gegen   !------
            !  ! den Inhalt der Speicherstelle ausge-    !
            !  ! tauscht, die durch den Inhalt des Stack-!
            !  ! pointers SP adressiert ist. Der Inhalt  !
            !  ! des Registers H wird gegen den Inhalt   !
            !  ! der Speicherstelle ausgetauscht, die    !
            !  ! durch den Inhalt des Stackpointers SP   !
            !  ! plus 1 adressiert ist.                  !
            !  ! H   --->  (SP+1)                        !
            !  ! L   --->  (SP)                          !
            !  !                                         !
EX   (SP),IX!23! Der niederwertige Teil des Indexregi-   !------
            !  !                                         !
EX   (SP),IY!23! sters IX (bzw. IY) wird gegen den Inhalt!------
            !  ! der Speicherstelle ausgetauscht, die    !
            !  ! durch den Inhalt des Stackpointers SP   !
            !  ! adressiert ist. Der höherwertige Teil   !
            !  ! des Indexregisters IX (bzw. IY) wird    !
            !  ! gegen den Inhalt der Speicherstelle aus-!
            !  ! getauscht, die durch den Inhalt des     !
            !  ! Stackpointers SP plus 1 adressiert ist. !
            !  ! höherwertiger Teil   --->  (SP+1)       !
            !  ! niederwertiger Teil   --->  (SP)        !
            !  !                                         !
```

### 3.7.4. Blocktransfer- und -suchbefehle

Mit einem einzigen Befehl können beliebig große Datenmengen im Speicher verschoben werden, bzw. es kann in einem Speicherbereich nach einem Datenbyte gesucht werden. Die Suche wird dabei beendet, wenn das Byte gefunden wurde bzw. der gesamte Bereich durchsucht wurde.

Beispiel: Die Befehlsfolge

```
LD      HL,#C000
LD      DE,#C001
LD      BC,#3FFF
LD      (HL),0
LDIR
```

beschreibt den gesamten Bildwiederholspeicher mit Null-Bytes (löscht den Bildschirm mit PEN 0).

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |
| LDI | 16 | Transport eines Datenbytes von der Speicherstelle, die durch das Registerpaar HL adressiert wird nach der Speicherstelle, die durch das Registerpaar DE adressiert wird. Die Register DE und HL werden um 1 erhöht, und das Register BC wird um 1 vermindert.<br>BC = 0 --> PV = 0<br>BC > 0 --> PV = 1 | --0*0- |
| LDIR | 21<br>16 | Transport mehrerer Datenbytes ab der Speicherstelle, die durch das Registerpaar HL adressiert wird nach der Speicherstelle, die durch das Registerpaar DE adressiert wird. Bie Byteanzahl ist im Registerpaar BC enthalten. Nach jeder Byteübertragung wird der Inhalt von HL und von DE um 1 erhöht und BC um 1 vermindert. Die Übertragung endet, wenn der Inhalt von BC Null ist. | --000- |
| LDD | 16 | Der Befehl wirkt wie LDI, nur werden die Register DE und HL um 1 vermindert | --0*0- |
| LDDR | 21<br>16 | Der Befehl wirkt wie LDIR, nur werden die Register DE und HL um 1 vermindert | --000- |
| CPI | 16 | Vergleich des Inhalts des durch HL adressierten Speicherplatzes mit dem Inhalt des Akkumulators (A-Register):<br>A = (HL) --> Z = 1<br>A > (HL) --> Z = 0 | ****1- |

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |
| | | Anschließend wird das Register HL um 1 erhöht und das Registerpaar BC um 1 vermindert. Das Registerpaar BC kann als Bytezähler arbeiten:<br>BC = 0 --> PV = 0<br>BC > 0 --> PV = 1 | |
| CPIR | 21<br>16 | Vergleich des Inhalts des Akkumulators (A-Register) mit dem Inhalt eines adressierten Speicherbereiches. Die Anfangs- | ****1- |

```
            !  ! adresse des Bereiches ist im Register-  !
            !  ! paar HL enthalten, Die Länge des Berei- !
            !  ! ches im Registerpaar BC. Der Vergleich  !
            !  ! endet, wenn A = (HL) oder wenn BC = 0   !
            !  ! ist. Der Befehl ist also ein wiederhol- !
            !  ! ter Aufruf von CPI mit den Abbruchbedin-!
            !  ! gungen Z = 1 oder PV = 0.               !
            !  !                                         !
CPD         !16! Der Befehl wirkt wie CPI, nur das Regi- !****1-
            !  ! ster HL wird vermindert                 !
            !  !                                         !
CPDR        !21! Der Befehl wirkt wie CPIR, nur das Regi-!****1-
            !16! ster HL wird vermindert                 !
            !  !                                         !
```

### 3.7.5. 8-Bit-Arithmetik- und -Logikbefehle

Diese Befehle arbeiten mit Daten, die sich im Akkumulator (Register A als ersten Operanden) und mit Daten in anderen Registern oder auf Speicherplätzen (als zweiten Operanden) befinden. Das Ergebnis dieser Operationen wird im Akkumulator abgelegt.

```
  Mnemonik  ! T!        Wirkungsweise des Befehls        !SZHPNC
            !  !                                         !  V
============================================================
            !  !                                         !
ADD   A,r   ! 4! Der Inhalt von Register r wird zum Akku-!***V0*
            !  ! mulatorinhalt addiert                   !
            !  !                                         !
ADD  A,(HL) ! 7! Der Inhalt des durch das Registerpaar HL!***V0*
            !  ! adressierten Speicherplatzes wird zum   !
            !  ! Inhalt des Akkumulators addiert         !
            !  !                                         !
ADD   A,n   ! 7! Die Konstante n wird zum Inhalt des     !***V0*
            !  ! Akkumulators addiert                    !
            !  !                                         !
ADD A,(IX+d)!19! Der Inhalt des durch das Indexregister  !***V0*
ADD A,(IY+d)!19! IX (bzw. IY) plus Adreßverschiebung d   !***V0*
            !  ! adressierten Speicherplatzes wird zum   !
```

```
  Mnemonik  ! T!    Wirkungsweise des Befehls            !SZHPNC
            !  !                                         !  V
============================================================
            !  !                                         !
            !  ! Inhalt des Akkumulators addiert. Das Er-!
            !  ! gebnis steht im Akkumulator             !
            !  !                                         !
ADC   A,r   ! 4! Der Inhalt von Register r plus Carry-   !***V0*
            !  ! Flag wird zum Akkumulatorinhalt addiert !
```

| | | | |
|---|---|---|---|
| ADC A,(HL) | 7 | Der Inhalt des durch das Registerpaar HL adressierten Speicherplatzes plus Carry-Flag wird zum Inhalt des Akkumulators addiert | ***V0* |
| ADC A,n | 7 | Die Konstante n plus Carry-Flag wird zum Inhalt des Akkumulators addiert | ***V0* |
| ADC A,(IX+d) | 19 | Der Inhalt des durch das Indexregister IX (bzw. IY) plus Adreßverschiebung d adressierten Speicherplatzes plus Carry-Flag wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator. | ***V0* |
| ADC A,(IY+d) | 19 | | ***V0* |
| SUB r | 4 | Der Inhalt von Register r wird vom Akkumulatorinhalt subtrahiert | ***V1* |
| SUB (HL) | 7 | Der Inhalt des durch das Registerpaar HL adressierten Speicherplatzes wird vom Inhalt des Akkumulators subtrahiert | ***V1* |
| SUB n | 7 | Die Konstante n wird vom Inhalt des Akkumulators subtrahiert | ***V1* |
| SUB (IX+d) | 19 | Der Inhalt des durch das Indexregister IX (bzw. IY) plus Adreßverschiebung d adressierten Speicherplatzes wird vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator. | ***V1* |
| SUB (IY+d) | 19 | | ***V1* |
| SBC A,r | 4 | Der Inhalt von Register r plus Carry-Flag wird vom Akkumulatorinhalt subtrahiert | ***V1* |
| SBC A,(HL) | 7 | Der Inhalt des durch das Registerpaar HL adressierten Speicherplatzes plus Carry-Flag wird vom Inhalt des Akkumulators subtrahiert | ***V1* |
| SBC A,n | 7 | Die Konstante n plus Carry-Flag wird vom Inhalt des Akkumulators subtrahiert | ***V1* |

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |
| SBC A,(IX+d) | 19 | Der Inhalt des durch das Indexregister IX (bzw. IY) plus Adreßverschiebung d | ***V1* |
| SBC A,(IY+d) | 19 | | ***V1* |

| | | | | |
|---|---|---|---|---|
| | | | adressierten Speicherplatzes plus Carry- | |
| | | | Flag wird vom Inhalt des Akkumulators | |
| | | | subtrahiert. Das Ergebnis steht im Akku- | |
| | | | mulator. | |
| AND | r | 4 | Logische UND-Verknüpfung eines Regi- | **1P00 |
| AND | (HL) | 7 | sters, Speicherbytes oder Konstanten mit | **1P00 |
| AND | n | 7 | dem Akkumulator. Das spezifizierte Byte | **1P00 |
| AND | (IX+d) | 19 | wird bitweise mit dem Inhalt des Akkumu- | **1P00 |
| AND | (IY+d) | 19 | lators konjunktiv verknüpft. Das logi- | **1P00 |
| | | | sche UND ist nur dann 1, wenn beide Bits | |
| | | | 1 sind. | |
| OR | r | 4 | Logische ODER-Verknüpfung eines Regi- | **0P00 |
| OR | (HL) | 7 | sters, Speicherbytes oder Konstanten mit | **0P00 |
| OR | n | 7 | dem Akkumulator. Das spezifizierte Byte | **0P00 |
| OR | (IX+d) | 19 | wird bitweise mit dem Inhalt des Akkumu- | **0P00 |
| OR | (IY+d) | 19 | lators disjunktiv verknüpft. Das logi- | **0P00 |
| | | | sche ODER ist nur dann 0, wenn beide | |
| | | | Bits 0 sind. | |
| XOR | r | 4 | Exklusiv-ODER-Verknüpfung eines Regi- | **0P00 |
| XOR | (HL) | 7 | sters, Speicherbytes oder Konstanten mit | **0P00 |
| XOR | n | 7 | dem Akkumulator. Das spezifizierte Byte | **0P00 |
| XOR | (IX+d) | 19 | wird bitweise mit dem Inhalt des Akkumu- | **0P00 |
| XOR | (IY+d) | 19 | lators exklusiv verknüpft. Das Exklu- | **0P00 |
| | | | siv-ODER ist dann 1, wenn ein Bit = 1 | |
| | | | und das andere Bit = 0 ist. | |
| CP | r | 4 | Der Inhalt eines Registers, eines Spei- | ***V1* |
| CP | (HL) | 7 | cherbytes oder eine Konstante wird mit | ***V1* |
| CP | n | 7 | dem Akkumulator verglichen. Der ur- | ***V1* |
| CP | (IX+d) | 19 | sprüngliche Inhalt des Akkumulators | ***V1* |
| CP | (IY+d) | 19 | bleibt dabei erhalten. Das Vergleichs- | ***V1* |
| | | | ergebnis ist durch die Flag-Stellungen | |
| | | | erkennbar. | |
| INC | r | 4 | Der Inhalt des Registers r wird um 1 er- | ***V0- |
| | | | höht | |
| INC | (HL) | 11 | Der Inhalt des durch HL adressierten | ***V0- |
| | | | Speicherplatzes wird um 1 erhöht | |

| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| INC | (IX+d) | 23 | Der Inhalt des durch IX (bzw. IY) plus | ***V0- |
| INC | (IY+d) | 23 | Verschiebung d adressierten Speicher- | ***V0- |
| | | | platzes wird um 1 erhöht | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| DEC | r | 4 | Der Inhalt des Registers r wird um 1 vermindert | ***V1- |
| DEC | (HL) | 11 | Der Inhalt des durch HL adressierten Speicherplatzes wird um 1 vermindert | ***V1- |
| DEC | (IX+d) | 23 | Der Inhalt des durch IX (bzw. IY) plus Verschiebung d adressierten Speicherplatzes wird um 1 vermindert | ***V1- |
| DEC | (IY+d) | 23 | | ***V1- |
| DAA | | 4 | Korrigiert nach Addition / Subtraktion zweier gepackter BCD-Zahlen den Akkumulatorinhalt so, daß im Akkumulator wieder die gepackte BCD-Darstellung erreicht wird | ***P-* |
| CPL | | 4 | Bitweises Negieren (Komplementieren) des Akkumulatorinhalts | --1-1- |
| NEG | | 8 | Subtrahieren des Akkumulatorinhalts von Null. Entspricht wertmäßig dem Zweierkomplement | ***V1* |
| CCF | | 4 | Komplementieren des Carry-Flags | --x-0* |
| SCF | | 4 | Setzen des Carry-Flags | --0-01 |

### 3.7.6. 16-Bit-Arithmetikbefehle

Arbeiten ähnlich wie die 8-Bit-Arithmetikbefehle, jedoch mit Doppelregistern. Als Akkumulator wird eines der Register HL, IX oder IY benutzt.

| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|---|
| | | | | |
| ADD | HL,dd | 11 | Der Inhalt des Registerpaares dd wird zum Inhalt des Registerpaares HL addiert | --x-0* |
| ADD | IX,IX | 15 | Der Inhalt des Indexregistes IX (bzw.IY) wird mit sich selbst addiert. Diese Verdoppelung ist gleichbedeutend mit einer Linksverschiebung der 16 Bit um eine Position | --x-0* |
| ADD | IY,IY | 15 | | --x-0* |
| ADD | IX,pp | 15 | Der Inhalt von pp wird zum Inhalt des | --x-0* |

| | | | | |
|---|---|---|---|---|
| ADD | IY,pp | 15 | Indexregisters IX (bzw. IY) addiert | --x-0* |
| ADC | HL,dd | 15 | Der Inhalt von dd plus Carry-Flag wird zum Inhalt des Registerpaares HL addiert | **xV0* |
| SBC | HL,dd | 15 | Der Inhalt von dd plus Carry-Flag wird vom Inhalt des Registerpaares HL subtrahiert | **xV1* |
| INC | dd | 6 | Der Inhalt des Doppelregisters dd wird um 1 erhöht | ------ |
| INC | IX | 10 | Der Inhalt des Indexregisters IX (bzw. | ------ |
| INC | IY | 10 | IY) wird um 1 erhöht | ------ |
| DEC | dd | 6 | Der Inhalt des Doppelregisters dd wird um 1 vermindert | ------ |
| DEC | IX | 10 | Der Inhalt des Indexregisters IX (bzw. | ------ |
| DEC | IY | 10 | IY) wird um 1 vermindert | ------ |

### 3.7.7. Programmverzweigungsbefehle

Es ist zwischen unbedingten und bedingten Sprüngen zu unterscheiden. Es sind weiterhin relative Sprünge möglich, mit denen zu Marken in einer näheren Umgebung (-126 bis +129 Byte) um die Befehlsadresse verzweigt werden kann. Im Quellprogramm ist dabei zwar die absolute Adresse der Marke anzugeben, im Befehlscode aber erscheint nur die relative Verschiebung zum momentanen Befehlszählerstand. Das Maschinenprogramm wird damit unabhängig von seiner Lage im Speicher.
Bei bedingten Sprüngen sind Flag-Bedingungen als Operanden anzugeben (s. Abschn. 3.3.), und es werden die entsprechenden Flag-Bits getestet. In Abhängigkeit von diesem Test wird der Sprungbefehl entweder ausgeführt oder ignoriert.

| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| JP | nn | 10 | Unbedingter Sprung nach Adresse nn, indem der Befehlzzähler PC mit nn geladen wird | ------ |
| JP | NZ,nn | 10 | Sprung nach Adresse nn, wenn das Z-Flag gleich 0 ist | ------ |
| JP | Z,nn | 10 | Sprung nach Adresse nn, wenn das Z-Flag gleich 1 ist | ------ |

| | | | | |
|---|---|---|---|---|
| JP | NC,nn | 10 | Sprung nach Adresse nn, wenn das C-Flag gleich 0 ist | ------ |
| JP | C,nn | 10 | Sprung nach Adresse nn, wenn das C-Flag gleich 1 ist | ------ |
| JP | PO,nn | 10 | Sprung nach Adresse nn, wenn das P/V-Flag gleich 0 ist | ------ |
| JP | PE,nn | 10 | Sprung nach Adresse nn, wenn das P/V-Flag gleich 1 ist | ------ |
| JP | P,nn | 10 | Sprung nach Adresse nn, wenn das S-Flag gleich 0 ist | ------ |
| JP | M,nn | 10 | Sprung nach Adresse nn, wenn das S-Flag gleich 1 ist | ------ |
| JR | nn | 12 | Unbedingter relativer Sprung nach Adresse nn | ------ |
| JR | NZ,nn | 12/7 | Relativer Sprung nach Adresse nn, wenn das Z-Flag gleich 0 ist | ------ |
| JR | Z,nn | 12/7 | Relativer Sprung nach Adresse nn, wenn das Z-Flag gleich 1 ist | ------ |
| JR | NC,nn | 12/7 | Relativer Sprung nach Adresse nn, wenn das C-Flag gleich 0 ist | ------ |

| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| JR | C,nn | 12/7 | Relativer Sprung nach Adresse nn, wenn das C-Flag gleich 1 ist | ------ |
| JP | (HL) | 4 | Unbedingter Sprung zur Adresse, die im Doppelregister HL steht | ------ |
| JP | (IX) | 8 | Unbedingter Sprung zur Adresse, die im | ------ |
| JP | (IY) | 8 | Indexregister IX (bzw. IY) steht | ------ |
| DJNZ | nn | 13/8 | Der Inhalt des Registers B wird um 1 vermindert. Bedingter relativer Sprung zur Adresse nn, wenn der Inhalt des Registers B ungleich 0 ist. | ------ |

### 3.7.8. Unterprogrammbefehle

Es ist wie bei Sprungbefehlen zwischen bedingten und unbedingten Befehlen zu unterscheiden. Der Unterprogrammaufruf geschieht mit einem CALL-Befehl, bei dem, zusätzlich zum Programmsprung, die dem CALL-Befehl im Speicher folgende Adresse (Rückkehradresse) in den Kellerspeicher (Stack) gerettet wird. Wird nun das Unterprogramm mit einem RET-Befehl abgeschlossen, so wird diese Rückkehradresse in den Befehlszähler aus dem Stack zurückgeladen, und das Programm wird an der alten Stelle fortgesetzt.

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| CALL nn | 17 | Unbedingter Unterprogrammaufruf zur Adresse nn. Die nach dem CALL-Befehl folgende Speicheradresse wird wie bei einem PUSH-Befehl in den Stack gerettet (Rückkehradresse). Danach erfolgt ein unbedingter Sprung zur Adresse nn, indem der Befehlzzähler PC mit nn geladen wird | ------ |
| CALL NZ,nn | 17/10 | Unterprogrammaufruf zur Adresse nn, wenn das Z-Flag gleich 0 ist | ------ |
| CALL Z,nn | 17/10 | Unterprogrammaufruf zur Adresse nn, wenn das Z-Flag gleich 1 ist | ------ |
| CALL NC,nn | 17/10 | Unterprogrammaufruf zur Adresse nn, wenn das C-Flag gleich 0 ist | ------ |

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| CALL C,nn | 17/10 | Unterprogrammaufruf zur Adresse nn, wenn das C-Flag gleich 1 ist | ------ |
| CALL PO,nn | 17/10 | Unterprogrammaufruf zur Adresse nn, wenn das P/V-Flag gleich 0 ist | ------ |
| CALL PE,nn | 17/10 | Unterprogrammaufruf zur Adresse nn, wenn das P/V-Flag gleich 1 ist | ------ |

| | | | | |
|---|---|---|---|---|
| CALL | P,nn | 17<br>10 | Unterprogrammaufruf zur Adresse nn, wenn das S-Flag gleich 0 ist | ------ |
| CALL | M,nn | 17<br>10 | Unterprogrammaufruf zur Adresse nn, wenn das S-Flag gleich 1 ist | ------ |
| RST | p | 11 | Der RST-Befehl ist ein spezieller Unterprogrammaufruf. Es sind folgende 8 Restart-Adressen zugelassen:<br>p = #00,#08,#10,#18,#20,#28,#30,#38<br>Der höherwertige Adreßteil ist dabei stets 0. Der RST-Befehl entspricht in seiner weiteren Wirkung dem unbedingten Unterprogrammaufruf. | ------ |
| RET | | 10 | Unbedingter Unterprogrammrücksprung. Die Ausführung erfolgt, indem die Rückkehradresse wie bei einem POP-Befehl aus dem Stack geholt und in den Befehlszähler PC geladen wird. | ------ |
| RET | NZ | 11<br>5 | Unterprogrammrücksprung, wenn das Z-Flag gleich 0 ist | ------ |
| RET | Z | 11<br>5 | Unterprogrammrücksprung, wenn das Z-Flag gleich 1 ist | ------ |
| RET | NC | 11<br>5 | Unterprogrammrücksprung, wenn das C-Flag gleich 0 ist | ------ |
| RET | C | 11<br>5 | Unterprogrammrücksprung, wenn das C-Flag gleich 1 ist | ------ |
| RET | PO | 11<br>5 | Unterprogrammrücksprung, wenn das P/V-Flag gleich 0 ist | ------ |
| RET | PE | 11<br>5 | Unterprogrammrücksprung, wenn das P/V-Flag gleich 1 ist | ------ |
| RET | P | 11<br>5 | Unterprogrammrücksprung, wenn das S-Flag gleich 0 ist | ------ |

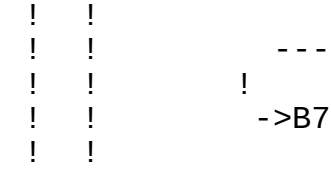
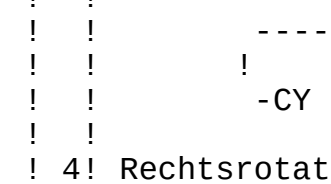
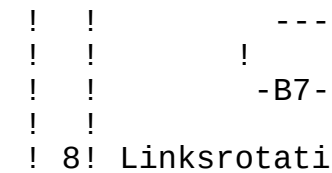
| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| RET | M | 11<br>5 | Unterprogrammrücksprung, wenn das S-Flag gleich 1 ist | ------ |

| | | | |
|---|---|---|---|
| RETI | 14 | Es erfolgt ein Rücksprung aus einer Interruptbehandlungsroutine, die durch einen maskierbaren Interrupt ausgelöst wurde. Dem peripheren Baustein, der den Interrupt auslöste, wird das Ende seines Programms mitgeteilt. Der Baustein gibt daraufhin die von ihm blockierte Interrupt-Kette wieder frei und ermöglicht damit die Abarbeitung niederwertiger Interrupts. Durch die RETI-Anweisung wird der maskierbare Interrupt nicht freigegeben. Es sollte deshalb vor jeder RETI-Anweisung ein EI-Befehl stehen, der die Annahme später folgender Interruptanforderungen ermöglicht. | ------ |
| RETN | 14 | Es erfolgt ein Rücksprung aus einer Interruptbehandlungsroutine, die durch einen nichtmaskierbaren Interrupt (NMI) ausgelöst wurde. Die Anweisung wirkt zunächst wie ein RET-Anweisung. Zusätzlich wird der Inhalt von IFF2 nach IFF1 übertragen, so daß die Abarbeitung maskierbarer Interrupts unmittelbar nach Ausführung des RETN-Befehls freigegeben ist, falls sie vor der NMI-Anforderung freigegeben war. | ------ |

### 3.7.9. Rotations- und Verschiebebefehle

Durch diese Befehle wird die Möglichkeit gegeben, im Akkumulator (Register A), in einem anderen Register oder in einem Speicherplatz Daten einfach zyklisch (bitweise) zu verschieben. Das aus dem Byte herausgeschobene Bit wird dabei im Carry-Flag abgelegt.

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |
| RLCA | 4 | Linksrotation des Akkumulatorinhalts. Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Das höchstwertige Bit 7 wird zum Inhalt des Bits 0 und des Carry-Flags.<br><br>----------><br>CY ----B7 ----B0 - | --0-0* |

| | | | | |
|---|---|---|---|---|
| RRCA | | 4 | Rechtsrotation des Akkumulatorinhalts. Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Das niederwertige Bit 0 wird zum Inhalt des Bits 7 und des Carry-Flags.<br>  | --0-0* |
| RLA | | 4 | Linksrotation des Akkumulatorinhalts durch das Carry-Flag. Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Das höchstwertige Bit 7 ersetzt das Carry-Flag, während das Carry-Flag das Bit 0 des Akkumulators ersetzt.<br>  | --0-0* |
| RRA | | 4 | Rechtsrotation des Akkumulatorinhalts durch das Carry-Flag. Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Das niederwertigste Bit 0 ersetzt das Carry-Flag, während das Carry-Flag das Bit 7 des Akkumulators ersetzt.<br>  | --0-0* |
| RLC | r | 8 | Linksrotation eines Registers oder Speicherbytes analog dem Befehl RLCA | **0P0* |
| RLC | (HL) | 15 | | **0P0* |
| RLC | (IX+d) | 23 | | **0P0* |
| RLC | (IY+d) | 23 | | **0P0* |

| *Mnemonik* | | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| RRC | r | 8 | Rechtsrotation eines Registers oder Speicherbytes analog dem Befehl RRCA | **0P0* |
| RRC | (HL) | 15 | | **0P0* |
| RRC | (IX+d) | 23 | | **0P0* |
| RRC | (IY+d) | 23 | | **0P0* |
| RL | r | 8 | Linksrotation eines Registers oder | **0P0* |

```
RL   (HL)   !15! Speicherbytes durch das Carry-Flag      !**0P0*
RL   (IX+d) !23! analog dem Befehl RLA                   !**0P0*
RL   (IY+d) !23!                                         !**0P0*
            !  !                                         !
RR   r      ! 8! Rechtsrotation eines Registers oder     !**0P0*
RR   (HL)   !15! Speicherbytes durch das Carry-Flag      !**0P0*
RR   (IX+d) !23! analog dem Befehl RRA                   !**0P0*
RR   (IY+d) !23!                                         !**0P0*
            !  !                                         !
SRA   r     ! 8! Rechtsverschiebung eines Registers oder !**0P0*
SRA   (HL)  !15! Speicherbytes um ein Bit durch das      !**0P0*
SRA   (IX+d)!23! Carry-Flag. Der Inhalt des höchstwerti- !**0P0*
SRA   (IY+d)!23! gen Bit 7 bleibt erhalten.              !**0P0*
            !  !                                         !
            !  !        -->                              !
            !  !       !   !                             !
            !  !        -B7---->B0---->CY                !
            !  !                                         !
SLA   r     ! 8! Linksverschiebung eines Registers oder  !**0P0*
SLA   (HL)  !15! Speicherbytes um ein Bit durch das      !**0P0*
SLA   (IX+d)!23! Carry-Flag. Das niederwertige Bit 0     !**0P0*
SLA   (IY+d)!23! wird 0.                                 !**0P0*
            !  !                                         !
            !  !       CY ----B7 ----B0 ----0            !
            !  !                                         !
SRL   r     ! 8! Rechtsverschiebung eines Registers oder !**0P0*
SRL   (HL)  !15! Speicherbytes um ein Bit durch das      !**0P0*
SRL   (IX+d)!23! Carry-Flag. Das höchstwertige Bit 7     !**0P0*
SRL   (IY+d)!23! wird 0.                                 !**0P0*
            !  !                                         !
            !  !       0---->B7---->B0---->CY            !
            !  !                                         !
RLD         !18! Zyklische Verschiebung nach links zwi-  !**0P0-
            !  ! schen dem Akkumulator und dem Inhalt des!
            !  ! durch HL adressierten Speicherplatzes.  !
            !  ! Die unteren 4 Bit des durch HL adres-   !
            !  ! sierten Speicherplatzes werden in die   !
            !  ! oberen 4 Bitstellen übertragen und      !
            !  ! diese ihrerseits in die unteren 4 Bit-  !
            !  ! stellen des Akkumulators. Die unteren 4 !
            !  ! Bits des Akkumulators werden in die un- !
            !  ! teren 4 Bitstellen der Speicherstelle   !
            !  ! transportiert. Die oberen 4 Bits des Ak-!
            !  ! kumulators bleiben unberührt.           !
```

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |

```
            !  !                                         !
RRD         !18! Zyklische Verschiebung nach rechts zwi- !**0P0-
```

```
            !   ! schen dem Akkumulator und dem Inhalt des!
            !   ! durch HL adressierten Speicherplatzes.   !
            !   ! Die unteren 4 Bit des durch HL adres-    !
            !   ! sierten Speicherplatzes werden in die    !
            !   ! unteren 4 Bitstellen des Akkumulators    !
            !   ! übertragen und diese in die oberen der   !
            !   ! durch HL adressierten Speicherstelle.    !
            !   ! Die oberen 4 Bits der durch HL adres-    !
            !   ! sierten Speicherstelle werden in die un-!
            !   ! teren 4 Bitstellen transportiert. Die    !
            !   ! oberen 4 Bits des Akkumulators bleiben   !
            !   ! unberührt.                               !
            !   !                                          !
```

### 3.7.10. Einzelbitbefehle

Diese Befehle erlauben es, einzelne Bits in Registern oder auf Speicherplätzen zu testen, zu setzen oder zu löschen.

```
  Mnemonik  ! T!        Wirkungsweise des Befehls         !SZHPNC
            !  !                                          !  V
============================================================
            !  !                                          !
BIT   b,r   ! 8! Die durch b gekennzeichnete Bitposition !x*1x0-
            !  ! wird in dem  Register r getestet. Das    !
            !  ! Komplement des zu testenden Bits wird in!
            !  ! das Z-Flag geladen.                      !
            !  !                                          !
BIT  b,(HL) !12! Die durch b gekennzeichnete Bitposition !x*1x0-
            !  ! wird in der Speicherstelle getestet, die!
            !  ! durch das Register HL adressiert ist.    !
            !  ! Das Komplement des zu testenden Bits     !
            !  ! wird in das Z-Flag geladen.              !
            !  !                                          !
BIT b,(IX+d)!20! Die durch b gekennzeichnete Bitposition !x*1x0-
BIT b,(IY+d)!20! wird in der Speicherstelle getestet, die!x*1x0-
            !  ! durch das Indexregister IX (bzw. IY)     !
            !  ! plus Verschiebung d adressiert ist. Das  !
            !  ! Komplement des zu testenden Bits wird in!
            !  ! das Z-Flag geladen.                      !
            !  !                                          !
SET   b,r   ! 8! Die durch b gekennzeichnete Bitposition !------
            !  ! wird in dem  Register r gesetzt (auf 1)  !
            !  !                                          !
SET  b,(HL) !15! Die durch b gekennzeichnete Bitposition !------
            !  ! wird in der Speicherstelle gesetzt, die  !
            !  ! durch das Register HL adressiert ist     !
```

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| SET b,(IX+d) | 23 | Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch das Indexregister IX (bzw. IY) plus Verschiebung d adressiert ist | ------ |
| SET b,(IY+d) | 23 | | ------ |
| RES b,r | 8 | Die durch b gekennzeichnete Bitposition wird in dem Register r gelöscht (Null) | ------ |
| RES b,(HL) | 15 | Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gelöscht, die durch das Register HL adressiert ist | ------ |
| RES b,(IX+d) | 23 | Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gelöscht, die durch das Indexregister IX (bzw. IY) plus Verschiebung d adressiert ist | ------ |
| RES b,(IY+d) | 23 | | ------ |

### 3.7.11. CPU-Steuerbefehle

Diese Befehle dienen zur Steuerung des Interruptsystems der CPU. Der Interruptmodus ist im KC compact auf IM 2 eingestellt.

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| NOP | 4 | Die CPU führt keine Operation aus. Es werden aber Refresh-Zyklen erzeugt. | ------ |
| HALT | 4 | Die CPU führt solange eine Folge von NOP-Befehlen aus, bis ein Interrupt oder der RESET-Eingang an der CPU aktiv wird. Es werden Refresh-Zyklen erzeugt. | ------ |
| DI | 4 | Der maskierbare Interrupt wird durch Rücksetzen der Interrupt-Freigabe-Flip-Flops IFF1 und IFF2 der CPU gesperrt. Nichtmaskierbare Interrupts werden anerkannt. | ------ |
| EI | 4 | Der maskierbare Interrupt wird durch Setzen der Interrupt-Freigabe-Flip-Flops IFF1 und IFF2 der CPU freigegeben. Während der Ausführung des Befehls | ------ |

```
            !   ! akzeptiert die CPU keine Interruptanfor-!
            !   ! derungen.                                !
            !   !                                          !
```

| *Mnemonik* | *!* | *T!* | *Wirkungsweise des Befehls* | *!SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| IM 0 | ! | 8! | Der Befehl bringt die CPU in den Interruptmodus 0 | !------ |
| IM 1 | ! | 8! | Der Befehl bringt die CPU in den Interruptmodus 1 | !------ |
| IM 2 | ! | 8! | Der Befehl bringt die CPU in den Interruptmodus 2 | !------ |

### 3.7.12. Ein- und Ausgabebefehle

Mit diesen Befehlen können Datenbytes zwischen Registern oder Speicheradressen und externen Bausteinen ausgetauscht werden. Der externe Baustein wird dabei über eine sog. Portadresse (8-Bit-Wert) angesprochen. Diese Portadresse wird je nach Befehl entweder direkt angegeben (als Konstante) oder muß im Register C zur Verfügung stehen. Ähnlich den Blocktransferbefehlen existieren auch hier Befehle für die Daten-Ein- und -Ausgabe ganzer Speicherbereiche. Diese können beim KC compact nicht verwendet werden, da nicht wie üblich im Register C sondern im Register B die Kanaladresse bereitgestellt werden muß (siehe /1/). Bei den Befehlen INI, INIR, IND, INDR, OUTI, OTIR, OUTD und OTDR wird außerdem B als Zählregister verwendet, und die Kanaladresse würde damit nach jedem IN/OUT um 1 dekrementiert.

| *Mnemonik* | *!* | *T!* | *Wirkungsweise des Befehls* | *!SZHPNC* |
|---|---|---|---|---|
| | | | | V |
| IN A,(n) | ! | 11! | Die Eingabekanaladresse wird mittels der Konstanten n eingestellt. Zielregister ist der Akkumulator. (n) --> A | !------ |
| IN r,(C) | ! | 12! | Die Eingabekanaladresse wird indirekt über das Register C eingestellt. Zielregister ist r. (C) --> r | !**0P0- |

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* |
|---|---|---|---|
| | | | V |
| INI | 16 | Die Eingabekanaladresse wird indirekt über das Register C eingestellt. Zieladresse ist der durch HL adressierte Speicherplatz. B kann als Bytezähler arbeiten.<br>B wird um 1 vermindert, HL um 1 erhöht.<br>(C) --> (HL)<br>B-1 --> B<br>HL+1 --> HL | x*xx1- |
| INIR | 21<br>16 | Die Eingabekanaladresse wird indirekt über das Register C eingestellt. Zieladresse ist der durch HL adressierte Speicherplatz. B arbeitet als Bytezähler<br>B wird um 1 vermindert, HL um 1 erhöht.<br>Es wird eine Blockübertragung durchgeführt bis B = 0 ist.<br>(C) --> (HL)<br>B-1 --> B<br>HL+1 --> HL<br>Wiederholen bis B = 0 | x1xx1- |
| IND | 16 | Die Eingabekanaladresse wird indirekt über das Register C eingestellt. Zieladresse ist der durch HL adressierte Speicherplatz. B kann als Bytezähler arbeiten.<br>B und HL werden um 1 vermindert.<br>(C) --> (HL)<br>B-1 --> B<br>HL-1 --> HL | x*xx1- |
| INDR | 21<br>16 | Die Eingabekanaladresse wird indirekt über das Register C eingestellt. Zieladresse ist der durch HL adressierte Speicherplatz. B arbeitet als Bytezähler<br>B und HL werden um 1 vermindert.<br>Es wird eine Blockübertragung durchgeführt bis B = 0 ist.<br>(C) --> (HL)<br>B-1 --> B<br>HL-1 --> HL<br>Wiederholen bis B = 0 | x1xx1- |
| OUT (n),A | 11 | Die Ausgabekanaladresse wird mittels der Konstanten n eingestellt. Quellregister | ------ |

| | | | |
|---|---|---|---|
| | | ist der Akkumulator. | |
| | | A --> (n) | |
| | | | |
| OUT (C),r | 12 | Die Ausgabekanaladresse wird indirekt über das Register C eingestellt. Quellregister ist r. r --> (C) | ------ |
| | | | |

| *Mnemonik* | *T* | *Wirkungsweise des Befehls* | *SZHPNC* V |
|---|---|---|---|
| | | | |
| OUTI | 16 | Die Ausgabekanaladresse wird indirekt über das Register C eingestellt. Quelladresse ist der durch HL adressierte Speicherplatz. B kann als Bytezähler arbeiten. B wird um 1 vermindert, HL um 1 erhöht. (HL) --> (C) B-1 --> B HL+1 --> HL | x*xx1- |
| | | | |
| OTIR | 21 16 | Die Ausgabekanaladresse wird indirekt über das Register C eingestellt. Quelladresse ist der durch HL adressierte Speicherplatz. B arbeitet als Bytezähler B wird um 1 vermindert, HL um 1 erhöht. Es wird eine Blockübertragung durchgeführt bis B = 0 ist. (HL) --> (C) B-1 --> B HL+1 --> HL Wiederholen bis B = 0 | x1xx1- |
| | | | |
| OUTD | 16 | Die Ausgabekanaladresse wird indirekt über das Register C eingestellt. Quelladresse ist der durch HL adressierte Speicherplatz. B kann als Bytezähler arbeiten. B und HL werden um 1 vermindert. (HL) --> (C) B-1 --> B HL-1 --> HL | x*xx1- |
| | | | |
| OTDR | 21 16 | Die Ausgabekanaladresse wird indirekt über das Register C eingestellt. Quelladresse ist der durch HL adressierte Speicherplatz. B arbeitet als Bytezähler arbeiten. B und HL werden um 1 vermindert. | x1xx1- |

```
!  ! Es wird eine Blockübertragung durchge-  !
!  ! führt bis B = 0 ist.                     !
!  ! (HL)  -->  (C)                           !
!  ! B-1   -->  B                             !
!  ! HL-1  -->  HL                            !
!  ! Wiederholen bis B = 0                    !
!  !                                          !
```

### 3.8. Abarbeitung des Assemblers

Der Assembler wird mit 'A' im Editor gestartet. Als erstes wird die, für die Markentabelle zu reservierende, Speichergröße abgefragt. In der Regel reicht hier die Betätigung der ^RETURN_-Taste. Damit wird nämlich ein von der Länge des Quelltextes abhängiger Durchschnittswert verwendet. Wird aber mit der 'include'-Option (Assembler-Kommando '*F') gearbeitet, reicht dieser Wert in der Regel nicht aus, und es muß bei 'Table size:' ein größerer Wert angegeben werden. Als zweites wird nach Assembler-Optionen gefragt.

#### 3.8.1. Assembler-Optionen

Es ist die gewünschte Optionsnummer einzugeben. Sollen mehrere Optionen gesetzt werden, sind deren Nummern zu addieren. Soll z.B. nur die Markentabelle auf dem Drucker ausgegeben werden, wird die Option 15 (1+2+4+8) angegeben.

Folgende Optionen sind möglich:

| Options-Nummer | Bedeutung |
|---|---|
| 1 | Markentabelle am Ende des 2. Passes ausgeben |
| 2 | Objektcodegenerierung unterdrücken |
| 4 | Assemblerlistenausgabe unterdrücken |
| 8 | Ausgabe der Assemblerliste auf den Drucker |
| 16 | Objektcode hinter der Markentabelle ablegen, auch wenn mit 'ORG' ein anderer Bereich festgelegt wird, so daß der Code zwar in einem Speicherbereich abgelegt wird aber nur auf einem anderem lauffähig ist ('ENT' wird außer Kraft gesetzt) . |
| 32 | nicht prüfen, wo der Objektcode abgelegt wird --> Beschleunigung der Assemblierung |

#### 3.8.2. Fehlermeldungen

Neben der Markentabellenerstellung wird im ersten Assemblerlauf auf Fehler geprüft. Bei einer Fehlererkennung wird die Fehler-

zeile und die Fehlernummer (siehe Anhang D) ausgegeben. Hier kann mit ^ESC_ in den Editor zurückgekehrt oder mit jeder anderen Taste die Assemblierung fortgesetzt werden. Nach Beenden des ersten Passes erfolgt die Meldung:

Pass 1 errors: nn

### 3.8.3. Assemblerliste

Während des zweiten Assemblerpasses kann neben der Objektcode-generierung eine Assemblerliste ausgegeben werden.

Form der Liste:

```
  Spalte 1  Spalte 2  Spalte 3   Spalte 4   Spalte 5   Spalte 6
 (Stelle 1)(Stelle 6)(Stelle 15)(Stelle 21)(Stelle 28)(Stelle 33)

z.B. 2800    C5        130        LOOP:      PUSH       BC
```

Bedeutung der Spalten:

| Spalte | Bedeutung |
|---|---|
| 1 | Wert des Befehlszählers, außer bei ORG, ENT und EQU, wo der Operandenwert in Spalte 1 ausgegeben wird |
| 2 | generierter Objektcode (bis zu acht Zeichen = 4 Bytes) |
| 3 | Zeilennummer |
| 4 | Name, falls vorhanden |
| 5 | Assemblerbefehl |
| 6 | Operanden und Kommentare |

Durch beliebige Tastenbetätigung kann die Auflistung unterbrochen werden. Mit ^ESC_ kann in den Editor zurückgekehrt und mit jeder anderen Taste die Auflistung fortgesetzt werden. Am Ende wird ausgegeben:

Pass 2 errors: nn

Markentabelle (wenn Option 1 gesetzt war)

Table used: xxxxx from yyyyy
Executes: nnnnn

'Table used:' gibt die gebrauchte und die reservierte Speichergröße der Markentabelle an.
'Executes:' zeigt dezimal den Befehlszählerstand für das Startkommando 'R' des Editors an.

# 4. D i s a s s e m b l e r / D e b u g g e r

## 4.1. Einführung

Mit ASSMON2 werden dem Assemblerprogrammierer ein Disassembler und viele notwendige Debuggerfunktionen bereitgestellt. Die Bedienung des Programms erfolgt über die Eingabe von Kommando-Buchstaben. Nach jedem Kommando werden alle Bildschirmausgaben aktualisiert. Einige Kommandos verlangen die Eingabe einer hexadezimalen Zahl. Werden hier mehr als 4 Ziffern eingegeben, werden nur die vier zuletzt eingegebenen Ziffern beachtet.

## 4.2. Bildschirmausgaben

Nach dem Laden des Programms und der ersten Ausschrift wird das folgende Bild aufgebaut:

```
  X+00  yyyyyyyy       zzzzzzzzzzzzzzz

    PC AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    SP AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    IY AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    IX AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    HL AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    DE AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    BC AAAA   a1 a2 a3 a4 a5 a6 a7 a8 a9
    AF AAAA   ffffffff
    IR AAAA

   X-12  xx  X-04  xx  X+04  xx  X+12  xx
   X-11  xx  X-03  xx  X+05  xx  X+13  xx
   X-10  xx  X-02  xx  X+06  xx  X+14  xx
   X-09  xx  X-01  xx  X+07  xx  X+15  xx
   X-08  xx >X+00  xx  X+08  xx  X+16  xx
   X-07  xx  X+01  xx  X+09  xx  X+17  xx
   X-06  xx  X+02  xx  X+10  xx  X+18  xx
   X-05  xx  X+03  xx  X+11  xx  X+19  xx

  >
```

Erklärungen:

```
X+00   - Adresse, auf die der Speicherzeiger weist
xx     - Inhalt der Adresse links daneben
y...y  - aktuelle Befehlsbytes (1-4)
z...z  - Mnemonik des aktuellen Befehls
AAAA   - Inhalt des links daneben stehenden Registerpaars
a1-a9  - Inhalt der Adresse AAAA links daneben, sowie Inhalte
         der folgenden Adressen
f...f  - gesetzte Flags
```

## 4.3. Kommandos

### 4.3.1. Disassembler

^CTRL_-^A_

Mit diesem Kommando wird eine Seite ab der aktuellen Speicherzeigeradresse disassembliert. Mit jeder weiteren Tastenbetätigung wird weiterdisassembliert. In den Kommandomodus gelangt man mit ^CTRL_-^A_ zurück.

T

Mit 'T' kann ein Maschinencodeblock disassembliert und als Listing auf dem Bildschirm, dem Drucker und/oder als Quelltext im Speicher abgelegt werden. Dazu sind als erstes die Anfangsadresse (First:) und die Endadresse (Last:) des zu übersetzenden Blockes einzugeben. Wird die sich anschließende Frage nach der Druckerausgabe (Printer?) mit 'Y' beantwortet, wird das Listing auf den Drucker ausgegeben. Alle anderen Zeichen bewirken die Ausgabe auf den Bildschirm. Anschließend ist die Anfangsadresse für das entstehende Quelltextprogramm anzugeben (Text:). Soll keine Datei erstellt werden, ist an dieser Stelle nur ^RETURN_ einzugeben. Wenn die Datei mit ASSMON1 bearbeitet werden soll, muß deren Anfangsadresse für Quelltexte in ASSMON1 übereinstimmen (siehe Kommando 'X' in Abschn. 2.4.) bzw. die Datei muß dorthin verschoben werden. Weiterhin ist die Textendeadresse, die der Disassembler zum Schluß ausgibt, an ASSMON1 zu übergeben. In ASSMON1 stehen dazu zwei Arbeitszellen bereit, in die diese Endadresse mittels 'POKE' gespeichert werden muß.
Die Arbeitszellen sind:

XXXX+7 - Low-Teil der Textendeadresse
XXXX+8 - High-Teil der Textendeadresse
XXXX - Ladeadresse von ASSMON1

Soll eine Textdatei erstellt werden, ist noch eine Anfangsadresse (Workspace:) für die entstehende Markentabelle einzugeben. Für jede erkannte Marke werden 2 Bytes benötigt. Liegt eine Marke innerhalb des zu übersetzenden Bereichs, wird die Marke in der Form 'Lnnnn' (nnnn-Hexadezimalwert der Marke) ausgegeben. Liegt die Marke außerhalb, wird nur der Hexadezimalwert eingetragen.
Als letztes vor dem Disassemblerstart können noch Datenbereiche innerhalb des zu übersetzenden Blocks angegeben werden, die mit 'DEFB' übersetzt werden. Dazu ist jeweils deren Anfangsadresse (First:) und Endadresse (Last:) anzugeben. Sollen keine bzw. keine weiteren Datenbereiche mehr eingegeben werden, ist bei 'First:' und 'Last:' nur ^RETURN_ einzugeben.
Nach dem Disassemblerstart wird der Bildschirm gelöscht und die Markentabelle erstellt. Anschließend wird dann die

Disassemblerliste auf den Bildschirm oder Drucker ausgegeben. Mit beliebiger Tastenbetätigung kann das Listen gestoppt und auch wieder fortgesetzt werden. Mit ^ESC_ nach dem Stoppen wird in den Kommandomodus zurückgekehrt.
Ungültige Codes werden mit 'NOP' übersetzt, und nach dem Code wird ein '*' im Listing ausgegeben.

Beispiel:
Die ersten 256 Bytes von ASSMON1 sollen zurückübersetzt werden. Dazu folgende Vorgehensweise als Vorschlag:
- Laden von ASSMON2 auf die Adresse 30000
- Verlassen von ASSMON2 mit ^CTRL_-^X_
- Laden von ASSMON1 auf die Adresse 1000
- Textanfangsadresse mit 'X' ausgeben lassen (9501=#251D)
- Sprung in ASSMON2 mit ^CTRL_-^J_
- Aufruf 'T'
- Eingabe First: 3E8 (=1000)
- Eingabe Last: 4E8
- Eingabe Printer? ^RETURN_
- Eingabe Text: 251D (siehe 'X'-Kommando von ASSMON1)
- Eingabe Workspace: 7000
- Eingabe First: ^RETURN_
- Eingabe Last: ^RETURN_
- Disassemblierung läuft; es folgt die Ausschrift:
  Text end = #2B19
- Verlassen von ASSMON2 mit ^CTRL_-^X_
- POKE 1007,&19:POKE 1008,&2B ^RETURN_
- Sprung in ASSMON1 mit CALL 1004
- Bearbeitung des rückübersetzten Quelltextes

### 4.3.2. Speicherzeigerkommandos

**Cursor rechts**

Der Speicherzeiger wird um eins erhöht.

**Cursor links**

Der Speicherzeiger wird um eins verringert.

**Cursor runter**

Der Speicherzeiger wird um acht erhöht.

**Cursor hoch**

Der Speicherzeiger wird um acht verringert.

**M**

'M' setzt den Speicherzeiger auf die nach dem erscheinenden Doppelpunkt einzugebende Adresse.

**O**

'O' verändert den Speicherzeiger relativ um den Wert, der in der Speicherzelle, auf den der Speicherzeiger momentan zeigt, steht. Das entspricht einem relativen Sprung (JR xx). Zu beachten ist, daß Werte, die größer als #7F sind, den Speicherzeiger verringern.

**U**

'U' setzt den Speicherzeiger wieder auf den Wert, der vor einem 'O'-Kommando eingestellt war. 'U' kann also nur nach einem 'O' verwendet werden. Zu beachten ist, daß nur jeweils ein Rücksprung möglich ist, falls mehrere 'O'-Kommandos nacheinander eingegeben wurden.

**X**

'X' setzt den Speicherzeiger auf die Adresse, deren Wert sich aus der Speicherzelle, auf die der Zeiger momentan weist, und der folgenden Zelle ergibt.
Beispiel: Der Zeiger steht auf einem CALL-Befehl (CD xx yy) und der Zeiger soll auf die Anfangsadresse des dort gerufenen Unterprogramms gestellt werden. Dazu ist der Zeiger mit Cursor rechts auf den Low-Teil der Adresse zu verschieben und anschließend 'X' einzugeben.

**V**

'V' setzt den Speicherzeiger wieder auf den Wert, der vor einem 'X'-Kommando bestanden hat. 'V' kann also nur nach einem 'X' verwendet werden. Wie bei 'U' ist auch mit 'V' jeweils nur ein Rücksprung bei mehreren hintereinander eingegebenen 'X'-Kommandos möglich.

**S**

Der Speicherzeiger wird auf die Adresse gesetzt, die sich gerade im Stack befindet (z.B. eine Rücksprungadresse aus einem gerufenen Unterprogramm).

### 4.3.3. Abarbeiten von Code

^CTRL_-^S_

Es wird der Befehl ausgeführt, der auf der Adresse steht, auf die der Speicherzeiger zeigt (Einzelschritt). Zu beachten ist, daß auch der Befehlszähler (PC) vorher auf diese Adresse zu setzen ist.

**J**

Es ist eine Adresse einzugeben, ab der der Maschinencode in Echtzeit ausgeführt werden soll. Soll noch einmal in den Kommandomodus zurückgekehrt werden, müssen im Maschinencode Unterbrechungspunkte gesetzt werden ('!'-Kommando).

**^CTRL_-^C_**

Der Maschinencode wird ab der Adresse, die im Befehlszähler (PC) steht, in Echtzeit ausgeführt. Soll noch einmal in den Kommandomodus zurückgekehrt werden, müssen Unterbrechungspunkte gesetzt werden.

**!**

Auf der Adresse, auf die der Speicherzeiger weist, wird ein Unterbrechungspunkt gesetzt. D.h., es wird ein CALL-Befehl auf eine Routine in ASSMON2 eingetragen. Diese Routine bewirkt die Unterbrechung. Die ursprünglichen drei Bytes werden gesichert und nach der Unterbrechung zurückgeschrieben.

**>**

'>' setzt einen Unterbrechungspunkt nach dem laufenden Befehl und ruft anschließend ^CTRL_-^C_ auf. Das ist z.B. bei Unterprogrammaufrufen (CALL) sinnvoll.

### 4.3.4. Speicher- und Registeränderungen

**I**

Mit 'I' wird ein Speicherblock an einen anderen Platz kopiert. Dazu wird die Anfangs- (First:), die End- (Last:) und die Zieladresse (To:) eingegeben. Ist die Endadresse kleiner als die Anfangsadresse, wird das Kommando abgebrochen.

**P**

Mit 'P' kann ein Speicherblock mit einem anzugebenden Byte gefüllt werden. Dazu ist die Anfangs- (First:), die Endadresse (Last:) und das Füllbyte (With:) anzugeben.

Speicheränderung

Der Inhalt der Adresse, auf die der Speicherzeiger weist, kann dadurch geändert werden, in dem einfach eine Hexadezimalzahl (es werden immer die 2 niederwertigsten Stellen genommen) gefolgt von einem Beendigungszeichen (nicht 0-9 oder A-F) eingegeben wird. Wird nur eine Ziffer eingegeben, wird die Zahl links mit Null ergänzt. Ist das Beendigungszeichen ein gültiges Kommando, wird die Zahl gespeichert und anschließend das Kommando ausgeführt. Stellt das Beendigungszeichen kein gültiges Kommando dar, wird es ignoriert.

. (Registeränderung)

Oben links auf dem Bildschirm stehen die Register und deren Inhalt. Ein Pfeil '>' zeigt auf ein Registerpaar (Registerzeiger). Der Registerzeiger kann mit Punkt '.' auf das nächste Registerpaar verschoben werden usw. Soll der Inhalt eines Registerpaares geändert werden, ist der Zeiger auf dieses Registerpaar zu verschieben und der neue Registerinhalt (Hexadezimalzahl) gefolgt von einem Punkt '.' einzugeben.

### 4.3.5. Lade- und Rette-Kommandos

**R**

Mit 'R' kann eine Datei von Band eingelesen werden. Dazu ist der Name der Datei (wird nur ^RETURN_ eingegeben, wird die erste gefundene Datei geladen) und die Adresse, ab der die Datei abgelegt werden soll, einzugeben.

**W**

Mit 'W' kann ein Speicherblock auf Band gerettet werden. Dazu ist ein Name für die Datei sowie die Anfangs- und Endadresse einzugeben.

### 4.3.6. Weitere Kommandos

**G**

Es kann mit 'G' nach einer Zeichenkette gesucht werden. Die Zeichenkette ist byteweise einzugeben (immer ein Byte nach ':' und ^RETURN_ eingeben). Ist die Kette vollständig eingegeben, wird nach dem Doppelpunkt nur ^RETURN_ eingegeben, und die Suche nach dem ersten Auftreten der Kette beginnt. Wird die Kette im Speicher gefunden, weist der Speicherzeiger auf das erste Zeichen

der gefundenen Zeichenkette.

**N**

'N' sucht das nächste Auftreten der Zeichenkette, die mit 'G' eingegeben wurde.

**L**

'L' löscht den Bildschirm und gibt einen 160 Bytes langen Speicherbereich ab der Adresse, auf die der Speicherzeiger weist, auf dem Bildschirm aus. Es werden die Adressen und deren Inhalt hexadezimal und im ASCII-Zeichenformat dargestellt (20 Zeilen mit jeweils 8 Bytes). Für die ASCII-Zeichendarstellung wird für Werte >#7F der Wert #80 abgezogen, und die Werte 0 bis #1F werden mit '.' dargestellt.

Danach kann mit ^ESC_ in den Kommandomodus zurückgekehrt oder mit jeder anderen Taste ein weiterer 160 Bytes langer Bereich gelistet werden.

**^CTRL_-^L_**

Dieses Kommando entspricht 'L'. Die Ausgabe erfolgt jedoch auf dem Drucker.

**H**

Eine Dezimalzahl wird in ihr hexadezimales Äquivalent umgewandelt. Dazu ist nach dem Doppelpunkt die dezimale Zahl einzugeben. Beendet wird die Eingabe mit einem nicht-numerischen Zeichen (außer 0-9). Danach erscheint ein '=' und das hexadezimale Äquivalent.

**Y**

Mit 'Y' kann eine ASCII-Zeichenkette eingegeben werden, die ab der Adresse abgelegt wird, auf die der Speicherzeiger verweist. Die Kette wird mit ^ESC_ abgeschlossen und mit ^DEL_ kann jeweils ein Zeichen gelöscht werden.

**^CTRL_-^X_**

Mit diesem Kommando kehrt man in den BASIC-Interpreter zurück. Mit CALL x+2 (x=Ladeadresse von ASSMON2) kann ASSMON2 von BASIC aus wieder gestartet werden.

**^CTRL_-^J_**

Mit diesem Kommando wird in das Programm ASSMON1 gesprungen, falls dieses geladen ist.

## 5. P r o g r a m m b e i s p i e l

Da der KCc sehr gute Grafikmöglichkeiten bietet, ist als Beispiel ein Sprite-Programm (Sprite=Grafikfigur, die über den Bildschirm bewegt werden kann, ohne dessen Inhalt zu zerstören) ausgewählt worden. Das Programm arbeitet im Bildschirmmodus 0.
Als erstes ist das im folgenden abgebildete Programm mit 'I' einzugeben (jede Zeile mit ^RETURN_ abschließen):

```
  1 ;*********************************
  2 ;* UEBUNGSPROGRAMM FUER ASSMON1  *
  3 ;*                               *
  4 ;* SPRITE-PROGRAMM 'STIEFEL'     *
  5 ;*                               *
  6 ;* VEB MIKROELEKTRONIK           *
  7 ;* 'WILHELM PIECK' MUEHLHAUSEN   *
  8 ;*********************************
  9
 10 ;VEREINBARUNGEN:
 11
 12 DOTPOS: EQU     #BC1D   ;BILDSCHIRMADRESSE AUS POSITION
 13 NEXTLI: EQU     #BC26   ;NAECHSTE ZEILENADRESSE
 14 MODUS:  EQU     #BC0E   ;BILDSCHIRMMODUS FESTLEGEN UND
CLS
 15 SPALTE: EQU     #BB6F   ;CURSOR IN SPALTE SETZEN
 16 ZEILE:  EQU     #BB72   ;CURSOR AUF ZEILE SETZEN
 17 TEXTOU: EQU     #BB5A   ;ASCII-ZEICHEN AUF BILDSCHIRM
 18 PEN:    EQU     #BB90   ;SCHREIBSTIFT EINSTELLEN
 19 WAITFL: EQU     #BD19   ;WARTEN AUF STRAHLRUECKLAUF
 20
 21 ;*********************************
 22 ; SPRITESETZROUTINE
 23
 24 SETSP:  CALL DOTPOS  ;ADRESSE BILDSPEICHER
 25         PUSH HL      ;MERKE ADRESSE
 26         LD     DE,SPEICH ;RETTESPEICHER
 27         LD   B,16    ;16 * 16 PIXEL
 28 LOOP:   PUSH BC      ;AUS BILDSPEICHER
 29         PUSH HL      ;IN FREIEN
 30         LD   BC,8    ;RAM-BEREICH
 31         LDIR         ;RETTEN
 32         POP  HL
 33         CALL NEXTLI  ;ADRESSE DARUNTER
 34         POP  BC
 35         DJNZ LOOP
 36
 37 ;AUSGABE SPRITE
 38
 39         LD   C,16
 40         LD   DE,SPRITE
 41         POP  HL
 42 LOOP3:  PUSH HL
 43         PUSH BC
 44         LD   B,8
 45 LOOP2:  LD   A,(DE)  ;PIXEL,
 46         LD   C,A     ;DIE IM SPRITE
```

```
          OR   (HL)     ;GESETZT SIND,
          SUB  C        ;AUF DEM 'BILDSCHIRM'
          LD   C,A      ;LOESCHEN UND
          INC  DE       ;MIT NEUER
          LD   A,(DE)   ;FARBE
          AND  C        ;VOM SPRITE
          LD   (HL),A   ;SETZEN
          INC  HL
          INC  DE
          DJNZ LOOP2
          POP  BC
          POP  HL
          CALL NEXTLI
          DEC  C
          JR   NZ,LOOP3
          RET
;*************************************
;START DES DEMO-PROGRAMMS

          ENT  $        ;STARTPUNKT FUER 'R'-KOMMANDO
          XOR  A
          CALL MODUS    ;MODE 0
;TEXTAUSGABE
          LD   A,1
          CALL SPALTE
          LD   A,14
          CALL ZEILE
          LD   DE,TEXT ;ZEIGER FUER TEXTAUSGABE
          LD   B,19
STRING:   LD   A,(DE)
          CALL TEXTOU
          LD   A,B
          INC  DE
          CALL PEN
          DJNZ STRING

;STIEFEL UEBER TEXT BEWEGEN

          LD   B,160-16 ;X-RECHTER RAND MINUS SPRITEBREITE
LOOP:     LD   H,0
          LD   L,100
          LD   D,H
          LD   E,B
          PUSH BC
          CALL SETSP ;SPRITE SETZEN

;KURZE WARTESCHLEIFE

          LD   HL,95    ;WERTE VON 1-#FFFF MOEGLICH
PAUSE:    DEC  HL
          LD   A,H
          OR   L
          JR   NZ,PAUSE

          POP  BC
          PUSH BC
          LD   H,0
          LD   L,100
          LD   D,H
          LD   E,B
```

```
107           CALL LOESCH;SPRITE LOESCHEN
108           POP  BC
109           DEC  B
110           DJNZ LOOP4
111           RET
112
113 TEXT:     DEFM "TEST SPRITEPROGRAMM"
114
115 ;*************************************
116 ;SPRITE LOESCHEN
117
118 LOESCH:   CALL DOTPOS
119           LD   DE,SPEICH
120           LD   B,16
121           CALL WAITFL
122 LOOP5:    PUSH BC       ;ALTEN
123           PUSH HL       ;BILDINHALT
124           EX   HL,DE    ;AUS DEM
125           LD   BC,8     ;PUFFER
126           LDIR          ;ZURUECKSCHREIBEN
127           EX   DE,HL
128           POP  HL
129           CALL NEXTLI
130           PIP  BC
131           DJNZ LOOP5
132           RET
133 ;*********************************
134 ;PIXELMASKEN UND -FARBEN FUER
135 ;DEN STIEFEL
136
137 SPRITE:
138           DEFB 0,0,0,0,0,0,0,0
139           DEFB #55,#44,#FF,#DC,0,0,0,0
140           DEFB 0,0,0,0,0,0,#FF,#CC
141           DEFB #FF,#CC,#FF,#DC,#AA,#A3,0,0
142           DEFB 0,0,#55,#44,#FF,#CC,#FF,#CC
143           DEFB #FF,#CC,#FF,#FC,#FF,#FC,0,0
144           DEFB 0,0,#FF,#CC,#FF,#CC,#FF,#CC
145           DEFB #FF,#DC,#FF,#A9,#AA,2,0,0
146           DEFB 0,0,#55,#44,#FF,#CC,#FF,#F3
147           DEFB #FF,#F3,#FF,3,#AA,2,0,0
148           DEFB 0,0,0,0,#FF,#F3,#FF,#F3
149           DEFB #FF,#F3,#FF,#A3,#AA,2,0,0
150           DEFB 0,0,0,0,#55,#51,#FF,#B3
151           DEFB #FF,#F3,#FF,#F3,#AA,2,0,0
152           DEFB 0,0,0,0,0,0,#FF,#F3
153           DEFB #FF,#73,#FF,#F3,#FF,3,0,0
154           DEFB 0,0,0,0,0,0,#55,#51
155           DEFB #FF,#73,#FF,#F3,#FF,#A3,#AA,2
156           DEFB 0,0,0,0,0,0,#FF,#F3
157           DEFB #FF,#F3,#FF,#F3,#FF,#F3,#FF,3
158           DEFB 0,0,#FF,#F3,#FF,#F3,#FF,#F3
159           DEFB #FF,#F3,#FF,#F3,#FF,#A3,#FF,3
160           DEFB #55,#51,#FF,#B3,#FF,#F3,#FF,#F3
161           DEFB #FF,#F3,#FF,#F3,#FF,#B3,#FF,#C0
162           DEFB #FF,#F3,#FF,#73,#FF,#F3,#FF,#F3
163           DEFB #FF,#F3,#FF,#F3,#FF,#E2,#AA,#80
164           DEFB #FF,#F3,#FF,#73,#FF,#F3,#FF,#F3
165           DEFB #FF,#F3,0,0,#FF,#C0,0,0
166           DEFB #FF,#F3,#FF,#F3,#FF,#F3,#FF,#E2
```

```
167          DEFB 0,0,0,0,0,0,0,0
168          DEFB #55,#40,#FF,#C0,#FF,#C0,#AA,#80
169          DEFB 0,0,0,0,0,0,0,0
170
171 ;RESERVIERUNG DES PUFFERS ZUM RETTEN
172 ;DES BILDINHALTS
173 SPEICH: DEFS   16*8
174
175 ;*********************************
176 ; ENDE DES PROGRAMMS
```

Als nächsten Schritt kann das Programm auf syntaktische Fehler überprüft werden. Dazu ist am einfachsten der Assembler mit 'A' und der Option 4 (Unterdrücken des Listens) aufzurufen. Wenn das Listing ordentlich eingegeben, wurde müßten folgende Fehler angezeigt werden:

*ERROR* 04 in Zeile 86 (Marke mehrfach definiert)
(Beseitigung: Aufruf 'E86' und Korrektur der Marke in 'LOOP4:')

*ERROR* 01 in Zeile 124 (Aufbau der Zeile falsch)
(Beseitigung: Aufruf 'E124' und Korrektur der Operandensyntax in 'EX DE,HL')

*ERROR* 02 in Zeile 130 (Befehl wurde nicht erkannt)
(Beseitigung: Aufruf 'E130' und Korrektur der Syntax in 'POP BC')

*WARNING* LOOP4 absend (LOOP4 konnte wegen der falschen Marke in Zeile 86 nicht gefunden werden)
(Beseitigung: ergibt sich aus Korrektur der Zeile 86)

Wenn alle syntaktischen Fehler beseitigt sind, sollte die Datei mit 'P1,176,STIEFEL' auf Band gerettet werden. Nach dem Retten und einem erneuten Assemblerlauf (nach der Fehlerbeseitigung) kann das Programm mit 'R' gestartet werden.
Zu sehen ist, daß das Programm noch logische Fehler enthält, denn an den Stellen auf dem Bild, wo der Sprite dargestellt werden soll, wird nur eine rechteckige Fläche gelöscht. Der Fehler liegt in der Zeile 52. Hier wurde an Stelle eines OR oder XOR ein AND verwendet. Dieser Fehler ist mit 'E' oder Zeilenneueingabe zu beseitigen. Nach der Beendigung der Routine ist MODE 0 eingestellt. Mit 'W' kann auf MODE 0 oder 1 zurückgestellt werden.
Es wäre denkbar, nach kleineren Änderungen die Spritesetz- und -löschroutine als RSX-Kommandos auch für die Nutzung von BASIC aus nutzbar zu machen.

# 6. L i t e r a t u r h i n w e i s e

/1/ Systemhandbuch KC compact. VEB Mikroelektronik "Wilhelm Pieck" Mühlhausen

/2/ Classen, L.: Programmierung des Mikroprozessorsystems U880-K1520. Reihe Automatisierungstechnik. VEB Verlag TEchnik, Berlin 1982

/3/ Schwarze, W.; Meyer, G.; Eckard, D: Mikrorechner. Wirkungsweise Programmierung, Applikation. VEB Verlag Technik, Berlin 1980

/4/ Robotron-Systemdokumentation MOS K1520. SCPX 1526, Anleitung für den Programmierer Teil II. VEB Robotron Buchungsmaschinenwerk, Karl-Marx-Stadt 1984.

/5/ Lampe, D.; Jorke, G.; Wengel,N.: Algorithmen der Mikrorechentechnik. VEB Verlag Technik, Berlin 1983.

/6/ Z80-Assemblersprache. Benutzerhandbuch in deutscher Sprache. ZILOG, 1977.

/7/ Klein, M.; Klein, R.-D.: Z80-Applikationsbuch. Franzisverlag GmbH. München 1983.

/8/ Osborn, A.: Einführung in die Mikrocomputer-Technik. te-wi Verlag GmbH, München 1982.

/9/ Barthold, H.; Bäurich, H.: Mikroprozessoren- Mikroelektronische Schaltkreise und ihre Anwendung. Reihe electronica Bde 186/188 oder 202/204 bzw. Nachauflagen. Berlin: Militärverlag der DDR.

/10/ Befehlsbeschreibung U880D. VEB Mikroelektronik "Karl-Marx" Erfurt, Stammbetrieb.

/11/ Mikroprozessorsystem der II. Leistungsklasse. VEB Mikroelektronik "Karl Marx" Erfurt

/12/ Kieser, H.; Meder M.: Mikroprozessortechnik. VEB Verlag Technik, Berlin 1985

# A n h a n g A

## Befehlscode-Tabelle

### 8-Bit-Ladebefehle

| | A | B | C | D | E | H | L | (HL) | (BC) | (DE) | (nn) | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LD A,. | 7F | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 0A | 1A | 3AXXXX | 3EXX |
| LD B,. | 47 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | | | | 06XX |
| LD C,. | 4F | 48 | 49 | 4A | 4B | 4C | 4D | 4E | | | | 0EXX |
| LD D,. | 57 | 50 | 51 | 52 | 53 | 55 | 55 | 56 | | | | 16XX |
| LD E,. | 5F | 58 | 59 | 5A | 5B | 5C | 5D | 5E | | | | 1EXX |
| LD H,. | 67 | 60 | 61 | 62 | 63 | 66 | 65 | 66 | | | | 26XX |
| LD L,. | 6F | 68 | 69 | 6A | 6B | 6C | 6D | 6E | | | | 2EXX |
| LD (HL),. | 77 | 70 | 71 | 72 | 73 | 77 | 75 | | | | | 36XX |
| LD (BC),. | 02 | | | | | | | | | | | |
| LD (DE),. | 12 | | | | | | | | | | | |
| LD (nn),. | 32XXXX | | | | | | | | | | | |

| | A | B | C | D | E | H | L |
|---|---|---|---|---|---|---|---|
| LD .,(IX+d) | DD7EXX | DD46XX | DD4EXX | DD56XX | DD5EXX | DD66XX | DD6EXX |
| LD .,(IY+d) | FD7EXX | FD46XX | FD4EXX | FD56XX | FD5EXX | FD66XX | FD6EXX |
| LD (IX+d),. | DD77XX | DD70XX | DD71XX | DD72XX | DD73XX | DD74XX | DD75XX |
| LD (IY+d),. | FD77XX | FD70XX | FD71XX | FD72XX | FD73XX | FD74XX | FD75XX |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| LD (IX+d),n | DD36XXXX | | | | LD (IY+d),n | FD36XXXX | | |

| | | S | Z | H | P/V | N | C |
|---|---|---|---|---|---|---|---|
| LD A,I | ED57 | * | * | 0 | F | 0 | - |
| LD A,R | ED5F | * | * | 0 | F | 0 | - |
| LD I,A | ED47 | - | - | - | - | - | - |
| LD R,A | ED4F | - | - | - | - | - | - |

### 16-Bit-Ladebefehle

| | BC | DE | HL | SP | IX | IY |
|---|---|---|---|---|---|---|
| LD ..,nn | 01XXXX | 11XXXX | 21XXXX | 31XXXX | DD21XXXX | FD21XXXX |
| LD ..,(nn) | ED4BXXXX | ED5BXXXX | 2AXXXX | ED7BXXXX | DD2AXXXX | FD2AXXXX |
| LD (nn),.. | ED43XXXX | ED53XXXX | 22XXXX | ED73XXXX | DD22XXXX | FD22XXXX |
| LD SP | | | F9 | | DDF9 | FDF9 |

| | BC | DE | HL | AF | IX | IY |
|---|---|---|---|---|---|---|
| PUSH .. | C5 | D5 | E5 | F5 | DDE5 | FDE5 |
| POP .. | C1 | D1 | E1 | F1 | DDE1 | FDE1 |

## Registeraustauschbefehle

| | | | | | |
|---|---|---|---|---|---|
| EX (SP),HL | E3 | EX DE,HL | EB | | |
| EX (SP),IX | DDE3 | EX AF,AF' | 08 | | |
| EX (SP),IY | FDE3 | EXX | D9 | BC-BC' DE-DE' HL-HL' | |

## Blocktransfer- und -suchbefehle

| | | S | Z | H | P/V | N | C | |
|---|---|---|---|---|---|---|---|---|
| LDI | EDA0 | - | - | 0 | * | 0 | - | LD(DE),(HL);INC HL;INC DE;DEC BC |
| LDIR | EDB0 | - | - | 0 | 0 | 0 | - | wie LDI, wiederholen bis BC=0 |
| LDD | EDA8 | - | - | 0 | * | 0 | - | LD(DE),(HL);DEC HL;DEC DE;DEC BC |
| LDDR | EDB8 | - | - | 0 | 0 | 0 | - | wie LDD, wiederholen bis BC=0 |
| CPI | EDA1 | * | * | * | * | 1 | - | CP A,(HL);INC HL;DEC BC |
| CPIR | EDB1 | * | * | * | * | 1 | - | wie CPI,wiederh.b.BC=0 od.A=(HL) |
| CPD | EDA9 | * | * | * | * | 1 | - | CP A,(HL);DEC HL;DEC BC |
| CPDR | EDB9 | * | * | * | * | 1 | - | wie CPD,wiederh.b.BC=0 od.A=(HL) |

## 8-Bit-Arithmetik- und -Logikbefehle

| | B | C | D | E | H | L | (HL) | A | n | (IX+d) | (IY+d) | S | Z | H | P/V | N | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD . | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | C6XX | DD86XX | FD86XX | * | * | * | V | 0 | * |
| ADC . | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | CEXX | DD8EXX | FD8EXX | * | * | * | V | 0 | * |
| SUB . | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | D6XX | DD96XX | FD96XX | * | * | * | V | 1 | * |
| SBC . | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | DEXX | DD9EXX | FD9EXX | * | * | * | V | 1 | * |
| AND . | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | E6XX | DDA6XX | FDA6XX | * | * | 1 | P | 0 | 0 |
| XOR . | A8 | A9 | AA | AB | AC | AD | AE | AF | EEXX | DDAEXX | FDAEXX | * | * | 1 | P | 0 | 0 |
| OR . | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | F6XX | DDB6XX | FDB6XX | * | * | 1 | P | 0 | 0 |
| CP . | B8 | B9 | BA | BB | BC | BD | BE | BF | FEXX | DDBEXX | FDBEXX | * | * | * | V | 1 | * |
| INC . | 04 | 0C | 14 | 1C | 24 | 2C | 34 | 3C | | DD34XX | FD34XX | * | * | * | V | 0 | - |
| DEC . | 05 | 0D | 15 | 1D | 25 | 2D | 35 | 3D | | DD35XX | FD35XX | * | * | * | V | 1 | - |

| | | S | Z | H | P/V | N | C | |
|---|---|---|---|---|---|---|---|---|
| DAA | 27 | * | * | * | P | - | * | BCD-Korrektur im Akku |
| CPL | 2F | - | - | 1 | - | 1 | - | Komplementiere Akku (1er-Komplement) |
| SCF | 37 | - | - | 0 | - | 0 | 1 | Setze Carry-Flag |
| CCF | 3F | - | - | x | - | 0 | * | Komplementiere Carry-Flag |
| NEG | ED44 | * | * | * | V | 1 | * | Komplementiere Akku (2er-Komplement) |

## 16-Bit-Arithmetikbefehle

| | BC | DE | HL | SP | IX | IY | S | Z | H | P/V | N | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD HL,.. | 09 | 19 | 29 | 39 | | | - | - | x | - | 0 | * |
| ADC HL,.. | ED4A | ED5A | ED6A | ED7A | | | * | * | x | V | 0 | * |
| SBC HL,.. | ED42 | ED52 | ED62 | ED72 | | | * | * | x | V | 1 | * |

```
ADD IX,.. DD09 DD19      DD39 DD29       - - x  -  0 *
ADD IY,.. FD09 FD19      FD39      FD29  - - x  -  0 *
INC ..    03   13   23   33   DD23 FD23  - - -  -  - -
DEC ..    0B   1B   2B   3B   DD2B FD2B  - - -  -  - -
```

Sprung- und Unterprogrammbefehle

```
      Z      NZ     C      NC     PE     PO     M      P
JP    CAXXXX C2XXXX DAXXXX D2XXXX EAXXXX E2XXXX FAXXXX F2XXXX
CALL  CCXXXX C4XXXX DCXXXX D4XXXX ECXXXX E4XXXX FCXXXX F4XXXX
RET   C8     C0     D8     D0     E8     E0     F8     F0
JR    28XX   20XX   38XX   30XX
```

```
     unbedingt (HL) (IX) (IY)     RST  00 08 10 18 20 28 30 38
JP    C3XXXX    E9  DDE9 FDE9          C7 CF D7 DF E7 EF F7 FF
CALL  CDXXXX
JR    18XX
```

```
DJNZ  10XX   DEC B;JR NZ,nn
RETI  ED4D   zurück vom Interrupt
RETN  ED45   zurück vom nicht maskierbaren Interrupt
```

Rotations- und Verschiebebefehle

```
             S Z H P/V N C
RLCA  07     - - 0  -  0 *   Rotiere Akku links
RRCA  0F     - - 0  -  0 *   Rotiere Akku rechts
RLA   17     - - 0  -  0 *   Rotiere Akku links durch Carry
RRA   1F     - - 0  -  0 *   Rotiere Akku rechts durch Carry
RLD   ED6F   * * 0  P  0 -   Rot.Ziffern links zw. Akku und (HL)
RRD   ED67   * * 0  P  0 -   Rot.Ziffern rechts zw.Akku und (HL)
```

```
      B    C    D    E    H    L    (HL) A    (IX+d)   (IY+d)
RLC   CB00 CB01 CB02 CB03 CB04 CB05 CB06 CB07 DDCBXX06 FDCBXX06
RRC   CB08 CB09 CB0A CB0B CB0C CB0D CB0E CB0F DDCBXX0E FDCBXX0E
RL    CB10 CB11 CB12 CB13 CB14 CB15 CB16 CB17 DDCBXX16 FDCBXX16
RR    CB18 CB19 CB1A CB1B CB1C CB1D CB1E CB1F DDCBXX1E FDCBXX1E
SLA   CB20 CB21 CB22 CB23 CB24 CB25 CB26 CB27 DDCBXX26 FDCBXX26
SRA   CB28 CB29 CB2A CB2B CB2C CB2D CB2E CB2F DDCBXX2E FDCBXX2E
SRL   CB38 CB39 CB3A CB3B CB3C CB3D CB3E CB3F DDCBXX3E FDCBXX3E
```

```
        S Z H P/V N C
RLC/RRC * * 0  P  0 *  Rotiere Register links/rechts
RL/RR   * * 0  P  0 *  Rotiere Register links/rechts durch Carry
SLA/SRA * * 0  P  0 *  Schiebe Register links/rechts arithm.
SRL     * * 0  P  0 *  Schiebe Register rechts logisch
```

## Einzelbitbefehle

| | B | C | D | E | H | L | (HL) | A | (IX+d) | (IY+d) |
|---|---|---|---|---|---|---|---|---|---|---|
| BIT 0 | CB40 | CB41 | CB42 | CB43 | CB44 | CB45 | CB46 | CB47 | DDCBXX46 | FDCBXX46 |
| BIT 1 | CB48 | CB49 | CB4A | CB4B | CB4C | CB4D | CB4E | CB4F | DDCBXX4E | FDCBXX4E |
| BIT 2 | CB50 | CB51 | CB52 | CB53 | CB54 | CB55 | CB56 | CB57 | DDCBXX56 | FDCBXX56 |
| BIT 3 | CB58 | CB59 | CB5A | CB5B | CB5C | CB5D | CB5E | CB5F | DDCBXX5E | FDCBXX5E |
| BIT 4 | CB60 | CB61 | CB62 | CB63 | CB64 | CB65 | CB66 | CB67 | DDCBXX66 | FDCBXX66 |
| BIT 5 | CB68 | CB69 | CB6A | CB6B | CB6C | CB6D | CB6E | CB6F | DDCBXX6E | FDCBXX6E |
| BIT 6 | CB70 | CB71 | CB72 | CB73 | CB74 | CB75 | CB76 | CB77 | DDCBXX76 | FDCBXX76 |
| BIT 7 | CB78 | CB79 | CB7A | CB7B | CB7C | CB7D | CB7E | CB7F | DDCBXX7E | FDCBXX7E |
| RES 0 | CB80 | CB81 | CB82 | CB83 | CB84 | CB85 | CB86 | CB87 | DDCBXX86 | FDCBXX86 |
| RES 1 | CB88 | CB89 | CB8A | CB8B | CB8C | CB8D | CB8E | CB8F | DDCBXX8E | FDCBXX8E |
| RES 2 | CB90 | CB91 | CB92 | CB93 | CB94 | CB95 | CB96 | CB97 | DDCBXX96 | FDCBXX96 |
| RES 3 | CB98 | CB99 | CB9A | CB9B | CB9C | CB9D | CB9E | CB9F | DDCBXX9E | FDCBXX9E |
| RES 4 | CBA0 | CBA1 | CBA2 | CBA3 | CBA4 | CBA5 | CBA6 | CBA7 | DDCBXXA6 | FDCBXXA6 |
| RES 5 | CBA8 | CBA9 | CBAA | CBAB | CBAC | CBAD | CBAE | CBAF | DDCBXXAE | FDCBXXAE |
| RES 6 | CBB0 | CBB1 | CBB2 | CBB3 | CBB4 | CBB5 | CBB6 | CBB7 | DDCBXXB6 | FDCBXXB6 |
| RES 7 | CBB8 | CBB9 | CBBA | CBBB | CBBC | CBBD | CBBE | CBBF | DDCBXXBE | FDCBXXBE |
| SET 0 | CBC0 | CBC1 | CBC2 | CBC3 | CBC4 | CBC5 | CBC6 | CBC7 | DDCBXXC6 | FDCBXXC6 |
| SET 1 | CBC8 | CBC9 | CBCA | CBCB | CBCC | CBCD | CBCE | CBCF | DDCBXXCE | FDCBXXCE |
| SET 2 | CBD0 | CBD1 | CBD2 | CBD3 | CBD4 | CBD5 | CBD6 | CBD7 | DDCBXXD6 | FDCBXXD6 |
| SET 3 | CBD8 | CBD9 | CBDA | CBDB | CBDC | CBDD | CBDE | CBDF | DDCBXXDE | FDCBXXDE |
| SET 4 | CBE0 | CBE1 | CBE2 | CBE3 | CBE4 | CBE5 | CBE6 | CBE7 | DDCBXXE6 | FDCBXXE6 |
| SET 5 | CBE8 | CBE9 | CBEA | CBEB | CBEC | CBED | CBEE | CBEF | DDCBXXEE | FDCBXXEE |
| SET 6 | CBF0 | CBF1 | CBF2 | CBF3 | CBF4 | CBF5 | CBF6 | CBF7 | DDCBXXF6 | FDCBXXF6 |
| SET 7 | CBF8 | CBF9 | CBFA | CBFB | CBFC | CBFD | CBFE | CBFF | DDCBXXFE | FDCBXXFE |

Flagbeeinflussung:

| | S | Z | H | P/V | N | C | |
|---|---|---|---|---|---|---|---|
| BIT | x | * | 1 | x | 0 | - | Komplement des Bits in Z |
| RES | - | - | - | - | - | - | 0 in Bit |
| SET | - | - | - | - | - | - | 1 in Bit |

## CPU-Steuerbefehle

| | | S | Z | H | P/V | N | C | |
|---|---|---|---|---|---|---|---|---|
| NOP | 00 | - | - | - | - | - | - | Leerbefehl |
| HALT | 76 | - | - | - | - | - | - | NOP bis RESET oder Interrupt |
| DI | F3 | - | - | - | - | - | - | Interrupts sperren |
| EI | FB | - | - | - | - | - | - | Interrupts freigeben |
| IM 0 | ED46 | - | - | - | - | - | - | Interrupt-Modus 0 |
| IM 1 | ED56 | - | - | - | - | - | - | Interrupt-Modus 1 |
| IM 2 | ED5E | - | - | - | - | - | - | Interrupt-Modus 2 |

## Ein- und Ausgabebefehle

| | A | B | C | D | E | H | L | S | Z | H | P/V | N | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IN .,(C) | ED78 | ED40 | ED48 | ED50 | ED58 | ED60 | ED68 | * | * | * | P | 0 | - |
| OUT (C),. | ED79 | ED41 | ED49 | ED51 | ED59 | ED61 | ED69 | - | - | - | - | - | - |

| | | S | Z | H | P/V | N | C | |
|---|---|---|---|---|---|---|---|---|
| IN A,(n) | DBXX | - | - | - | - | - | - | |
| OUT (n),A | D3XX | - | - | - | - | - | - | |
| INI | EDA2 | x | * | x | x | 1 | - | IN (HL),(C);INC HL;DEC B |
| INIR | EDB2 | x | 1 | x | x | 1 | - | wie INI,wiederholen solange B >0 |
| IND | EDAA | x | * | x | x | 1 | - | IN (HL),(C);DEC HL;DEC B |
| INDR | EDBA | x | 1 | x | x | 1 | - | wie IND,wiederholen solange B >0 |
| OUTI | EDA3 | x | * | x | x | 1 | - | OUT (C),(HL);INC HL;DEC B |
| OTIR | EDB3 | x | 1 | x | x | 1 | - | wie OUTI,wiederholen solange B >0 |
| OUTD | EDAB | x | * | x | x | 1 | - | OUT (C),(HL);DEC HL;DEC B |
| OTDR | EDBB | x | 1 | x | x | 1 | - | wie OUTD,wiederholen solange B >0 |

## Flag-Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | S | Z | X | H | X | P/V | N | C |

| | gesetzt | nicht gesetzt | wird bei |
|---|---|---|---|
| C Carry-Flag | C | NC | Übertrag von Bit 7 |
| N Add-/Subtract-Flag | | | Subtraktion |
| P/V Parity-/Overflow-Flag | PE | PO | gerader Parität |
| H Half-Carry-Flag | | | Übertrag von Bit 3 |
| Z Zero-Flag | Z | NZ | Ergebnis 0 |
| S Sign-Flag | M | P | negatives Ergebnis |

X nicht verwendet

Beeinflussung:

- \- unverändert
- 1 gesetzt
- 0 zurückgesetzt
- * entsprechend dem Ergebnis der Operation
  (gesetzt wenn erfüllt
  zurückgesetzt wenn nicht erfüllt)
- x unbestimmt
- V Overflow-Funktion
- P Parity-Funktion
- F Inhalt des Interrupt-Flip-Flops IFF2

# A n h a n g B

Pseudobefehle des Assemblers:
---

```
        ORG     nn      - Adreßzählerzuordnung
Marke:  EQU     nn      - Wertzuweisung für Marke
        DEFB    n       - Legt n auf die nächste Speicherstelle
        DEFW    nn      - Legt nn auf die nächsten zwei Speicher-
                          stellen (dabei zuerst niederwertiger,
                          dann höherwertiger Teil)
        DEFM    "Text"  - Legt auf die nächsten Speicherstellen
                          die ASCII-Werte der Zeichen des Textes
        DEFS    nn      - reserviert  nn  Speicherplätze und  be-
                          schreibt diese mit Nullen
        ENT             - legt Startadresse für 'R' fest
```

Dabei sind:

```
        n   -  8-Bit-Konstante
        nn  -  16-Bit-Konstante
```

bedingte Pseudobefehle:
---

```
IF ausdruck    - in  Abhängigkeit von 'ausdruck'  wird  folgender
                 Quelltext  bis END bzw.  ELSE  assembliert  oder
                 nicht
ELSE           - in  Abhängigkeit  von  'ausdruck' nach  IF  wird
                 nachfolgender Quelltext bis END assembliert oder
                 nicht
END            - Beenden der bedingten Assemblierung
```

Assemblerkommandos
---

```
*L-            - Auflistung unterdrücken
*L+            - Auflistung starten
*D+            - Befehlszähler dezimal ausgeben
*D-            - Befehlszähler hexadezimal ausgeben
*E             - Ausgabe drei Leerzeilen bzw. Blattvorschub
*Hs            - String s wird als Kopfzeile definiert
*S             - Stoppen der Assemblerliste
*T+s           - Objektcode auf Band ausgeben
*T-            - Beenden der Bandausgabe
*F s           - Quelltext vom Band assemblieren
```

# A n h a n g C

Übersicht ASSMON1 und ASSMON2

ASSMON1
---

| | |
|---|---|
| Assemble | - Aufruf des Assemblers |
| Bye | - Rücksprung zum BASIC |
| Current state | - Ausgabe der aktuellen Argumente |
| Delete | - Löschen von Quelltextzeilen |
| Edit | - Aufruf des Editors |
| Find | - Durchsuchen von Quelltext nach einer Zeichenkette |
| Get text | - Quelltext von Band einlesen |
| Help | - Aufruf des Hilfsmenüs |
| Insert | - Eingabe von Quelltext |
| CTRL/J -> ASSMON2 | - Sprung in ASSMON2 |
| List | - Auflisten von Quelltext |
| Move | - Kopieren von Quelltextzeileninhalten |
| reNumber | - Neunummerierung von Quelltextzeilen |
| Objekt | - assemblierten Objektcode auf Band retten |
| Put text | - Quelltext auf Band retten |
| Q put ASCII | - Quelltext als reine ASCII-Datei auf Band retten |
| Run | - Start eines assemblierten Objektcodes |
| Separator | - Eingabe eines neuen Trennzeichen für die Argumenteingabe |
| Tape speed | - Umschalten zwischen 1000 und 2000 Baud Übertragungsrate bei der Bandausgabe |
| Upper line | - Ausgabe der Nummer der letzten verwendeten Quelltextzeile |
| Verify | - Überprüfung der Bandaufzeichnung |
| Width | - Umschalten zwischen 40 und 80 Zeichen pro Zeile |
| teXt info | - dezimale Anzeige des Quelltextanfanges- und -endadresse |
| Y print lenght | - Festlegen der Zeilenzahl pro Seite bei Druckerausgabe |
| Z print text | - Ausgabe von Quelltext auf dem Drucker |

ASSMON2
---

| | |
|---|---|
| Cursor rechts | - SZ um eins erhöhen |
| Cursor links | - SZ um eins vermindern |
| Cursor runter | - SZ um acht erhöhen |
| Cursor hoch | - SZ um acht vermindern |
| G | - Suchen nach Zeichenkette |

```
H                 - Umwandlung Dezimalzahl in Hexadezimalzahl
I                 - Kopieren eines Speicherblocks
J                 - Ausführung von Objektcode in Echtzeit
L                 - Hex-Dump auf Bildschirm ausgeben
M                 - SZ setzen
N                 - Fortsetzen der Suche nach der mit 'G' einge-
                    gebenen Zeichenkette
O                 - SZ = SZ + (SZ)
P                 - Füllen eines Speicherblocks mit Byte
R                 - Objektcode von Band lesen
S                 - SZ = Adresse die im Stack liegt
T                 - Speicherblock disassemblieren
U                 - SZ wieder auf den Wert vor Aufruf 'O'
                    bringen
V                 - SZ wieder auf den Wert vor Aufruf 'X'
                    bringen
W                 - Ausgabe eines Speicherblocks auf Magnetband
X                 - SZ = (SZ) + (SZ+1)
Y                 - Eingabe einer ASCII-Zeichenkette
>                 - Unterbrechungspunkt nach laufenden Befehl
                    setzen und '^CTRL_-C' aufrufen
!                 - Unterbrechungspunkt bei SZ setzen
.                 - Registerzeiger bzw. Registerinhalt ändern
^CTRL_-^A_        - eine Seite disassemblieren
^CTRL_-^C_        - Fortsetzung der Programmausführung
^CTRL_-^J_        - Sprung in ASSMON1
^CTRL_-^L_        - Hex-Dump auf Drucker ausgeben
^CTRL_-^S_        - Einzelschritt
^CTRL_-^X_        - Rücksprung zu BASIC

Erklärung:   SZ   = Speicherzeigeradresse
             (SZ) = Inhalt der Speicherzeigeradresse
```

# A n h a n g D

## Fehlermeldungen

*ERROR* 1 Aufbau der Zeile ist fehlerhaft
*ERROR* 2 unbekannte Mnemonik
*ERROR* 3 Statement schlecht geformt
*ERROR* 4 Name mehrfach definiert
*ERROR* 5 Zeile enthält ein unerlaubtes Zeichen
*ERROR* 6 ein Operand in dieser Zeile ist nicht erlaubt
*ERROR* 7 Name ist ein reserviertes Wort
*ERROR* 8 falsche Registerbenutzung
*ERROR* 9 zu viele Register in der Zeile
*ERROR* 10 der WERT eines Ausdrucks ist unzulässig groß (8-Bit-Wort ist maximal zulässig)
*ERROR* 11 die Befehle JP (IX+n) und JP (IY+n) sind nicht erlaubt
*ERROR* 12 Fehler bei der Anwendung einer Assembleranweisung
*ERROR* 13 unzulässige Vorwärtsreferenz, z.B. wenn in einem EQU-Befehl eine Marke als Operand auftritt, die vorher nicht definiert wurde
*ERROR* 14 Division durch Null
*ERROR* 15 Überlauf in einer Multiplikationsoperation
*ERROR* 19 verschachteltes, bedingtes Statement

Bad ORG! Eine ORG-Anweisung würde mit der zugewiesenen Adresse ASSMON1, sein Textfile oder die Markentabelle überschreiben. Die Kontrolle geht an den Editor zurück.

Out of Table space Tritt während des 1. Laufes auf, wenn der reservierte Platz für die Markentabelle nicht ausreicht. Die Kontrolle geht an den Editor zurück.

Bad Memory! Es ist kein Platz für weiteren Quelltext vorhanden. In diesem Fall sollte die gesamte Textdatei oder ein Teil davon auf Band gerettet werden

Warnungen:

** File closed ** während mit 'T+' code auf Band ausgegeben wird tritt eine ORG-Anweisung auf

*WARNING* label absend Name nicht definiert

# A n h a n g E

## Reservierte Wörter und gültige Befehls-Mnemonik

Reservierte Wörter dürfen Teil eines Namens, nicht aber selber Namen sein.

```
  A   B   C   D   E   H
  L   I   R   $  AF  AF'
 BC  DE  HL  IX  IY  SP
 NC   Z  NZ   M   P  PE
 PO
```

gültige Mnemonik:

```
ADC  ADD  AND  BIT  CALL CCF  CP   CPD
CPDR CPI  CPIR CPL  DAA  DEC  DI   DJNZ
EI   EX   EXX  HALT IM   IN   INC  IND
INDR INI  INIR JP   JR   LD   LDD  LDDR
LDI  LDIR NEG  NOP  OR   OTDR OTIR OUT
OUTD OUTI POP  PUSH RES  RET  RETI RETN
RL   RLA  RLC  RLCA RLD  RR   RRA  RRC
RRCA RRD  RST  SBC  SCF  SET  SLA  SRA
SRL  SUB  XOR

DEFB DEFM DEFS DEFW ELSE END  ENT  EQU
IF   ORG
```