Z 8 0 - A s s e m b l e r

B e n u t z e r h a n d b u c h

V e r s i o n 1 . 4 h

2 7 . 0 6 . 1 9 8 5

Anhang

Tabellen-Verzeichnis

1 Voraussetzungen für den Betrieb des Z80-Assemblers

Der Assembler benötigt einen Z80 oder einen kompatiblen Prozessor (es werden nur die von Zilog erwähnten Instruktionen verwendet). Es müssen mindestens 48 KByte Speicher zur Verfügung stehen, um sinnvoll arbeiten zu können. Mindestens ein Diskettenlaufwerk muß vorhanden sein. Für die Assemblierung von Programmen mit sehr vielen Labels ist eine RAM-Disk oder zumindest ein weiteres Laufwerk empfehlenswert, da unter Umständen die Symbol table auf 'Diskette' ausgelagert werden muß. Der Assembler läuft unter folgenden Betriebssystemen (und kompatiblen):

(a) NEWDOS/80 (TRS-80, Model I, im folgenden einfach TRS-80 genannt)
(b) CP/M

Für die jeweiligen Betriebssysteme sind verschiedene Versionen des Assemblers zu benutzen!
Weiter ist ein Texteditor (z.B. WordStar, SCRIPSIT) nötig, womit die Quelldateien erstellt werden können. Dieser Texteditor muß eine ASCII-Datei erzeugen, bei der die Zeilen durch CR (TRS-80) bzw. CR/LF (CP/M) abgeschlossen werden. Bit 7 der einzelnen Zeichen wird vom Assembler ignoriert (Ausnahme: Zeilennummer, hier muß Bit 7 gesetzt sein).
Sollen REL-Dateien verwendet werden, so ist der entsprechende Linker von Microsoft oder ein hierzu kompatibler zu verwenden.
Für die Verwendung von RLD-Dateien ist der entsprechende Linker von Computer Applications Unlimited erforderlich.
CRL-Dateien erfordern den BDS C - Linker.
Für die Umwandlung von HEX- in COM-Dateien kann LOAD, DDT, SID oder ein ähnliches Programm benutzt werden.
Versionen für weitere Betriebssysteme können erstellt werden (auf Anfrage).

2 Bedienung des Assemblers

2.1 Aufruf des Assemblers

Es gibt zwei Möglichkeiten den Assembler aufzurufen:

(1) Z80 <RETURN>

oder

(2) Z80 Kommandos <RETURN>

Im Fall (1) erscheint ein '*' (Prompt) und der Assembler wartet auf die Eingabe einer Kommandozeile. Nach dem Abarbeiten der Kommandos verlangt der Assembler eine neue Eingabe (bis CTRL-C (CP/M), BREAK (TRS-80) oder /Q eingegeben wird).

Im Fall (2) arbeitet der Assembler die Kommandos ab und springt dann zurück ins Betriebssystem.

Anmerkung:

In der CP/M-Version werden die Switches durch '/' markiert, in der TRS-80-Version durch '-':
Im Fall (1) läßt sich der Assembler durch Eingabe von /Q (TRS-80: -Q) verabschieden. Im folgenden wird üblicherweise '/' verwendet.

2.2 Die Kommandos

Kommandos werden durch Strichpunkte getrennt. Ein Semikolon am Ende einer Kommandozeile bewirkt, daß der Assembler den Prompt '*' ausgibt und erneut eine Kommandozeile verlangt.

Es sind folgende Kommandos möglich (in Klammern gesetzte Parameter können weggelassen werden):

Switches	Nur Switches.
Datei (Switches)	Kurzaufruf.
/Q	Abbruch.
Datei/E	Fehlerdatei öffnen.
/N	Fehlerdatei schließen.
(Object-Datei)(,Listing-Datei)=Quelldatei (switches)	Allgemeiner Aufruf.

2.2.1 Kurzaufruf

Beim Kurzaufruf sind die Bezeichnungen der Quell- und der Object-Datei identisch. Der Dateityp (falls angegeben) wird für die Quell-Datei verwendet. Für die Object-Datei wird immer der Standarddateityp eingesetzt, die Laufwerksangabe bezieht sich auf beide Dateien. Beispiele:

TEST	entspricht	TEST=TEST
XYZ/I	entspricht	XYZ=XYZ/I
BEISPIEL.SRC	entspricht	BEISPIEL=BEISPIEL.SRC

2.2.2 Fehlerdatei

In der Fehlerdatei (sofern geöffnet) werden sämtliche Fehler, die während des Assemblierens auftreten, vermerkt. Dies ist für den Aufruf des Assemblers mit mehreren Dateien gedacht. Sollen keine Fehler mehr gespeichert werden, so ist die Fehlerdatei mit /N zu schließen. Vor dem Verlassen des Assemblers mit CTRL-C ist /N einzugeben, da sonst die Fehlerdatei nicht geschlossen wird! Beim Verlassen des Assemblers mit /Q wird die Fehlerdatei automatisch geschlossen. In der TRS-80-Version des Assemblers wird die Fehlerdatei immer automatisch geschlossen.

2.2.3 Allgemeiner Aufruf

Beim allgemeinen Aufruf muß eine Quelldatei, kann eine Listing-Datei und kann eine Object-Datei angegeben werden. Zuerst wird die Object-Datei angegeben. Danach folgt der Name der Listing-Datei, von einem Komma eingeleitet. Vor die Quelldateibezeichnung wird ein Gleichheitszeichen gesetzt. Die Dateibezeichnungen können hier, anders als beim Kurzaufruf, verschieden sein. Falls kein Dateityp vorhanden ist, wird der Standarddateityp eingesetzt:

	CP/M	TRS-80
Object-Datei:	REL COM HEX CMD RLD CRL	REL COM HEX CMD RLD CRL
Listing-Datei:	PRN	LST
Quelldatei:	MAC	MAC
Fehlerdatei:	ERR	ERR

Tabelle 2-1: Standarddateitypen

Statt der Quelldateibezeichnung kann TTY: (Eingabe von Tastatur) angegeben werden. Statt der Listingdateibezeichnung kann TTY: (Ausgabe auf Bildschirm) oder LST: (Ausgabe auf Drucker) angegeben werden. (TRS-80-Version: *KI für Tastatur, *DO für Bildschirm, *PR für Drucker.) Die Eingabe von der Tastatur hat einige wenige Einschränkungen zur Folge, die bei den entsprechenden Statements vermerkt sind.

2.2.4 Dateibezeichnungen

CP/M-Version:
Dateibezeichnungen bestehen aus einer optionalen Laufwerksangabe, dem Dateinamen und dem optionalen Dateityp. Die Laufwerksangabe (A..P) wird durch einen Doppelpunkt vom Dateinamen getrennt. Der Dateityp besteht aus einem bis drei erlaubten Zeichen und wird mit einem Punkt vom Dateinamen getrennt. Erlaubte Zeichen in Dateinamen sind:

A..Z, a..z, 0..9, $, -, +

Wenn kein Dateityp angegeben wird, wird der Standarddateityp eingesetzt (siehe oben). Wird kein Laufwerk angegeben, wird das momentan selektierte Laufwerk benutzt.

TRS-80-Version:
Dateibezeichnungen bestehen aus dem Dateinamen, dem optionalen Dateityp und einer optionalen Laufwerksangabe. Der Dateityp besteht aus einem bis drei erlaubten Zeichen, wobei das erste ein Buchstabe sein muß. Er wird durch einen Schrägstrich (/) vom Dateinamen getrennt. Die Laufwerksangabe (0..3) wird durch einen Doppelpunkt vom vorhergehenden Dateinamen oder Dateityp getrennt. Erlaubte Zeichen sind:

A..Z, a..z, 0..9

Wenn kein Dateityp angegeben ist, wird der Standarddateityp eingesetzt (siehe oben). Bei fehlender Laufwerksangabe werden alle angeschlossenen Laufwerke durchsucht.

2.2.5 Beispiele für erlaubte Kommandos

```
TEST.OBJ,TEST.LST=TEST.SRC/L                    (CP/M)
```

Die Datei TEST.SRC wird assembliert. Der Object-Code wird in die Datei TEST.OBJ geschrieben, TEST.LST erhält das Listing, welches mit Zeilennummern versehen wird.

```
1234/X                                          (CP/M)
```

Die Datei 1234.MAC wird assembliert. Der Object-Code wird im Intel-Hex-Format in die Datei 1234.HEX geschrieben (falls nicht im Quelltext anders angegeben).

```
/Q                                              (CP/M)
```

Der Assembler wird verlassen. Die Fehlerdatei wird, falls geöffnet, geschlossen.

```
errors-e;pgm1;pgm2;-n;pgm3                      (TRS-80)
```

Die Dateien PGM1/MAC und PGM2/MAC werden assembliert. Der Dateityp der Object-Dateien hängt von den in den Programmen selektierten Optionen ab (falls keine derartigen Optionen angegeben sind, erzeugt der Assembler REL-Dateien). Die auftretenden Fehler werden in die Datei ERRORS/ERR geschrieben. Danach wird PGM3/MAC assembliert. Die hier auftretenden Fehler werden nicht in ERRORS/ERR vermerkt.

```
=DUMP                                           (CP/M)
```

Die Datei DUMP.MAC wird assembliert. Es wird kein Object-Datei und keine Listing-Datei erzeugt. Dieses Format dient dem schnellen Überprüfen von Programmen auf das Vorhandensein von Fehlern, ohne daß eine Listing- oder Object-Datei erzeugt wird.

```
,TTY:=TTY:                                      (CP/M)
,*DO=*KI                                        (TRS-80)
```

Die Eingabe erfolgt von der Tastatur. Das Listing wird auf dem Bildschirm ausgegeben. Es wird keine Object-Datei erzeugt. Um das Listing zu sehen, ist zuerst ein END einzugeben, damit der Assembler in den zweiten Durchlauf kommt.

2.3 Die Switches

2.3.1 Liste der Switches

/A Beim Auftreten eines Fehlers wird der Assembler verlassen.
/B Die Listing-Zahlenbasis ist 2.
/C Es wird eine COM-Datei erzeugt.
/D Die Listing-Zahlenbasis ist 10.
/E Fehlerdatei Öffnen (s.o.).
/F SET hat die Bedeutung von DEFL.
/G Symbol table nicht sortieren.
/H Die Listing-Zahlenbasis ist 16 (Standard).
/I Es wird überprüft, ob das Programm 8080-kompatibel ist.
/Jx Die Symbol table wird auf das Laufwerk x ausgelagert.
/K -- Nicht verwendet --
/L Listing mit Zeilennummern versehen.
/M Es wird eine CMD- und RLD-Datei erzeugt (TRS-80).
/N Fehlerdatei schließen (s.o.).
/O Die Listing-Zahlenbasis ist 8.
/P Jeder P-Switch bewirkt die Reservierung von 256 bytes mehr für den Stack (für tief geschachtelte Macros oder tief geschachtelte Compiler-Statements).
/Q Abbruch.
/R REL-Datei erzeugen (Standard).
/S Symbol table auf Bildschirm ausgeben.
/T CMD-Datei erzeugen (TRS-80).
/U Liste der nicht definierten Labels auf Bildschirm ausgeben.
/V Bei jeder assemblierten Zeile ein Zeichen ausgeben.
/W Beim Auftreten eines Fehlers anhalten und auf <RETURN> warten.
/X HEX-Datei erzeugen (Intel-Hex-Format).
/Y SYM-Datei erzeugen.
/Z Nicht überprüfen, ob das Programm 8080-kompatibel ist (Standard).
/6 Labels werden auf 6 Zeichen gekürzt.

2.3.2 Die Switches C, X, M und T

Die Switches /C und /X sind nur in der CP/M-Version sinnvoll, /M und /T sind nur in der TRS-80-Version sinnvoll, da die entsprechenden Dateitypen nicht unter diesen Betriebssystemen verarbeitet werden können.

2.3.3 Sich ausschließende Switches

Die Switches /C, /M, /R, /X und /T schließen einander jeweils aus (es kann nur eine Object-Datei erzeugt werden). Falls in der Quelldatei angegeben ist, welche Object-Datei erzeugt werden soll (z.B. .HEX) werden diese Switches ignoriert (dies wird wahrscheinlich in einer späteren Version geändert).

Bei den sich ausschließenden Switches gilt immer der zuletzt eingegebene, wobei keine Fehlermeldung ausgegeben wird.

2.3.4 Die Switches B, D, O und H: Listing-Zahlenbasis

Die Switches /B, /D, /O und /H schließen einander jeweils aus (es kann natürlich nur eine Listing-Zahlenbasis angegeben werden). Die Angabe in der Quelldatei hat Vorrang (.LRADIX).

2.3.5 Die Switches I und Z: Einstellung des Prozessortyps

Die Switches /I und /Z schließen sich gegenseitig aus. Auch diese Switches werden durch die entsprechenden Direktiven außer Kraft gesetzt, jedoch bleiben sie auch bei den nächsten Kommandos aktiv, bis das Gegenteil angegeben wird, genauer, durch /I und /Z wird die Vorwahl getroffen, die für die Dauer der Assemblierung einer Datei durch .8080 und .Z80 außer Kraft gesetzt werden kann.

2.3.6 Der W-Switch: Beim Auftreten eines Fehlers anhalten

Wenn /W angegeben wird, hält der Assembler bei jeder fehlerhaften Zeile an und wartet auf einen Tastendruck. Bei Eingabe von CTRL-C bei CP/M oder dem Pfeil nach oben beim TRS-80 wird der Assembliervorgang abgebrochen, bei jeder anderen Taste wird fortgefahren. Die Wirkung ist die gleiche wie beim Antreffen von *PAUSE (siehe dort).

2.3.7 Der V-Switch: Bei jeder Zeile wird ein Zeichen ausgegeben

/V bewirkt die Ausgabe eines Zeichens für jede bearbeitete Zeile sowie für jedes der angeführten speziellen Statements. /V wird ignoriert, wenn die Ausgabe ohnehin auf den Bildschirm geht (Listing-Datei = TTY: oder *DO) oder die Eingabe von der Tastatur erfolgt. Es werden folgende Zeichen ausgegeben:

.	Normale Zeile
+	Macro-Expansion
!	LINK-Statement
<	INCLUDE/MACLIB/SYMLIB/COMLIB-Statement
>	Ende der INCLUDE/MACLIB/SYMLIB/COMLIB-Datei
:	Vier Bytes gelesen (COMLIB)
$	Label gelesen (SYMLIB)
!	LINK-Statement
R	*RELTAB HERE
D	*DATA HERE

Am Ende eines Durchlaufs wird ein Zeilenvorschub ausgegeben.

2.3.8 Der Y-Switch: Schreiben einer SYM-Datei

Wird der Y-Switch gesetzt, wird eine Label-Datei erzeugt. Darin werden sämtliche absoluten Labels eingetragen. Es muß eine Object-Datei angegeben werden. Als Name für die Label-Datei wird der Name der Object-Datei mit SYM-Dateityp verwendet. Beispiel:

```
Z80 TEST/Y          oder        Z80 ABC=DEF/Y
```

Hiermit wird TEST.SYM bzw. ABC.SYM erzeugt.

2.3.9 Aufbau der SYM-Dateien

Für jedes Label ist eine eigene Zeile vorhanden, in der zuerst der Wert steht, der dem Label zuletzt zugewiesen wurde (DEFL!), gefolgt von einem Leerzeichen und dem Label selbst. Der Wert wird immer hexadezimal angegeben. Die Zeile wird durch CR/LF (CP/M) bzw. CR (TRS-80) abgeschlossen.

2.3.10 Der A-Switch: Abbruch bei Fehler

Wenn ein Fehler auftritt und der A-Switch gesetzt ist, wird der Assembler nach Ausgabe der Fehlermeldung verlassen. Unter NEWDOS/80 wird das Chaining abgebrochen, die CP/M-Version löscht die Datei $$$.SUB auf Laufwerk A, um die Befehlsverkettung durch SUBMIT abzubrechen.

2.3.11 Der J-Switch: Auslagern der Symbol table

Durch Angabe des Jx-Switches kann die Symbol table auf Diskette ausgelagert werden. Dies ist erforderlich, falls der Speicherbedarf für die Labels zu groß wird. Auf diese Weise können etwa doppelt so viele Labels benutzt werden (unabhängig von der Länge derselben), jedoch muß mit einer drastischen Senkung der Arbeitsgeschwindigkeit gerechnet werden. Deshalb ist es empfehlenswert, eine sogenannte RAM-Disk zu verwenden. Für x ist das Laufwerk anzugeben, auf dem die Datei für die ausgelagerten Labels anzulegen ist. Hier ist möglichst nicht das Laufwerk zu benutzen, auf dem die Quelldatei steht! In der TRS-80-Version ist für x 0 bis 3, in der CP/M-Version A bis P erlaubt. In der TRS-80-Version kann x weggelassen werden. Dann wird das beim vorigen Jx-Switch angegebene Laufwerk, oder falls noch kein J-Switch angegeben wurde, das Laufwerk :1, benutzt. Außerdem empfiehlt es sich, falls nicht unbedingt eine sortierte Symbol table im Listing erforderlich ist, den G-Switch anzugeben, da die Sortierung erhebliche Zeit in Anspruch nehmen kann. Dies ist jedoch nicht erforderlich, falls die Symbol table nicht ausgegeben wird, daher kann, wenn überhaupt keine Symbol table gewünscht ist, diese durch Anbringung eines .XLIST am Ende der Quelldatei unterdrückt werden.

3 Der Quelltext

3.1 Labels

Labels bestehen aus beliebig (im Rahmen der maximal zulässigen Zeilenlänge) vielen Zeichen: Das erste muß ein Buchstabe oder ein erlaubtes Sonderzeichen sein, die restlichen (falls vorhanden) können außerdem Ziffern und '%' sein. Erlaubte Sonderzeichen sind:

$ # @ . _ ?

Bei '#' ist eine Besonderheit zu beachten: Wenn zwei '#' aufeinanderfolgen, wird das Label als 'extern' betrachtet. Nach diesen zwei '#' dürfen keine weiteren Zeichen, die zum Label gehören, folgen. Vorsicht: Manche Linker haben Probleme mit diesen Sonderzeichen in Labels!
Groß- und Kleinschreibung wird in Labels nicht unterschieden. Falls keine andere Option gewählt wurde, sind die ersten 32 Zeichen der Labels signifikant, d.h. es werden die ersten 32 Zeichen verglichen. Wenn sich zwei Labels erst im 33. Zeichen unterscheiden, sind sie für den Assembler identisch. Wurde der 6-Switch gesetzt, so werden nur die ersten 6 Zeichen verglichen, bei Angabe von .SYMLEN n werden die ersten n Zeichen verglichen.
Wird eine REL-Datei erzeugt, so werden die ersten 7 Zeichen von externen und globalen Labels sowie Requests in die Object-Datei geschrieben. Durch Angabe des 6-Switches, .SYMLEN 6 (alle Labels 6 Zeichen lang) oder .EXTLEN 6 (nur Labels für Linker 6 Zeichen lang) läßt sich eine Label-Länge von 6 Zeichen (oder weniger, je nach Parameter bei .SYMLEN und .EXTLEN) erzwingen.
Labels dürfen keine Registernamen (HL,DE,R, ...) sein. Mnemonics (NOP, LDIR, ...) sowie Direktiven (DEFW, .RADIX, ...) und Operatoren (NOT, AND, XOR, GT, ...) sind jedoch erlaubt!

3.1.1 Beispiele für gültige Labels

```
ANFANG          ende            #Of_Bytes       $$$$
NaCl            IBM.3740        fertig?         line#
Test%12         HIGH            and             x%y
Eingabe_einer_Zeile             print@          CPIR
Die.Laenge.der.Marken.ist.nahezu.unbegrenzt
OUTPUT##  (OUTPUT ist external)
```

3.1.2 Beispiele für ungültige Labels

```
4711            Störung         HL              error###
%test           x##y            A               2nd
$         (Variable für PC, s.u.)
```

3.2 Das Eingabeformat

3.2.1 Zeilenaufbau

Die Eingabe erfolgt formatfrei mit einer maximalen Zeilenlänge von 128 Zeichen. Eine Zeile enthält keine, eine oder mehrere Label-Definitionen, keine, eine oder mehrere Z80-Instruktionen oder Direktiven und/oder einen optionalen Kommentar. Zeilen, die mit einem '*' anfangen (davor beliebig viele Leerzeichen), werden als Kommentarzeilen interpretiert. Ausnahme: Das '*'-Zeichen gehört zu einer der Direktiven, die mit '*' beginnen (s.u.).

3.2.2 Label-Definitionen

Eine Label-Definition besteht aus einem gültigen Label, gefolgt von einem Doppelpunkt oder von zwei Doppelpunkten. In letzterem Falle wird das Label gleich global definiert. Dem Label wird der aktuelle PC-Wert zugeordnet. Der Typ des Segments, in dem der PC steht, wird ebenfalls dem Label zugewiesen (ASEG, CSEG, DSEG oder COMMON).
Soll nach der Label-Definition eine Direktive stehen, die mit einem '*' anfängt, so ist ein ! einzufügen (damit das '*' am Anfang einer 'Zeile' steht).

3.2.2.1 Beispiele für gültige Label-Definitionen

```
marke:
ENTRY:A0100:START:
global_label::
```

3.2.2.2 Beispiele für ungültige Label-Definitionen

```
TEST:::                 ; Ein Doppelpunkt zuviel
1984:                   ; Fängt mit Ziffer an
hallo:hallo:            ; Doppelt definiert
hier                    ; Doppelpunkt fehlt
```

3.2.3 Statements

Nach der Label-Definition (falls vorhanden) folgt ein Statement oder eine Direktive oder ein Kommentar oder, im einfachsten Fall, nichts. Somit sind auch 'leere' Zeilen, und Zeilen, in denen nur ein Label definiert wird, sowie reine Kommentarzeilen erlaubt.

Ein Statement besteht aus einem Mnemonic und optionalen Parametern (hängt vom Mnemonic ab). Es können mehrere Statements und Label-Definitionen in einer Zeile stehen, die dann durch '!' getrennt werden.

3.2.4 Kommentare

Ein Kommentar beginnt entweder mit einem ';' oder mit '--'. Am Zeilenanfang kann der Kommentar auch mit '*' anfangen. Alle Zeichen nach dieser Markierung bis zum Zeilenende werden ignoriert.

3.2.5 Beispiele für gültige Quellzeilen

```
LOOP:   JR      LOOP            ; Endlosschleife

mul8:SLA A!mul4:SLA A!mul2:SLA A ; Multiplikation mit 8, 4 oder 2
        SLS     A               --> A := 2 * A + 1

divide_HL_by_2:SRL H! RR L      --  HL := HL / 2

-------------------------------- Kommentar

end_of_code:

; Kommentar

************************
*   Auch ein Kommentar *
************************

test    :       or a            ; Ist Accu = 0 ?
```

3.2.6 Beispiele für ungültige Quellzeilen

```
          HALLO LD  HL,1234H

          test: or  a           * Ist Accu = 0 ?

                ADD A,A   ADD A,A

                INC A           - - A := A + 1
```

4 Ausdrücke

4.1 Übersicht über die Operatoren

4.1.1 Operatoren mit zwei Operanden

*	Multiplikation (ohne Vorzeichen, 16 bit)
/	Division (ohne Vorzeichen, 16 bit, ganzzahlig)
+	Addition
-	Subtraktion
MOD	Rest bei Division
AND	Logische Und-Verknüpfung (16 bit)
OR	Logische Oder-Verknüpfung (16 bit)
XOR	Logische Exklusiv-Oder-Verknüpfung (16 bit)
SHL	Nach links schieben
SHR	Nach rechts schieben
EQ	Vergleich: Gleich ?
=	Vergleich: Gleich ?
NE	Vergleich: Ungleich ?
<>	Vergleich: Ungleich ?
LT	Vergleich: Kleiner ?
<	Vergleich: Kleiner ?
LE	Vergleich: Kleiner oder gleich ?
<=	Vergleich: Kleiner oder gleich ?
GT	Vergleich: Größer ?
>	Vergleich: Größer ?
GE	Vergleich: Größer oder gleich ?
>=	Vergleich: Größer oder gleich ?
SLT	Vergleich: Kleiner (mit Vorzeichen) ?
SLE	Vergleich: Kleiner oder gleich (mit Vorzeichen) ?
SGT	Vergleich: Größer (mit Vorzeichen) ?
SGE	Vergleich: Größer oder gleich (mit Vorzeichen) ?

Tabelle 4-1: Operatoren mit zwei Operanden

4.1.2 Operatoren mit einem Operanden

-		Vorzeichenwechsel
+		Kein Vorzeichenwechsel
LOW	%LOW	Bits 0-7 des Operanden
HIGH	%HIGH	Bits 8-15 des Operanden -> Bits 0-7
NOT	%NOT	Logische Nicht-Verknüpfung
NUL	%NUL	Folgt kein Argument ?
TYPE	%TYPE	Typ des Operanden

Tabelle 4-2: Operatoren mit einem Operanden

4.2 Konstanten

FALSE	%FALSE	0000H
TRUE	%TRUE	FFFFH

Tabelle 4-3: Konstanten

4.3 Reihenfolge der Abarbeitung von Operatoren

Die Operatoren werden in der Reihenfolge von oben nach unter in dieser Tabelle abgearbeitet.

```
TYPE NUL  *Byte-Liste
( )  <opcode>
LOW  HIGH
*    /    MOD  SHR  SHL
-    (Vorzeichen)
+    -
EQ   NE   LT   LE   GE   GT   SLT  SLE  SGT  SGE  <= <> < = > >=
NOT
AND
OR   XOR
```

Tabelle 4-4: Reihenfolge der Operatoren

Gleichrangige Operationen werden von links nach rechts berechnet. Enthält ein Unterausdruck einen Operator mit höherem Rang, so wird zuerst der Unterausdruck berechnet. Die Berechnungsreihenfolge kann mit runden Klammern geändert werden. Ausdrücke in Klammern werden zuerst berechnet. Vorsicht:

```
LD HL,(5)+3
```

ist erlaubt und bedeutet das gleiche wie

```
LD HL,(8)
```

4.4 Die Modi

4.4.1 Absolut

Die absolute Adresse steht schon beim Assemblieren fest. Absolut sind alle Zahlen, sofern sie nicht von ' oder " gefolgt werden, alle Labels, denen mit EQU oder DEFL ein absoluter Wert zugewiesen wurde, die mi ':' definiert wurden, nachdem ASEG angegeben wurde und alle Labels, wenn eine COM-, HEX- oder CMD-Datei erzeugt wird. Dieser Modus wird für Konstanten benutzt und wenn der Programmierer den Code an einer bestimmten Stelle im Speicher haben möchte. Dieser Modus wird mit ASEG gewählt.

4.4.2 Relativ zum Datensegment

Die Adresse ist relativ zum Datensegment (data relative) und relozierbar. Beim Linken kann die Basisadresse des Datensegments mit dem D-Switch des Linkers gesetzt werden. Dieser Modus wird üblicherweise für Daten benutzt, die geändert werden und somit im RAM stehen müssen. Dieser Modus wird mit DSEG gewählt.

4.4.3 Relativ zum Programmsegment

Die Adresse ist relativ zum Programmsegment (code relative) und relozierbar. Beim Linken kann die Basisadresse des Programmsegments mit dem P-Switch des Linkers gesetzt werden. Dieser Modus wird üblicherweise für das Programm benutzt und wird mit CSEG gewählt.

4.4.4 COMMON

Die Adresse ist relativ zu einem COMMON-Bereich und relozierbar. Dieser Modus wird gebraucht, um Daten mit einem FORTRAN-Programm auszutauschen. Dieser Modus wird mit COMMON gewählt.

4.4.5 Allgemeines

Alle Adressen mit einem nicht absoluten Modus können vom Linker reloziert (verschoben) werden. Dies ist nötig, wenn ein Programm aus verschiedenen Moduln zusammengefügt wird. Die absoluten Adressen werden dann vom Linker erzeugt.

Zusätzlich gibt es noch zwei Modi zur Übergabe von Adressen zwischen Programmoduln:

4.4.6 Externe Labels

Der Wert eines Labels mit Modus extern ist zur Zeit der Assemblierung noch nicht bekannt. Das Label wird vielmehr in einem anderen Modul definiert (es ist dort als globales Label definiert) und der Wert wird vom Linker eingesetzt.

4.4.7 Globale Labels

Eine Adresse wird mit einem Namen assoziiert, so daß sie in anderen Moduln verwendet werden kann. Wenn das Modul, in dem ein solches Label definiert wurde, vom Linker geladen wird, ersetzt er sämtliche Zugriffe von anderen Moduln auf dieses Label durch die nun bekannte absolute Adresse.

4.5 Operatoren und Konstanten

4.5.1 Konstanten

FALSE Hat den Wert 0000H

TRUE Hat den Wert FFFFH = -1

Existiert ein Label mit dem Namen FALSE oder TRUE, so ist für die Konstante %FALSE bzw. %TRUE zu verwenden, damit es nicht zu Konflikten kommt,da das Label Vorrang hat.

4.5.2 Operatoren mit einem Operanden

+ Keine Vorzeichenänderung (d.h. überhaupt keine Wirkung).

- Ändert das Vorzeichen des Operanden. Der Operand muß absolut sein.

HIGH Isoliert das höherwertige Byte des Operanden, der absolut sein muß. HIGH 1234H ergibt 12H.

LOW Isoliert das niederwertige Byte des Operanden, der absolut sein muß. LOW 1234H ergibt 34H.

NUL Ergibt TRUE, falls kein Operand, d.h. ein Kommentar, ein Ausrufezeichen oder das Ende der Zeile folgt. Sonst ergibt NUL den Wert FALSE.

TYPE TYPE liefert ein Byte, das den Typ und Modus des Arguments kennzeichnet. Bits 0 und 1 ergeben den Modus:

0: absolut
1: relozierbar, relativ zum Programmsegment
2: relozierbar, relativ zum Datensegment
3: relozierbar, COMMON

Falls das Argument extern ist, wird Bit 7 des Bytes gesetzt. Wenn das Argument nicht definiert oder extern ist, wird Bit 5 des Bytes gesetzt.

NOT Logisches NICHT. Invertiert alle Bits des absoluten Arguments.

Existiert ein Label mit dem Namen HIGH, LOW, NUL, TYPE oder NOT, so ist für den Operator %HIGH, %LOW, %NUL, %TYPE bzw. %NOT zu verwenden, da sonst das gleichnamige Label eingesetzt wird.

4.5.3 Operatoren mit zwei Operanden

* Ergibt das Produkt der beiden Operanden. Beide Operanden müsse absolut sein. Das Vorzeichen wird beachtet.

/ Ergibt den Quotienten der beiden Operanden. Beide Operanden müssen absolut sein. Der zweite Operand darf nicht 0 sein. Das Vorzeichen der Operanden wird beachtet.

+ Addiert die beiden Operanden. Zu einem absoluten Operanden kann ein beliebiger anderer addiert werden. Zu einem externen Operanden darf kein anderer externer Operand addiert werden. Werden zwei relozierbare Operanden addiert, so müssen sie den gleichen Modus haben und es muß im gleichen Ausdruck ein relozierbarer Operand mit gleichem Modus abgezogen werden, so daß das Ergebnis entweder einfach relozierbar oder aber absolut ist.

- Subtrahiert den zweiten Operanden vom ersten. Der zweite Operand darf kein externer sein. Wenn zwei relozierbare Operanden subtrahiert werden, müssen sie den gleichen Modus haben. Das Ergebnis ist dann absolut.

MOD MOD ergibt den Rest bei der ganzzahligen Division der beiden Operanden. Weiteres siehe bei '/'. 10 MOD 3 ergibt 1.

SHR Shift Right. SHR wird von einem Operanden gefolgt, der angibt, um wieviele Bit-Positionen der erste Operand nach rechts geschoben wird. 17 SHR 2 ergibt 4.

SHL Shift Left. SHL wird von einem Operanden gefolgt, der angibt, um wieviele Bit-Positionen der erste Operand nach links geschoben wird. 4 SHL 2 ergibt 16.

EQ = Equal. Wenn beide Operanden gleich sind ergibt EQ bzw. = den Wert TRUE, ansonsten FALSE.

NE <> Not Equal. Wenn beide Operanden gleich sind ergibt NE bzw. <> den Wert FALSE, sonst den Wert TRUE.

LT < Less Than. Ergibt TRUE, wenn der erste Operand kleiner als der zweite ist (ohne Vorzeichen).

LE <= Less than or Equal. Ergibt TRUE wenn der erste Operand nicht größer als der zweite ist (ohne Vorzeichen).

GT > Greater Than. Ergibt TRUE wenn der erste Operand größer als der zweite ist (ohne Vorzeichen).

GE >= Greater than or Equal. Ergibt TRUE wenn der erste Operand nicht kleiner als der zweite ist (ohne Vorzeichen).

SLT Signed Less Than. Ergibt TRUE wenn der erste Operand kleiner als der zweite ist (mit Vorzeichen).

SLE Signed Less than or Equal. Ergibt TRUE wenn der erste Operand nicht größer als der zweite Operand ist (mit Vorzeichen).

SGT Signed Greater Than. Ergibt TRUE wenn der erste Operand größer als der zweite ist (mit Vorzeichen).

SGE Signed Greater than or Equal. Ergibt TRUE, wenn der erste Operand nicht kleiner als der zweite ist (mit Vorzeichen).

Bei den Vergleichsoperationen müssen beide Operanden den gleichen Modus haben. Sie dürfen nicht extern sein. Das Ergebnis ist absolut.

AND Logisches UND. Ergibt TRUE, wenn beide Operanden TRUE sind. Ein Bit im Ergebnis wird nur dann auf 1 gesetzt, wenn die entsprechenden Bits der Operanden auf 1 gesetzt sind (bei beiden zugleich). Beide Operanden müssen absolut sein.

OR Logisches ODER. Ergibt TRUE, wenn mindestens einer der beiden Operanden TRUE ist. Ein Bit im Ergebnis wird dann auf 1 gesetzt, wenn eines der entsprechenden Bits der Operanden auf 1 gesetzt ist. Beide Operanden müssen absolut sein.

XOR Logisches EXKLUSIV-ODER. Ergibt FALSE, wenn beide Operanden TRUE oder beide Operanden FALSE sind. Ein Bit im Ergebnis wird auf 1 gesetzt, wenn die entsprechenden Bits der Operanden verschieden gesetzt sind. Beide Operanden müssen absolut sein.

4.6 Ergebnismodus bei gemischten Ausdrücken

Relativ zum Programmsegment wird mit 'code' abgekürzt, relativ zum Datensegment mit 'data'.

Operation	Ergebnis
absolut + - absolut	-> absolut
absolut + code	-> code
absolut + data	-> data
absolut + extern	-> extern
absolut + COMMON	-> COMMON
code + - absolut	-> code
code - code	-> absolut
code + extern	-> extern
data + - absolut	-> data
data + extern	-> extern
data - data	-> absolut
extern + - absolut	-> extern
extern + code	-> extern
extern + data	-> extern
COMMON + - absolut	-> COMMON
COMMON - COMMON	-> absolut (nur bei gleichem COMMON-Block)

Tabelle 4-5: Ergebnismodus bei gemischten Ausdrücken

Die Addition von externen Labels und relativen Werten ist nur sinnvoll, wenn der externe Operand ein absoluter ist.
Während des Ausrechnens des Ausdruckes darf auch code+code und data+data gerechnet werden, falls später wieder code bzw. data subtrahiert wird (z.B. code+code+code-code-code -> code).

4.7 Die Operanden

4.7.1 Zahlen

Eine Zahl kann einen Wert zwischen 0 und 65535 (jeweils dezimal und einschließlich) haben. Zahlen können auch mit einem Vorzeichen behaftet sein. Dann sind Werte zwischen -32768 und 32767 (jeweils dezimal und einschließlich) erlaubt. Zahlen können in der aktuellen Zahlenbasis angegeben werden (siehe .RADIX) oder mit einem Buchstaben am Ende als Binär-, Oktal-, Dezimal- oder Hexadezimalzahl gekennzeichnet werden. Folgt auf eine Folge von Ziffern aus dem Bereich 0 bis 1 ein 'B', so wird diese Zahl als Binärzahl (Dualzahl) interpretiert. Ein 'O' oder 'Q' nach einer Folge von Ziffern aus dem Bereich von 0 bis 7 kennzeichnet eine Oktalzahl (Zahlenbasis ist 8), ein 'D' nach einer Folge von Ziffern aus dem Bereich von 0 bis 9 kennzeichnet eine Dezimalzahl, ein 'H' nach einer Folge von Ziffern aus dem Bereich von 0 bis 9 und A bis F markiert eine Hexadezimalzahl (Zahlenbasis ist 16). Die 'Ziffern' A bis F stehen für die Werte 10 bis 15. Eine Zahl (insbesondere eine Hexadezimalzahl) muß mit einer Ziffer anfangen. Vor eine Zahl, die mit einem Buchstaben beginnt, ist also eine 0 zu schreiben. Wenn dies nicht geschieht, wird die 'Zahl' als Label interpretiert. Falls der angehängte Buchstabe (B, D, O, Q oder H) im Bereich der Ziffern der aktuellen Zahlenbasis vorkommt, wird die aktuelle Zahlenbasis verwendet. Zwischen den Ziffern können zur Erhöhung der Übersichtlichkeit '$' oder '_' eingefügt werden. Falls die Zahlenbasis größer als 10 ist, werden die restlichen Ziffern durch die Buchstaben ab 'A' repräsentiert. Eine Zahl hat normalerweise den Modus 'absolut'. Falls aber der relative oder relozierbare erwünscht ist, so kann der Zahl ein ' für Programmsegment- oder ein " für Datensegment-relativ nachgestellt werden. X'nnnn' ist eine andere Möglichkeit zur Darstellung von Hexadezimalzahlen.
An eine Dezimalzahl kann ein 'K' angehängt werden, welches die Zahl mit 1024 (1 K) multipliziert. Zulässige Werte sind 0 bis 63, wobei die Zahl immer dezimal interpretiert wird.

Zahlenbasen	Kennzeichnung
2	B
8	O, Q
10	D
16	H
10, x 1K	K

Tabelle 4-6: Kennzeichnung von Zahlenbasen

Beispiele:	(dezimal:)
12_34	1234
X'A'	10
101B	5
777Q"	511 (relativ zum Datensegment)
32K	32768

4.7.2 Control-Codes

Die Werte 0 bis 26 können auch als Control-Codes angegeben werden. Hierzu ist ein Pfeil nach oben (ASCII 5EH) gefolgt von @ oder A..Z anzugeben. Beispiel: LD A,^H

4.7.3 Labels

Labels können auch für Operanden verwendet werden. Der Modus des Operanden ist dann der Modus des Labels.

4.7.4 Aktuelle Adresse: $

Das Dollarzeichen ist eine Variable, die den aktuellen Wert des PC enthält. Sie hat den Modus, den das Segment hat, in dem sich der PC augenblicklich befindet. Wenn der PC in einem PHASE-Block steht, ist $ absolut. $ darf nicht als Label verwendet werden!

4.7.5 Texte als Operanden

Ein Text wird zwischen ' oder zwischen " eingeschlossen. Der Anfang und das Ende müssen aber mit demselben Zeichen markiert werden. Soll ein solches Anführungszeichen in einem Text erscheinen und ist es identisch mit denen am Anfang und am Ende des Textes, so muß es doppelt geschrieben werden. In Ausdrücken sind nur Texte erlaubt, die aus einem oder aus zwei Zeichen bestehen. Bei DEFM/DM, DEFC/DC und DEFH/DH sind jedoch auch längere Texte erlaubt. Falls keine Operatoren auf den Text angewandt werden, sind lange Texte auch bei DEFB/DB zugelassen. Da DEFB/DB weggelassen werden kann, braucht nur der Text als solcher angegeben werden (siehe Beispiel). Kleinbuchstaben, die zwischen Anführungszeichen stehen, werden nicht in Großbuchstaben umgewandelt. Ein Text kann nicht durch das Zeilenende beendet werden; es muß ein Anführungszeichen gesetzt werden. Innerhalb eines Textes dürfen auch Control-Codes (ASCII 01H..1FH) stehen, wobei allerdings CTRL-M, CTRL-J und CTRL-L nicht vorkommen sollten, da sie anders interpretiert werden (Zeilenende bzw. Seitenende). CTRL-Z ist nicht erlaubt, da es das Ende der Datei kennzeichnet. Vorsicht bei CTRL-I (tab): Da vor einem Text zumindest ein Anführungszeichen steht, verschiebt sich der Text um mindestens ein Zeichen. Dies kann bewirken, daß der nachher ausgegebene Text anders als im Text-Editor eingegeben aussieht, da eventuell um eine Tabulator-Position weiter gesprungen wird!
Zwei direkt aufeinanderfolgende Anführungszeichen werden als 0 interpretiert (nicht jedoch bei DEFB und DEFM: Hier wird ein solcher 'leerer' Text ignoriert).

Beispiele:

```
'Let''s go'

DEFM    "Let's go"
```

Die beiden Beispiele liefern identischen Code!

4.7.6 *Byte-Liste

(1) Bei Erzeugung einer REL-Datei:
Die Byte-Liste, die auf das '*'-Zeichen folgt, wird in das Datensegment geschrieben. Für die Byte-Liste gelten die gleichen Regeln wie bei DEFB (s.u.). Es ist allerdings kein <opcode>-Operand erlaubt. Als Ergebnis liefert '*' die Adresse der Byte-Liste (Typ: Relativ zum Datensegment). Das PC-Segment muß CSEG oder ASEG sein.

(2) Bei Erzeugung einer HEX-, COM- oder CMD-Datei:
Die Byte-Liste, die auf das '*'-Zeichen folgt, wird vom Assembler in internen Tabellen gespeichert. Für die Byte-Liste gelten die gleichen Regeln wie bei DEFB (s.u). Es ist allerdings kein <opcode>-Operand erlaubt. Als Ergebnis liefert '*' die Adresse der Byte-Liste. Trifft der Assembler das Statement *DATA HERE an, so werden alle bis dahin gespeicherten Listen an dieser Stelle eingefügt. *DATA HERE kann mehrfach verwendet werden, um z.B. bei sich verschiebenden Programmen nur einen Teil der Listen mitzuverschieben.

(3) Mit *DATA HERE bei Erzeugung einer REL-Datei:
Falls man auch bei REL-Dateien mit *DATA HERE arbeiten möchte, muß man vor der ersten Benutzung von *Byte-Liste ein *DATA HERE anbringen, damit der Assembler weiß, daß diese Daten nicht in das Datensegment geschrieben werden sollen. Nach dieser Angabe kann wie unter (2) beschrieben gearbeitet werden.

Dieses Feature wird vor allem für die Ausgabe von Texten verwendet, ist aber auch nützlich, falls man einen Wert aus einer kleinen Tabelle holen möchte, ohne sich extra hierfür ein Label ausdenken zu müssen.

Beispiel (REL-Datei wird erzeugt):

```
        LD   DE,*0DH,0AH,"Eingabe: $"
        LD   C,9
        CALL 5
```

Dies entspricht:

```
        DSEG
   L1:  0DH,0AH,"Eingabe: $"
        CSEG
        LD   DE,L1
        LD   C,9
        CALL 5
```

(sonstige Datei wird erzeugt):

```
        LD   HL,*1,2,4,8,16,32,64,128
        ADD  HL,DE
        RET
        *data here
```

Dies entspricht:

```
        LD   HL,TAB
        ADD  HL,DE
        RET
   TAB: 1,2,4,8,16,32,64,128


        DEFW    1,2,(*3,4,5),6,7
```

Dies entspricht:

```
        DEFW    1,2,$$DEFW,6,7
$$DEFW: 3,4,5
```

4.7.7 <opcode>

Als Operand ist auch eine Z80-Instruktion erlaubt. Sie muß dann zwischen spitze Klammern geschrieben werden. Bei Instruktionen mit der Länge eins (z.B. HALT; LD (HL),E; RET) erhält man als Wert den Opcode der Instruktion, wobei Bits 8 bis 15 mit 0 aufgefüllt werden. Bei Zwei-Byte-Instruktionen kommt das erste Byte des Opcodes in das niederwertige Byte des Ergebnisses, das zweite Byte des Opcodes kommt in das höherwertige Byte des Ergebnisses. Bei Drei- und Vier-Byte-Befehlen werden die ersten beiden Bytes wie bei Zwei-Byte-Instruktionen behandelt, die restlichen werden ignoriert. Wird ein 8-bit-Ergebnis gewünscht, so muß dafür gesorgt werden, daß Bits 8 bis 15 auf 0 gesetzt sind. Dies kann durch die Angabe einer 0 als Operand oder durch LOW bzw. HIGH (letzteres bei ED/CB/DD/FD-Befehlen) erreicht werden.

Beispiele:

```
<NOP>                   0000H
<LD A,"*">              2A3EH
<LDIR>                  B0EDH
<LD IX,1234H>           21DDH
<LD DE,1234H>           3411H
<LD (HL),<HALT>>        7636H
```

Eine tiefere Schachtelung als wie beim letzten Beispiel ist nicht erlaubt. Das Ergebnis hat immer den Modus absolut. Außer Z80-Instruktionen sind auch verkürzte DEFB-Statements (d.h. ohne 'DEFB') und DEFW-Statements erlaubt. Beispiele:

```
<DW 1234H>              1234H
<1,2>                   0201H
<5>                     0005H
```

Da <opcode> auch als Argument für DEFB zugelassen ist, können damit zwei Opcodes angegeben werden. Beispiele:

```
<<RLCA>,<INC A>>        3C07H
<<LD A,0>,<XOR A>>      AF3EH
```

Ein <opcode>-Argument kann auch als Displacement bei der IX- und IY-indizierten Adressierung benutzt werden:

```
LD   (IX+<INC A>),<XOR A>
CP   (IY+<EXX>)
```

Nicht erlaubt ist ein solches Argument in folgenden Fällen:

```
LD ( ),BC
LD ( ),DE
LD ( ),IX
LD ( ),IY
LD ( ),SP
LD SP,( )
LD BC,( )
LD DE,( )
```

Innerhalb der spitzen Klammern kann auch ein Label definiert werden, allerdings nicht, wenn <<opcode>,<opcode>> angewandt wird. Das Label bekommt die Adresse zugewiesen, an die das Argument geschrieben wird.

Beispiele:

```
LD   HL,<ADDR:DW 0>
LD   A,(IX+<displacement:0>)
CP   <cmd::EXX>
LD   (HL),<byte:255>
```

Dies findet vor allem beim Schreiben von sich selbst verändernden Programmen Anwendung.

Anmerkung:

```
LD   A,(IX-<dis:0>)
```

ist nicht sinnvoll, da es vom Displacement abhängt, ob addiert oder subtrahiert wird. Das Minuszeichen wird nur beim Einsetzen des Anfangswertes durch den Assembler beachtet.

Bei einigen Instruktionen kann das Argument entfallen, falls sie innerhalb von <> angewandt werden. Dies sind CALL, JP, JR, DJNZ sowie CALL, JP und JR mit Bedingung. Beispiele:

```
LD   A,<CALL>
LD   (IX+5),<DJNZ>
LD   B,<JP NZ>
```

Hier folgen nun noch einige allgemeine Beispiele zur Anwendung von solchen Argumenten:

```
        LD   HL,<JR $>        -->  21H 18H FEH
        LD   (forever),HL

ON:     LD   A,<OFF:XOR A>

ON:     <LD  A,0>
OFF:    XOR  A      --> entspricht dem vorigen Beispiel!

push_nothing:  <CP  0>
      push_hl:      PUSH HL
```

Weitere Beispiele und Einschränkungen folgen, wenn zusätzliche Features des Assemblers beschrieben werden.

5 Nicht dokumentierte Z80-Instruktionen

Dieser Assembler kann die folgenden, von Prozessorherstellern nicht dokumentierten, aber bei den meisten Chips (herstellerabhängig) vorhandenen Instruktionen assemblieren:

5.1 SLIA / SLI / SLS

SLIA reg SLI reg SLS reg

Das Register reg wird um ein bit in Richtung Bit 7 geschoben, wobei Bit 7 ins Carry-Flag geschoben wird und Bit 0 auf 1 gesetzt wird (reg:=reg*2+1).

5.2 TSTI

TSTI oder TSTI (C)

Der Prozessor liest den Port C (Register C) und setzt die Statusflags entsprechend dem gelesenen Byte. Verändert werden folgende Flags: Sign, Zero, Parity, Halfcarry:=0, Subtraction:=0.
Diese Instruktion hat den Opcode der nicht existierenden Instruktion IN (HL),(C). Verwendet wird hier der Z800-Mnemonic.

5.3 Die Hälften der Indexregister IX und IY

Die beiden Hälften des IX- und des IY-Registers können als 8-bit-Register verwendet werden. LX oder IXL ist Bit 0-7 von IX, HX oder IXH ist Bit 8-15 von IX, LY oder IYL ist Bit 0-7 von IY, HY oder IYH ist Bit 8-15 von IY. In einer Instruktion können entweder LX,HX,IXL,IXH oder LY,HY,IYL,IYH verwendet werden, beide zusammen jedoch nie. Die gemeinsame Verwendung von LX,HX,LY,HY,IXL,IXH, IYL,IYH und H,L,HL,IX,IY ist nicht möglich. Diese Registerhälften können auch nicht in Instruktionen verwendet werden, die ein 0EDH oder 0CBH benötigen (z.B. OUT (C),reg). Eine Instruktion mit einer solchen Registerhälfte benötigt 4 Taktzyklen mehr als die gleiche Instruktion mit L/H.

Erlaubt ist z.B.:

```
LD      HX,17H          ADD     A,HY
INC     IXH             LD      C,LY
```

Nicht erlaubt ist z.B:

```
IN      LX,(C)          LD      HX,(HL)
LD      HX,HY           LD      L,HY
LD      (IY+7),IYH      LD      LX,(IY-20H)
BIT     5,LX            SLA     HY
```

6 Assembler-Direktiven

6.1 Direktiven, die Code erzeugen

6.1.1 DEFB oder DB, Define Byte

Das Argument wird ausgerechnet und wird als Einzelbyte in den Object-Code eingefügt. Falls das Ergebnis nicht als Einzelbyte darstellbar ist, wird eine Fehlermeldung ausgegeben. Es können auch mehrere Argumente, durch Kommata getrennt, angegeben werden. Texte in Anführungszeichen dürfen auch vorkommen. Außerdem kann DEFB bzw. DB weggelassen werden; dann müssen nur noch die Argumente in der Quelldatei stehen.

Beispiele:

```
            DEFB 1,2,3,4,5,6,7,8
            DB   "Ausgabe:",0DH,0AH,"--------",0DH,0AH
TWOPWR::    1,2,4,8,10H,20H,40H,80H
            DEFB "A",HIGH 1234H,"C"+80H,7 XOR 2
```

6.1.2 DEFW oder DW, Define Word

Die Argumente werden ausgerechnet und als Wörter in den Object-Code eingefügt. Ist das Ergebnis ein Einzelbyte, so werden die restlichen Bits mit 0 aufgefüllt. Die Argumente werden durch Kommata getrennt.

Beispiele:

```
TWOPWR::  DEFW 1,2,4,8,10H,20H,40H,80H,100H,200H,400H
          DW   "A"+5,"AB","CD"+8080H
```

6.1.3 DEFM oder DM, Define Message

Als Argument ist nur ein Text in Anführungszeichen erlaubt. Die Zeichen des Textes werden einzeln in den Object-Code eingefügt.

Beispiele:

```
            DEFM "Define Message"
            DM   'Eingabe:'
```

6.1.4 DEFC oder DC, Define Character

Als Argumente sind nur Texte in Anführungszeichen erlaubt. Falls mehrere Argumente angegeben werden, so müssen sie durch Kommata getrennt werden. Die Funktion ist ähnlich wie die von DEFM/DM, aber Bit 7 des jeweils letzten Zeichens wird auf 1 gesetzt.

Beispiel:

```
CMDS:  DEFC "FOR","GOTO",'NEXT','LET'
```

dies entspricht

```
CMDS:"FO","R"+80H,"GOT","O"+80H,"NEX","T"+80H,"LE","T"+80H
```

6.1.5 DEFH oder DH

Diese Direktive ist ähnlich wie DEFC/DC, aber es wird bei jedem Zeichen Bit 7 auf 1 gesetzt.

6.1.6 FILL, Speicherbereich mit Bytes füllen

FILL a,b BYTES a,b DEFS a,b

Das Argument b wird a-mal in den Object-Code eingefügt. Das Argument a kann 16 bit lang sein, das Argument b muß mit 8 Bits darstellbar sein.

Beispiel:

```
DISKBUFFER: FILL 128,0E5H
```

6.1.7 WORDS, Speicherbereich mit Worten füllen

WORDS a,b

Das Argument b wird a-mal als Wort in den Object-Code eingefügt. Beide Argumente sind 16 bit lang.

Beispiel:

```
DISKBUFFER: WORDS 64,0E5E5H     ; = BYTES 128,0E5H
```

6.1.8 *RELTAB, Tabellen für die Verschiebung von Programmen

*RELTAB parameter Relocation Table

Erzeugung einer Tabelle für Programme, die sich selbst verschieben. In dieser Tabelle werden die Adressen eingetragen, an denen Adressen stehen, die bei der Verschiebung geändert werden müssen. *RELTAB darf nicht benutzt werden, wenn eine RLD- oder REL-Datei erzeugt wird. Es können 127 verschiedene Tabellen erzeugt werden. Zwischen *RELTAB ON und *RELTAB OFF darf ORG nicht angewandt werden. Statt dessen ist DEFS zu verwenden.

Anwendung:

*RELTAB (n) ON
Ab hier steht das zu verschiebende Programmstück. Die Adressen werden in der Tabelle n (1..127) vermerkt. Fängt n nicht mit einer Ziffer an, so ist es in Klammern zu schreiben. Wird n weggelassen, so wird 1 dafür angenommen.

*RELTAB (n) ON, m
Da .PHASE/.DEPHASE innerhalb *RELTAB ON/OFF nicht angewandt werden darf, wird dieses Statement als Ersatz genommen. Das folgende Programmstück bis zum zugehörigen *RELTAB (n) OFF wird für die Adresse m assembliert.

*RELTAB (n) OFF
Hier endet das zu verschiebende Programmstück.

*RELTAB (n) HERE

Tabelle n (falls nicht angegeben: 1) an dieser Adresse erzeugen.

*RELTAB (n) HERE, x

Tabelle n (s.o.) so erzeugen, als ob das Programmstück, für das die Tabelle erzeugt wird, an der Adresse x stehen würde. Insbesondere kann hiermit die Tabelle so erzeugt werden, als ob das Programmstück bei 0 beginnen würde, womit in manchen Fällen die Berechnung der Adressen vereinfacht wird. Wurde bei *RELTAB ON ein m angegeben, so ist x durch x-a zu ersetzen, wobei a die Adresse ist, an der das *RELTAB ON steht (a:!*RELTAB ON,m).

*RELTAB (n) HERE, a..b

Tabelle n (s.o.) für alle Adressen, die zwischen a und b liegen (das m von *RELTAB ON, m wird hier nicht beachtet), erzeugen.

*RELTAB (n) HERE, x, a..b

Die Tabelle wird so erzeugt, als ob das Programmstück an der Adresse x stehen würde. Es werden nur Adressen eingetragen, die zwischen a und b liegen (s.o). Es kann auch

*RELTAB (n) HERE, a..b, x

angegeben werden.

Aufbau der Tabelle:

Es gibt zwei mögliche Typen dieser Tabelle, welche über die Direktive *OPT gewählt werden. Beim Typ RTD (Standard-Einstellung) wird das Ende der Tabelle mit dem Wert 0FFFFH markiert, beim Typ RTC beginnt die Tabelle mit einem Wort, das die Anzahl der folgenden Worte angibt. Das Ende ist dann nicht markiert.

Beispiel:

An den Adressen 1234H, 3141H und 5678H befinden sich Wörter, die beim Verschieben geändert werden müssen:

```
*RELTAB ON
ORG     1234H
DW      $
ORG     3141H
DW      $
ORG     5678H
DW      $
*RELTAB OFF
```

Dann ergeben sich folgende Tabellen:

```
*OPT    RTD         ; Standard-Einstellung
*RELTAB HERE
1234
3141
5678
FFFF

*OPT    RTC
*RELTAB HERE
0003
1234
3141
```

Nach einem *RELTAB HERE dürfen keine Labels mehr definiert werden!! Dies deshalb, weil die Länge der Tabelle erst unmittelbar vor deren Erstellung festliegt (im zweiten Durchlauf) und die Werte von danach definierte Labels somit zu Anfang des zweiten Durchlaufs nicht bekannt sind und somit auch nicht verwendet werden können. Dies wirft Probleme auf, falls man mehrere Tabellen verwendet, da die Anfangsadresse der zweiten und der folgenden nicht mit Labels benannt werden können und dürfen. Die Lösung besteht darin, daß man die Adresse 'at runtime' feststellt, d.h. das Programm selbst muß die Adresse herausfinden. Dies geschieht durch Suche nach FFFFH (*OPT RTD) oder durch Addition der Tabellenlänge (*OPT RTC). Nach Verwendung einer Tabelle steht der verwendete Zeiger ohnehin auf dem Beginn der nächsten, falls die Tabellen unmittelbar aufeinanderfolgen (dies ist auch sinnvoll, da danach keine Labels mehr definiert werden können). Man muß die Tabellen nur in der richtigen Reihenfolge anbringen!

Beispiel: (für TRS-80)

```
        .cmd
        org   5200h
himem   equ   4049h              ; Zeiger auf Speicherende
entry:  ld    hl,(himem)
        ld    de,-pgm_len
        add   hl,de
        ld    (himem),hl         ; Speichergrenze korrigieren
        inc   hl                 ; Adresse für Programm
        push  hl
        ld    de,-pgm_start
        add   hl,de              ; Versatzadresse
        ex    de,hl
        ld    hl,reloc_tab       ; Tabelle
        loop
         ld   c,(hl)
         inc  hl
         ld   b,(hl)
         inc  hl
         exitif bc=0ffffh        -->  Ende der Tabelle
         ld   a,(bc)             ; Adresse korrigieren
         add  a,e
         ld   (bc),a
         inc  bc
         ld   a,(bc)
         adc  a,d
         ld   (bc),a
        endloop
        ld    hl,pgm_start
        pop   de
        ld    bc,pgm_len
        ldir                     ; Verschieben
; Programmstück ist nun verschoben.
; ...

pgm_start:
      *reltab on
-------------------------------------------------
; Hier steht das zu verschiebende Programmstück ;
-------------------------------------------------
     *reltab off
pgm_len equ $-pgm_start

reloc_tab:
     *reltab here

     end  entry
```

6.2 Direktiven zur Label-Definition

6.2.1 EXTERNAL, Definition von externen Labels

EXTERNAL EXTRN EXT

Die Labels, die nach EXTERNAL stehen, werden als extern deklariert. Diese Direktive ist nur bei REL-, CRL- und RLD-Object-Dateien erlaubt. Bei CRL-Dateien dürfen diese Labels dann nur für Unterprogrammaufrufe verwendet werden, nicht für Daten (dies wird aber nicht vom Assembler überprüft!).

Beispiele:

```
EXTERNAL  INPUT,OUTPUT,GET,PUT
EXT       $CH,$MB
```

Labels können auch bei der Benutzung als extern vereinbart werden, indem ## angehängt wird (nicht bei CRL-Dateien). Wurde ein Label einmal mit einem ## benutzt, so ist es für das restliche Programm extern definiert. Dabei muß bei der ersten Benutzung auf jeden Fall ## angehängt werden.

6.2.2 GLOBAL, ENTRY und PUBLIC, Definition von globalen Labels

Die Labels, die nach GLOBAL stehen, werden global deklariert. Hiermit wird dem Linker der Zugriff auf diese Labels ermöglicht. Andere Programmoduln können auf diese Labels zugreifen. Diese Direktive ist nur bei REL- und RLD- Object-Dateien erlaubt. Durch GLOBAL wird den Labels kein Wert zugewiesen. Labels können auch bei der Definition als global deklariert werden (wenn ein doppelter Doppelpunkt folgt).

Beispiele:

```
GLOBAL    MUL,DIV
PUBLIC    MOD,PWR
```

6.2.3 EQU, Equate

label EQU value

Dem Label wird der Wert value zugewiesen. Der Modus hängt nicht vom Segment ab, in dem sich der PC befindet, sondern vom Modus des Arguments. Der Modus darf nicht extern sein. Der Wert eines Labels kann mit EQU nicht verändert werden (siehe DEFL). Nach dem Label darf kein Doppelpunkt folgen. Somit muß das Label, falls es global sein soll, zuerst mit GLOBAL deklariert werden. Alle Labels, die im Argument verwendet werden, müssen schon definiert sein.

Beispiele:

```
zwei      EQU   2
here      EQU   $
LENGTH    EQU   $-START
FALSE     EQU   0
TRUE      EQU   NOT FALSE
TEST      EQU   TRUE
```

6.2.4 DEFL, DL, ASET und SET, Define Label

label DEFL DL ASET SET n

Dem Label wird der Wert n zugewiesen. Der Modus hängt nicht vom Segment ab, in dem sich der PC befindet, sondern vom Modus des Arguments. Er darf nicht extern sein. Mit DEFL kann der Wert eines schon definierten Labels geändert werden. Falls das Label global sein soll, muß es mit GLOBAL deklariert werden. SET ist nur erlaubt, wenn der F-Switch angegeben wurde. Falls SET in diesem Sinne gebraucht werden soll, muß der F-Switch angegeben werden; soll SET als Label benutzt werden, so darf der F-Switch nicht angegeben werden. Sämtliche SET-Statements müssen dann durch DEFL ersetzt werden. Wenn das Label global ist, so hat es beim Linken den Wert, den es beim Assemblieren zuletzt hatte.

Beispiele:

```
COUNT     DEFL      COUNT+1

maxval    dl   -32768
max       macro     n
          if n > maxval
maxval    aset n
          endc
          endm

size      defl      $-100H
```

6.2.5 LOCAL, Definition von lokalen Labels

Die Labels, die bei LOCAL angegeben werden, sind für den Macro, in dem sich das Statement befindet, lokal vereinbart. Dies dient dazu, daß Labels, die in einem Macro definiert werden, nicht (unerlaubterweise) mehrfach definiert werden, wenn der Macro mehrfach aufgerufen, oder der Wiederholungsblock mehrfach durchlaufen wird. Die Labels werden durch Labels der Form ..xxxx ersetzt, wobei xxxx eine Hexadezimalzahl ist. Ein Macro kann maximal 64 lokale Labels besitzen, die jedoch beliebig viele Labels ergeben können. LOCAL kann auch in REPT und IRP/IRPC angewandt werden, wobei bei jedem Schleifenanfang die lokalen Labels neu generiert werden. Mit lokalen Labels können wie mit Macro-Parametern neue Labels zusammengesetzt werden.

Beispiele:

```
          local  lbl1,lbl2,NEXT

print macro text
      local $$$text,$$$skip
      ld    de,$$$text
      call  print_string
      jr    $$$skip
$$$text:db   text
$$$skip:
      endm
```

6.3 Direktiven die PC verändern

6.3.1 ORG, Set Origin

ORG n

PC wird auf n gesetzt. Falls PC in einem Code-, Daten- oder COMMON-Segment (relativ zum Programm- oder Datensegment) ist, darf n eben diesen Modus nicht haben. N darf immer absolut sein. Der Segmenttyp läßt sich nicht mit ORG wechseln (siehe ASEG, CSEG, DSEG, COMMON). Wird eine Intel-Hex-Datei, eine CMD-Datei oder eine RLD-Datei erzeugt, dann wird ein neuer Record angefangen. Bei einer COM-Datei wird mit 0 aufgefüllt. Bei einer COM-Datei wird nur der Code erzeugt, der an Adressen steht, die größer oder gleich 100H sind, d.h. es muß zu Anfang ein ORG 100H stehen.

Beispiele:

```
        ORG     100H
        ORG     $+100H
        ORG     start_address
```

Beim letzten Beispiel muß darauf geachtet werden, daß das verwendete Label schon vor dem ORG definiert wird (im ersten Durchlauf)!
ORG ist nicht innerhalb von PHASE-Blöcken und *RELTAB ON/*RELTAB OFF erlaubt.

6.3.2 DEFS und DS, Define Space

DEFS n DS n

Es werden n Bytes im Object-Code freigelassen. Bei einer COM-Datei wird n mal 0 eingefügt. Bei den anderen Object-Dateitypen wird ein neuer Record angefangen.

DEFS n,m DS n,m

Es werden n Bytes mit dem Byte m gefüllt (siehe FILL).

Beispiele:

```
        DEFS    256
x:      DS      2,0ffh
```

Alle verwendeten Labels müssen schon definiert sein!

6.4 Direktiven zur Segmentwahl

6.4.1 CSEG, Code Segment

Der Code, der nach CSEG erzeugt wird, ist relativ zum Programmsegment. Beim Linken wird der Code somit ins Programmsegment geschrieben. CSEG ist nur erlaubt, wenn eine REL-Datei erzeugt wird. Wenn eine REL- oder RLD-Datei erzeugt wird, ist CSEG Standard-Einstellung.

6.4.2 DSEG, Data Segment

Der Code, der nach DSEG erzeugt wird, ist relativ zum Datensegment. Beim Linken wird der Code somit ins Datensegment geschrieben. DSEG ist nur erlaubt, wenn eine REL-Datei erzeugt wird.

6.4.3 ASEG, Absolute Segment

Der Code, der nach ASEG erzeugt wird, ist absolut. Beim Linken wird der Code somit an die Adresse geschrieben, die bei ORG (nach ASEG) angegeben wurde. ASEG ist Standard-Einstellung, wenn keine REL- bzw. RLD-Datei erzeugt wird.

Das Segment kann beliebig oft innerhalb eines Programmes gewechselt werden. Der Assembler merkt sich die jeweiligen Werte der drei PCs.

6.4.4 COMMON, COMMON-Block definieren

COMMON /segment/

Der Code, der nach COMMON erzeugt wird, hat den Modus COMMON. Beim Linken wird der Code somit in den bei COMMON angegebenen Bereich 'segment' geschrieben. Der unbenannte COMMON-BLock wird durch // gekennzeichnet. COMMON ist nur erlaubt, wenn eine REL-Datei erzeugt wird. Bei jedem Aufruf eines COMMON-Blocks wird der PC auf 0 gesetzt (auch wenn der gleiche COMMON-Block mehrfach aufgerufen wird).

6.4.5 .PHASE und .DEPHASE, Assemblierung für andere Adressen

.PHASE n .DEPHASE

Der Code, der zwischen .PHASE und .DEPHASE steht, wird an der aktuellen PC-Position (und im aktuellen Segment) eingefügt, aber er wird für die (absolute) Adresse n assembliert. Dies ist nützlich, wenn ein Teil des Programmes an eine andere Adresse verschoben werden soll, die zur Zeit der Assemblierung bekannt ist. Wenn diese nicht bekannt ist, so muß *RELTAB verwendet werden. Der Code zwischen .PHASE und .DEPHASE ist immer absolut. Die Labels, die in einem .PHASE-Block definiert werden, haben den Wert, den der PC hätte, wäre der .PHASE-Block für sich alleine an der Adresse n assembliert worden. Wenn von außerhalb Labels innerhalb von .PHASE/.DEPHASE vor dem Verschieben angesprochen werden sollen, so muß die Adresse, an der der Block begonnen worden ist, abzüglich n, zu den Labels addiert werden. PHASE-Blöcke dürfen nicht ineinander geschachtelt werden. ASEG, CSEG, DSEG, COMMON und ORG sind innerhalb eines solchen Blockes nicht erlaubt.

Beispiel:

```
        org  100h

dst     equ  8000h          -- Dorthin soll das Programm


        ld   hl,pgm
        ld   de,dst
        ld   bc,pgmend-pgm
        ldir                -- Verschieben
        jp   dst            -- Anspringen
-------------------------------------------------
pgm:
        .phase dst
        --
        -- Hier steht das zu verschiebende Programmstück
        --
        -- z.B.
        ld   hl,100h
loop:ld  (hl),0e5h
        inc  l
        jr   nz,loop
        jp   0
        -- Ende des zu verschiebenden Programmstücks
        .dephase
pgmend:
```

6.4.6 Direktiven zur Ausgabe auf den Bildschirm

6.4.7 .PRINTX, Textausgabe

.PRINTX delim text delim

Der Text, der zwischen den Begrenzern 'delim' liegt, wird mit den Begrenzern ausgegeben.

Beispiele:

```
.PRINTX  -Achtung-
.printx  *Ende des ersten Teils*
if2!.printx "Pass 2"!endif
```

6.4.8 .PRINTL, Textausgabe

.PRINTL text

Der Text, der nach .PRINTL steht, wird auf den Bildschirm ausgegeben.

Beispiele:

```
.PRINTL  Teil Nr. 1 assembliert.

ifb <paraml>
.printl  Unerlaubter Parameter für Macro XXX!
endif

if ende-anfang > 1000h
.printl Das Programm ist zu lang
endif
```

6.4.9 .PRINTN, Zahlenausgabe

.PRINTN Ausdruck

Das Argument von .PRINTN wird in der aktuellen Listing-Zahlenbasis auf den Bildschirm ausgegeben (siehe .LRADIX). Das Argument muß absolut sein.

Beispiel:

```
.PRINTL  Länge des Programmes:
.printn  $-ANFANG
```

6.5 Direktiven zur Listing-Gestaltung

6.5.1 .LRADIX, Zahlenbasis für Listing

.LRADIX n

Die Listing-Zahlenbasis wird auf n geändert. N wird immer dezimal angegeben. Erlaubte Werte sind 2, 8, 10 und 16. Diese Zahlenbasis wird auch bei .PRINTN benutzt (s.o.).

Beispiel:

```
.LRADIX 10
.printn ende-anfang
.lradix 16
```

6.5.2 TITLE, Überschrift

TITLE text *TITLE 'text'

Der Text 'text' wird als Überschrift im Listing verwendet. Er wird in der Zeile der Assembler-Überschrift gedruckt (vor derselben).

Beispiel:

```
TITLE I/O
```

6.5.3 SUBTTL, Überschrift

SUBTTL text $TITLE ('text') *TITLE ('text')

Der Text 'text' wird unter der Überschrift im Listing gedruckt.

Beispiele:

```
$TITLE ('Ein- und Ausgabe')
SUBTTL
-- Überschrift löschen.
```

6.5.4 PAGE, Seitenvorschub

.EJECT n $EJECT n *EJECT n PAGE n

Es wird ein Seitenvorschub ins Listing eingefügt. Danach wird die Überschrift neu gedruckt.
Wird ein Wert dahinter angegeben, so wird kein Seitenvorschub erzeugt und die Seitenlänge entsprechend eingestellt, wobei n größer als 5 sein muß. Wenn n auf 0 gesetzt wird, ist die Seitenlänge unendlich, d.h. es erfolgt nie ein automatischer Seitenumbruch. Die Standardeinstellung für die Seitenlänge ist 50.

Beispiele:

```
$eject                  ; Nur Seitenvorschub
PAGE 72                 ; Kein Seitenvorschub, Seitenlänge nun 72
```

6.5.5 .LIST, Listing einschalten

.LIST *LIST ON

Das Listing wird eingeschaltet (Standardeinstellung).

6.5.6 .XLIST, Listing ausschalten

.XLIST *LIST OFF

Das Listing wird ausgeschaltet. Das Wiedereinschalten des Listings geschieht mit .LIST oder *LIST ON. Die Symbol table im Listing kann durch

6.5.7 .LALL, Listingkontrolle bei Macro-Expansion

.LALL *MACLIST ON List Macro Text

Das Listing der Macro-Expansion wird eingeschaltet. Es werden alle Zeilen gelistet.

6.5.8 .XALL, Listingkontrolle bei Macro-Expansion

.XALL Exclude Noncode Macro Lines

Der erzeugte Code eines Macros wird gelistet. Die Source-Statements werden nur gelistet, wenn Code erzeugt wird. Hierdurch wird das Listing verkürzt, in dem z.B. IF, ENDIF und EQU-Statements weggelassen werden und nur die Statements gelistet werden, die tatsächlich Code erzeugen. (.XALL ist Standardeinstellung).

6.5.9 .SALL, Listingkontrolle bei Macro-Expansion

.SALL *MACLIST OFF Suppress Macro Listing

Es wird nur der erzeugte Code gelistet. Im Listing erscheint nur der Macro-Aufruf und der vom Macro erzeugte Code, ein Macro erscheint also so, als ob er ein einziges Statement wäre.

6.5.10 .LFCOND, Listingkontrolle bei bedingter Assemblierung

.LFCOND List False Conditionals

Es wird auch der Teil eines IF-Blocks gelistet, der nicht assembliert wird (Standardeinstellung).

6.5.11 .SFCOND, Listingkontrolle bei bedingter Assemblierung

.SFCOND Suppress False Conditionals

Es wird nur der Teil des IF-Blocks gelistet, der assembliert wird.

6.5.12 .TFCOND, Listingkontrolle bei bedingter Assemblierung

.TFCOND Toggle False Listing Conditionals

Umschaltung zwischen .SFCOND und .LFCOND.

6.5.13 .CREF und .XCREF, nicht implementiert

.CREF und .XCREF

Die Direktiven für ein Cross-Reference-Listing sind (noch) nicht implementiert und werden ignoriert.

6.5.14 *SPACE, Leerzeilen ins Listing einfügen

*SPACE n

Es werden n Leerzeilen ins Listing eingefügt. *SPACE selbst erscheint nicht im Listing. Ohne n erzeugt *SPACE eine Leerzeile. Die Zeile, in der *SPACE steht, wird nicht gelistet.

Beispiel:

```
*SPACE 4
```

6.6 Direktiven für bedingte Assemblierung

Ein IF-BLock wird mit dem IF-Statement eingeleitet. Falls die Bedingung zutrifft, werden die folgenden Statements ausgeführt. Falls die Bedingung nicht zutrifft und ein ELSE folgt, werden die Statements nach ELSE ausgeführt. Falls kein ELSE folgt, wird nach dem ENDIF/ENDC fortgefahren. IF-Blöcke können beliebig geschachtelt werden.

IF expression	Wahr, wenn das Argument nicht 0 ist.
IFT expression	Wahr, wenn das Argument nicht 0 ist.
COND expression	Wahr, wenn das Argument nicht 0 ist.
IFE expression	Wahr, wenn das Argument 0 ist.
IFF expression	Wahr, wenn das Argument 0 ist.
IF1	Wahr, wenn im ersten Durchlauf.
IF2	Wahr, wenn im zweiten Durchlauf.
IFDEF label	Wahr, wenn das Label definiert ist.
IFNDEF label	Wahr, wenn das Label undefiniert ist.
IFIDN <arg1>,<arg2>	Wahr, wenn arg1 = arg2 ist.
IFDIF <arg1>,<arg2>	Wahr, wenn arg1 <> arg2 ist.
IFB <argument>	Wahr, wenn das Argument leer ist.
IFNB <argument>	Wahr, wenn das Argument vorhanden ist.

Beispiele:

```
IF1!.PRINTX-Pass1-!ELSE!.PRINTX-Pass2-!ENDC

IF TEST
ORG 8000H
ELSE
ORG 0100H
ENDIF

IFNDEF temp
temp:
ENDC

IFNB <arg3>
DEFB arg3
ENDIF

if $ > 4000h
 .printl Programm zu lang!
endif

Output  MACRO     reg
IFDIF <reg>,<A>
        LD        A,reg
ENDIF
        CALL      Output_A
        ENDM
```

6.7 Die Macro-Direktiven

6.7.1 MACRO, Definition eines Macros

macrosymbol MACRO parameters

Hiermit wird ein Macro definiert. Er erhält den Namen macrosymbol. Für einen Macro-Namen gelten die gleichen Vorschriften wie für Labels (siehe oben). Ein Macro darf den gleichen Namen haben wie ein Label. Ebenso ist die Verwendung von Namen von Z80-Instruktionen sowie Direktiven erlaubt. Durch die Definition eines Macros mit dem Namen einer Z80-Instruktion oder einer Direktive werden letztere durch den Macro verdeckt, d.h. bei Verwendung wird der Macro aufgerufen und die ursprüngliche Funktion geht verloren. Die Angabe von Parametern ist optional. Die Namen der Parameter gelten als lokal vereinbart. Die Definition eines Macros wird mit ENDM beendet.

6.7.2 EXITM, Abbruch einer Macro-Expansion

EXITM Exit Macro

Wird EXITM während der Expansion eines Macros angetroffen, dann wird der Macro verlassen, d.h. es wird mit dem Statement nach dem Macro-Aufruf fortgefahren. Wird EXITM in einem IF-Block angetroffen, so wird der IF-BLock und dann der Macro verlassen. Es werden solange IF-Blöcke verlassen, bis ein Macro-Block verlassen werden kann.

6.7.3 ENDM, Ende einer Macro-Definition

Mit ENDM wird die Definition eines Macros abgeschlossen. Das Ende eines REPT-, IRP- oder IRPC-Blockes wird ebenfalls mit ENDM markiert. Falls noch offene IF-Blöcke vorhanden sind, werden diese automatisch geschlossen.

6.7.4 REPT, Feste Anzahl von Wiederholung von Statements

REPT n Repeat

Es wird der folgende Block (bis zum zugehörigen ENDM) n-mal wiederholt. Im Block dürfen Labels lokal vereinbart werden (siehe LOCAL). Wenn n=0 ist, wird der Block keinmal assembliert.

6.7.5 IRP, Wiederholung mit verschiedenen Parametern

IRP dummy,<parameter-list> Indefinite Repeat

Im einfachen Fall folgt nach IRP eine Dummy-Variable, die lokal definiert ist. Ihr werden dann bei der Assemblierung des Blockes die Werte zugewiesen, die in der Parameter-Liste stehen. Im allgemeinen Fall können beliebig viele Dummy-Variablen angegeben werden (im folgenden: n Stück). Bei der ersten Assemblierung des Blockes werden den Dummy-Variablen die ersten n Parameter zugewiesen, bei der nächsten Wiederholung die nächsten n Parameter u.s.w. . Der Block muß mit ENDM beendet werden. Es sind lokale Labels erlaubt. Ist die Parameter-Liste leer, so wird der Block einmal mit leeren Dummy-Variablen assembliert. Die spitzen Klammern müssen angegeben werden. Die einzelnen Parameter werden durch Kommata oder Leerzeichen voneinander getrennt.

6.7.6 IRPC, Wiederholung mit verschiedenen Parametern

IRPC dummy,<parameter-list> Indefinite Repeat Character

IRPC dient der Wiederholung eines Blockes mit verschiedenen Parametern, die jeweils genau ein Zeichen lang sind. Die spitzen Klammern sind hier optional. Es ist nur eine Dummy-Variable erlaubt. Ihr werden nacheinander die einzelnen Zeichen der Parameter-Liste zugeordnet. Die einzelnen Zeichen werden nicht durch Kommata getrennt. Ist die Parameterliste leer, so wird der Block einmal assembliert. Die Dummy-Variable ist dann leer. Lokale Labels sind möglich.

6.7.7 Kommentare in Macros

;; Kommentar

Ein Kommentar, der mit zwei Strichpunkten beginnt, wird nicht in den Macro übernommen. Diese Kommentare erscheinen dann nicht bei der Macro-Expansion. Alle anderen Kommentare werden in den Macro übernommen und benötigen demzufolge mehr Speicherplatz.

6.7.8 Labels mit '&' zusammensetzen

Wird ein & in der Quelldatei angetroffen, so wird überprüft, ob ein Dummy-Parameter folgt. Wenn dem so ist, wird er eingesetzt und das & entfernt. Folgt auf den Dummy-Parameter ein weiteres &, ohne daß darauf ein Dummy-Parameter folgt, so wird dieses ebenfalls entfernt. Damit lassen sich neue Labels zusammensetzen und Parameter in Texte einsetzen. Außerhalb von Texten ist & nicht nötig (falls kein in Labels erlaubtes Zeichen unmittelbar an den Dummy-Parameter anschließt). Soll nach dem Dummy-Parameter ein & stehen, so ist dieses doppelt zu schreiben. Diese Ausführungen treffen auch auf lokale Labels zu.

Beispiele:

```
text      MACRO     num,txt,comment
text&num: DB "&txt",0DH,0AH,"$" comment
          ENDM

text 5,Hallo!!!!,<; Beispiel 1>
          ergibt:
text5: DB "Hallo!!",0DH,0AH,"$" ; Beispiel 1

text 7,<Eins, zwei, drei>
          ergibt:
text7: DB "Eins, zwei, drei",0DH,0AH,"$"

text ,0
          ergibt:
text: DB "0",0DH,0AH,"$"
```

Beispiele zur Entfernung des & (x werde durch abc ersetzt):

Dummy-Parameter	ergibt
&x&	abc
&x&&	abc&
x&x	abcabc
&&x&&x	&abc&abc
x&d	abcd

Zu beachten ist hier, daß innerhalb von Anführungszeichen ein & vorangestellt werden muß, falls nicht schon vorhanden.

6.7.9 Macro-Parameter

Zu den Parametern von IRP/IRPC und den Parametern beim Aufruf eines Macros:

Die Parameter der Parameter-Liste werden durch Kommata getrennt. Soll ein Sonderzeichen, z.B. ein Komma oder ein Semikolon als Parameter übergeben werden, so ist ein Ausrufezeichen voranzustellen (!, bzw. !;). Das Ausrufezeichen bewirkt, daß das nächste Zeichen direkt übernommen wird. Leerzeichen werden vor den Parametern entfernt. Mit '! ' kann ein Leerzeichen erzwungen werden. Soll eine ganze Liste von Parametern (zum Beispiel Argumente, die vom Macro an einen weiteren Macro übergeben werden sollen) oder ein Parameter, der Leerzeichen oder Kommata enthält, übergeben werden, so ist diese in spitze Klammern zu setzen, damit die ganze Liste einem Dummy-Parameter zugewiesen wird. Spitze Klammern lassen sich mit !< und !> übergeben. Spitze Klammern und Ausrufezeichen lassen sich beliebig schachteln. Beim Aufruf eines Macros kann das Ausrechnen und das Ersetzen eines Parameters durch eine Zahl erzwungen werden, indem ein % davor gesetzt wird. Die Zahl wird in der zur Zeit eingestellten Zahlenbasis dargestellt. Soll ein <opcode>-Argument an einen Macro übergeben werden, so muß es nochmals zwischen spitze Klammern geschrieben werden. Bei jeder Übergabe eines Parameters an einen Macro wird ein Paar spitze Klammern (falls vorhanden) entfernt. Ebenso wird bei jedem Macro-Aufruf das Zeichen nach einem Ausrufezeichen eingesetzt. Soll ein solches Zeichen an tiefer geschachtelte Macros übergeben werden, so ist eine entsprechende Anzahl von Ausrufezeichen voranzustellen. Ein Ausrufezeichen läßt sich übergeben, indem ein weiteres vorangestellt wird.

Nun folgen Beispiele zu den Macro-Parametern:

```
REPT 4
LD   (HL),"-"
INC  HL
ENDM

IRPC char,Test
LD   (HL),char
INC  HL
ENDM

IRP arg,addr,<*,mul,+,add,-,sub,/,div,!,,komma>
CP   "&arg"
JP   Z,addr
ENDM
```

6.7.10 Beispiele

Beispiel: Berechnung des größten gemeinsamen Teilers

```
ggt  macro m,n
;; Größter gemeinsamer Teiler von m und n -> ggtn
ggtm dl    m          ;; Parameter können nicht verändert werden
ggtn dl    n
     rept  65535      ;; Wird vorher abgebrochen
ggtx dl    ggtm/ggtn
ggtr dl    ggtm-ggtx*ggtn
     if    ggtr = 0
     exitm            ;; Fertig
     endc
ggtm dl    ggtn
ggtn dl    ggtr
     endm
     endm
```

Beispiel: Fakultät, rekursiv

```
fac  macro n
;; Fakultät von n berechnen. Ergebnis -> facr
     if      n < 2
facr dl      1
     else
     fac     %n-1
facr dl facr*n
     endc
     endm
```

```
      CHECK MACRO REG
      INC   REG
      DEC   REG
      ENDM

PRINT MACRO TEXT
      LOCAL $TEXT,CONT
      LD    DE,$TEXT
      LD    C,9
      CALL  5
      JP    CONT
$TEXT:DB    TEXT,"$"
CONT:
     ENDM

; Das gleiche nochmal:
PRINT2  MACRO TEXT
        LD DE,*TEXT,"$"
        LD C,9
        CALL 5
        ENDM
```

6.8 Sonstige Direktiven

6.8.1 INCLUDE, Einfügen einer Datei

```
MACLIB Dateiname
INCLUDE Dateiname
*INCL Dateiname
*INCLUDE Dateiname
$INCLUDE Dateiname
```

Die Datei mit dem Namen 'Dateiname' wird nach der aktuellen Zeile eingefügt. Wenn das Ende der Datei erreicht ist, wird an der Stelle in der vorigen Datei weitergelesen, wo die Einfügung stattfand. Die Zeilennumerierung mit dem L-Switch beginnt bei der eingefügten Datei wieder bei 00001. Bei der Rückkehr in die aufrufenden Datei wird mit der Zeilennummer fortgefahren, bei der der Aufruf erfolgte. Die eingefügte Datei darf ihrerseits wieder weitere Dateien einfügen. Bei gesetztem V-Switch wird die Einfügung durch < bzw. > angedeutet. Falls kein Dateityp angegeben wird, setzt der Assembler MAC ein.

6.8.2 LINK, Anhängen einer Datei

```
LINK Dateiname
.CHAIN Dateiname
```

Es wird mit der Datei mit dem Namen Dateiname.MAC fortgefahren. In dieser Datei darf wiederum ein LINK-Statement auftreten. Die Zeilennumerierung mit dem L-Switch beginnt wieder bei 00001. Hiermit können beliebig viele Dateien (natürlich durch den für die Symbol table verfügbaren Speicher begrenzt) verkettet werden. Bei gesetztem V-Switch wird ein ! ausgegeben. Dieser Befehl kann nicht verwendet werden, wenn die Eingabe von der Tastatur erfolgt.

Beispiele:

```
LINK TEIL2
.CHAIN TEIL3.ASM    -- Wird nicht durch .MAC ersetzt!
```

6.8.3 COMLIB, Binär-Datei einfügen

```
COMLIB Dateiname,n
```

Aus der angegebenen Datei werden n Bytes gelesen und an der aktuellen Position des PC eingefügt. Bei gesetztem V-Switch wird alle vier Bytes ein : ausgegeben. Dieser Befehl ist nützlich, falls eine Tabelle ins Programm eingefügt werden soll, die von einem anderen Programm erzeugt wurde, oder deren Assemblierung sehr lange dauern würde (z.B. falls sie mit REPT oder IRP erzeugt wird).

Beispiel:

```
COMLIB TAB1,256    -- Liest 256 Bytes aus TAB1
```

6.8.4 SYMLIB, Label-Datei lesen

SYMLIB Dateiname oder SYMLIB Dateiname,masks

Die in der Datei stehenden Labels werden in die Symbol table übernommen. Als Dateityp wird .SYM verwendet. Falls Masken angegeben wurden (die einzelnen Masken werden durch Kommata getrennt), so werden nur diejenigen Labels übernommen, die mindestens einer Maske entsprechen. Eine Maske besteht aus einem Label, das optional von einem * gefolgt werden kann. Ist dem so, dann paßt diese Maske auf alle Labels, die bis zum * mit ersterer übereinstimmen. Z.B. werden mit GET* alle Labels erfaßt, die mit GET anfangen. Es können beliebig viele Masken angegeben werden. Dieser Befehl wird nur im ersten Durchlauf durchgeführt, im zweiten dagegen ignoriert.

Beispiel:

```
SYMLIB DATEI1,START,ENDE,X*,Y*
SYMLIB DATEI2
```

6.8.5 *ERROR, Ausgabe einer Fehlermeldung

*ERROR n

Hierdurch wird die Fehlermeldung mit der Nummer n erzeugt (Tabelle im Anhang). Der Bereich reicht von 1 bis 52.

*ERROR OFF *ERROR ON

Hiermit kann man die Fehlermeldungen aus- und wiedereinschalten. Nach *ERROR OFF werden keine Fehlermeldungen mehr ausgegeben (außer Schreib-/ Lesefehler und Speicherüberlauf) und gezählt. Mit *ERROR ON wird der Normalzustand wiederhergestellt.

6.8.6 *OPT, Optionen einstellen

*OPT option

Es gibt folgende Optionen:

LJ	Bei LOOP wird ein JP zum Verlassen der Schleife verwendet.
LR	Bei LOOP wird ein RET zum Verlassen der Schleife verwendet. Diese beiden Optionen dürfen nicht innerhalb einer solchen Schleife geändert werden! Weiteres siehe LOOP.
RTD	Das Ende der mit *RELTAB HERE erzeugten Tabelle wird mit 0FFFFH markiert.
RTC	Am Anfang der Tabelle steht die Länge derselben (siehe *RELTAB).

6.8.7 .RADIX, Einstellung der Zahlenbasis

.RADIX n

Die Zahlenbasis wird auf n eingestellt, wobei n immer dezimal angegeben wird. N darf Werte zwischen 2 und 36 annehmen. Die Standardeinstellung ist 10. Bei Zahlenbasen über 10 werden die Ziffern mit Werten über 9 mit den Buchstaben ab 'A' dargestellt.

6.8.8 .REQUEST, Bibliotheken für Linker anmelden

.REQUEST Dateinamen

Nur bei REL-Dateien erlaubt. Es wird dem Linker die Anweisung gegeben, daß er, falls noch Labels nicht definiert sind, die angegebenen Dateien durchsucht. Es können mehrere Dateinamen angegeben werden, die durch Kommata getrennt werden. Für die Dateinamen gelten die gleichen Bestimmungen wie für Labels (!), insbesondere kann kein Dateityp angegeben werden.

Beispiel:

```
        .REQUEST Graphik,Mathe,Input,Output
```

6.8.9 .COMMENT, Mehrzeilige Kommentare

.COMMENT delim

Alle folgenden Zeilen werden als Kommentar aufgefaßt und somit ignoriert, bis in einer Zeile das Zeichen 'delim' auftaucht.

Beispiel:

```
        .COMMENT +
        Diese Zeilen
        werden ignoriert.
        ++++ Hier geht's weiter ++++
```

6.8.10 NOTICE, Kommentarblöcke

NOTICE delim

Gleiche Funktion wie .COMMENT, aber bei der Erzeugung von CMD-Dateien werden die Zeichen bis zum nächsten Auftreten von 'delim' als Kommentarblock in die Object-Datei eingefügt.

Beispiel:

```
                NOTICE X
                Copyright (c) 1985 by me
                Version 1.0
                X
```

6.8.11 NAME, Modul benennen

NAME ('text')

Nur bei der Erzeugung einer REL-Datei erlaubt. Als Name des Moduls wird 'text' verwendet. Ist kein NAME-Statement vorhanden, so wird der Text von TITLE als Name verwendet. Wurde kein TITLE-Statement angegeben, so wird der Name der Quelldatei dafür verwendet.

Beispiel:

```
NAME ('UP1')
```

6.8.12 .8080, Test auf 8080-Kompatibilität

Der Assembler überprüft, ob der erzeugte Object-Code von einem 8080 ausgeführt werden kann. Wenn nicht, so erfolgen Fehlermeldungen. SIM und RIM sind erlaubt, der Compiler erzeugt nur absolute Sprünge.

6.8.13 .Z80, Aufhebung von .8080

Hebt .8080 auf (Standardeinstellung). SIM und RIM sind nun nicht mehr erlaubt, der Compiler versucht, relative Sprünge zu erzeugen.

6.8.14 .SYMLEN, Label-Länge einstellen

.SYMLEN n Symbol Length

Der Assembler wird angewiesen, die Labels nach dem n-ten Zeichen abzuschneiden. Dies ist manchmal aus Kompatibilitätsgründen erforderlich. N darf Werte zwischen 3 und 32 annehmen. Standardeinstellung sind 32 Zeichen.

6.8.15 .EXTLEN, Label-Länge für REL-Datei einstellen

.EXTLEN n External Length

Die Labels, die in die REL-Datei geschrieben werden (GLOBAL, EXTERNAL und .REQUEST), werden nach dem n-ten statt dem siebenten Zeichen abgeschnitten. Zulässige Werte für n sind 3 bis 7.

6.8.16 END, Ende des Programmes, Startadresse

END n

END markiert das Ende des Programmes. Alle Zeilen danach werden ignoriert. Bei COM-Dateien wird n ignoriert. Bei anderen Object-Dateien wird der Entrypoint auf n gesetzt. END muß nicht vorhanden sein. Wenn END nicht vorhanden ist, wird der Entrypoint auf 0 gesetzt und folgende Meldung ausgegeben:

```
No END statement encountered in input file
```

6.8.17 MODEND, Ende eines Moduls

MODEND n

MODEND hat eine ähnliche Funktion wie END, es wird aber nach dem Abschluß des zweiten Durchlaufs nach MODEND weiterassembliert. Dies dient der Erstellung von Unterprogramm-Bibliotheken und ist somit nur bei der Erzeugung von REL-Dateien sinnvoll. Alle Macros und Labels werden ungültig, die Wirkung ist, als ob ein separates Programm assembliert werden würde, mit dem Unterschied, daß bei REL-Dateien weitergeschrieben wird. Wenn MODEND verwendet wird, ist die Benutzung von LINK nicht mehr erlaubt. Statt dessen ist INCLUDE zu verwenden, wobei jede Datei, die eingefügt wird, durch ein MODEND abzuschließen ist. Die letzte eingefügte Datei kann mit END aufhören.

6.8.18 .AG, Alle Labels global

Wenn .AG angegeben wird, werden alle Labels, die nicht extern sind, als globale Labels in die RLD- oder REL-Datei geschrieben. In der symbol table erscheinen allerdings nur die Labels als global, die mit einem GLOBAL-Statement als solche deklariert wurden. Dieses Feature wird benötigt, falls ein symbolischer Debugger verwendet wird und der Linker die Label-Datei erzeugt, denn hierzu müssen die Labels dem Linker bekannt gemacht werden.

6.8.19 .SLI, Special-LINK-Item erzeugen

.SLI n

.SLI ist nur bei der Erzeugung einer REL-Datei erlaubt und erzeugt den Special-LINK-Item mit der Nummer n. Danach müssen, falls erforderlich, noch das A-Field und das B-Field angegeben werden (in dieser Reihenfolge, letzteres in Anführungszeichen). Die Argumente werden durch Kommata getrennt.

6.8.20 .IF, Mnemonics WHEN/IF vertauschen

Die Mnemonics für das Verzweigungs-Statement und für die bedingte Assemblierung werden ausgetauscht. Weitere .IFs werden ignoriert. Es werden folgende Mnemonics vertauscht:

IF IFT COND	<->	WHEN
ELSE	<->	OTHERWISE
ENDIF ENDC	<->	ENDWHEN ENDW

6.8.21 *PAUSE, Assemblierung anhalten

Es wird folgende Meldung ausgegeben:

-------- Pause --------

Dann wird auf die Eingabe eines Zeichens gewartet. Ist diese erfolgt, dann wird mit der Assemblierung fortgefahren. Wenn CTRL-C eingegeben wird, dann wird die Assemblierung abgebrochen. Das Auftreten eines Fehlers bei gesetztem W-Switch bewirkt das gleiche wie *PAUSE. In der TRS-80-Version wird statt CTRL-C der Pfeil nach oben benutzt. Der Assembler läßt sich in dieser Version jederzeit durch Drücken der Pfeil-nach-rechts-Taste anhalten und mit dem Pfeil nach oben abbrechen.

6.8.22 .RLD, RLD-Datei erzeugen

.RLD .MZAL

Es wird eine RLD- und eine CMD-Datei erzeugt (für TRS-80). Diese Dateien sind kompatibel zu den von M-ZAL erzeugten und können somit mit dem zugehörigen Linker verarbeitet werden.

6.8.23 .CMD, CMD-Datei erzeugen

Es wird eine CMD-Datei erzeugt (für TRS-80).

6.8.24 .COM, COM-Datei erzeugen

Es wird eine COM-Datei erzeugt (für CP/M). Dies wird auch benötigt, um reine Code-Dateien (ohne dazwischen eingestreute Ladeinformationen für das Betriebssystem) zu erzeugen, um z.B. Overlays zu implementieren.

6.8.25 .HEX, HEX-Datei erzeugen

Es wird eine HEX-Datei erzeugt (Intel-Hex-Format). Dies wird benötigt, falls unter CP/M die Ladeadressen in der Quelldatei nicht in aufsteigender Reihenfolge angeordnet sind (muß dann mit DDT geladen werden) oder falls die Datei an ein PROM-Programmiergerät gesendet werden soll.

6.8.26 .CRL, CRL-Datei erzeugen

Es wird eine CRL-Datei erzeugt (für BDS C - Linker). Die Quelldatei muß dann aus Funktionen zusammengesetzt werden (s.u.). Zwischen .CRL und dem ersten FUNCTION darf kein Code erzeugt werden!

6.8.26.1 FUNCTION, Funktion definieren

FUNCTION name

Die Funktion mit dem Namen 'name' wird angelegt. Sie muß durch ENDFUNC abgeschlossen werden. Innerhalb einer Funktion können mit EXTERNAL externe Funktionen definiert werden. Vorsicht: Es wird nicht überprüft, ob externe Labels fälschlicherweise für Daten verwendet werden! Zwischen FUNCTION und ENDFUNC darf kein ORG verwendet werden.

6.8.26.2 ENDFUNC, Ende einer Funktion

Die zuvor angefangene Funktion wird beendet. Nach ENDFUNC darf kein Code mehr erzeugt werden. Danach darf nur FUNCTION oder END folgen, was aber nicht überprüft wird. ENDFUNC ist in der Funktion ähnlich zu MODEND, bitte dort nachlesen. Nach dem letzten ENDFUNC startet der Assembler einen weiteren (überflüssigen) Durchlauf, gibt entsprechende Meldungen aus (Anzahl der Fehler, END Statement) und führt das Listing weiter. Dies hat keine Auswirkung auf die erzeugte Datei. Das Listing kann bei Bedarf mit einem Texteditor korrigiert werden.

6.8.27 Hinweis

In einem Programm kann nur eine der obigen Direktiven zur Object-Dateiwahl vorkommen, da der Assembler nur eine Object-Datei erzeugen kann. Falls keine angegeben wurde, so wird die Einstellung durch die Switches übernommen. Wenn kein Switch angegeben wurde, so wird eine REL-Datei erzeugt.

7 Wiederholung von Statements

Statements bzw. ganze Zeilen lassen sich wiederholen, indem die Anzahl der gewünschten Wiederholungen gefolgt von einem Doppelpunkt vor das Statement geschrieben wird. Auf diese Weise lassen sich keine Macro-Definitionen, keine Direktiven zur bedingten Assemblierung, kein END, MODEND und kein .COMMENT- oder NOTICE-Block wiederholen. Alles, was hinter dem Wiederholungsfaktor steht, wird entsprechend oft wiederholt, einschließlich den Statements, die hinter einem eventuell vorhandenen Ausrufezeichen stehen. Hinter einem Ausrufezeichen kann ein weiterer Wiederholungsfaktor angegeben werden.

Beispiele:

```
        4:      ADD  A,A
        2:      SRL H!RR L
ADD4:   4:INC   HL

BSP:    2:ADD HL,HL!2:ADD HL,DE
```

Das letzte Beispiel ergibt folgenden Code:

```
BSP:    ADD HL,HL
        ADD HL,DE
        ADD HL,DE
        ADD HL,HL
        ADD HL,DE
        ADD HL,DE
```

8 Simulation von zusätzlichen Instruktionen

8.1 EX HL,DE

Statt EX DE,HL kann auch EX HL,DE geschrieben werden.

8.2 LD

Es kann ein Registerpaar direkt mit einem anderen geladen werden.

Statement	Ergibt
LD HL,DE	LD L,E ; usw. LD H,D
LD IX,BC	LD LX,C ; usw. LD HX,B
LD HL,IX	PUSH IX ; usw. POP HL
LD HL,SP	LD HL,0 ; auch mit IX oder IY ADD HL,SP

LD DE,SP	EX DE,HL
	LD HL,0
	ADD HL,SP
	EX DE,HL

LD BC,SP	PUSH HL
	LD HL,2
	ADD HL,SP
	LD C,L
	LD B,H
	POP HL

LD (nn),AF	PUSH HL PUSH AF POP HL LD (nn),HL POP HL

Außerdem:

8.3 EXTS

EXTS	LD L,A ; A mit Vorzeichen -> HL
	ADD A,A
	SBC A,A
	LD H,A

Tabelle 8-1: Erweiterte Z80-Instruktionen

8.4 PUSH, POP, INC und DEC

Bei PUSH, POP, INC und DEC können mehrere Register angegeben werden:

```
PUSH HL,DE,BC,AF        PUSH HL
                        PUSH DE
                        PUSH BC
                        PUSH AF

POP DE,IX               POP  DE
                        POP  IX

INC HL,DE,(IX+5),E      INC  HL
                        INC  DE
                        INC  (IX+5)
                        INC  E
```

Bei PUSH und POP kann auch (Adresse) angegeben werden. Dann wird der Inhalt der entsprechenden Speicherzellen (ein Wort) auf den Stack geschoben oder vom Stack geholt. Das Register HL wird als Zwischenspeicher verwendet.

```
PUSH (1234H),(QQ)       LD   HL,(1234H)
                        PUSH HL
                        LD   HL,(QQ)
                        PUSH HL

POP (QQ),(1234H)        POP  HL
                        LD   (QQ),HL
                        POP  HL
                        LD   (1234H),HL
```

8.5 OUT und Fortsetzung LD

Bei LD und OUT können weitere Register verwendet werden:

```
OUT (0),E                   LD   A,E
                            OUT  (0),A

OUT (0FFH),(ON)             LD   A,(ON)
                            OUT  (0FFH),A

LD (DE),(BC)                LD   A,(BC)
                            LD   (DE),A

LD (TEMP),LX                LD   A,LX
                            LD   (TEMP),A

LD B,(TEMP)                 LD   A,(TEMP)
                            LD   B,A

LD R,I                      LD   A,I
                            LD   R,A

LD E,(BC)                   LD   A,(BC)
                            LD   E,A
```

8.5.1 Hinweise, Ausnahmen

Natürlich sind auch andere Kombinationen möglich. Aber es gibt auch Kombinationen, die nicht möglich sind: Wenn LX, HX, LY oder HY (IXL, IXH, IYL oder IYH) zusammen mit H, L, (HL), (IX+) oder (IY+) verwendet wird.

Beispiele für derartige Fälle:

```
LD LX,H
LD (IX+5),HX
LD HY,(HL)
```

VORSICHT!!!!
Bei den bisher diskutierten simulierten Instruktionen kann <opcode> nicht benutzt werden!

8.6 Sprungbedingungen: OV und NV

Für Sprünge, RETs und CALLs können auch die Bedingungen OV (overflow) und NV (no overflow) verwendet werden. Sie werden durch PE bzw. PO ersetzt:

```
RET OV          ->   RET PE
CALL NV,HaHa    ->   CALL PO,HaHa
```

8.7 RIM und SIM

Falls .8080 oder der I-Switch angegeben wurde, kann auch RIM und SIM (8085-Instruktionen) assembliert werden, sonst ergeben RIM und SIM eine Fehlermeldung.

8.8 NEG mit anderen Registern

Bei NEG kann auch eines der folgenden Register angegeben werden: B, C, D, E, H, L, (HL), (IX+), (IY+), LX, HX, LY, HY, HL, DE, BC, IX, IY. Bei der Durchführung dieser Instruktionen wird das Register A verändert:

```
NEG  IXL          XOR  A
                  SUB  IXL
                  LD   IXL,A

NEG  HL           XOR  A
                  SUB  L
                  LD   L,A
                  SBC  A,H
                  SUB  L
                  LD   H,A
```

8.9 BIT, SET und RES mit Doppelregistern

Bei BIT, SET und RES können auch Doppelregister verwendet werden. Es wird dann die entsprechende Operation mit einem Teilregister durchgeführt:

```
BIT 15,HL    -->  BIT  7,H
```

9 Der Compiler: Konstruktionen zur strukturierten Programmierung

9.1 condition

Im folgenden wird öfters eine Bedingung, condition, verwendet. Hier n
die Definition (Optionales ist eingeklammert):

9.1.1 Definition

```
condition::=
term (OR condition)

term::=
factor (AND factor)

factor::=
< condition >
0000H                                   (%FALSE)
0FFFFH                                  (%TRUE)
NOT factor
NC C NZ Z PO PE OV NV P M
subfactor IN < list >
BIT n,register                          (n:0..7)
BIT n,registerpair                      (n:0..15)

list::=
listelement (,list)

listelement::=
constant (..constant)

subfactor::=
register
constant
registerpair

register::=
B C D E H L A LX HX LY HY (HL) (IX+) (IY+) (DE) (BC) R I

registerpair::=
HL DE BC IX IY

relation::=
<    <=      =       >       >=

constant::=
expression::=
Normaler Ausdruck, ohne AND, OR, LE, LT, GT und GE
```

9.1.2 Beschreibung und Beispiele

Da diese Definitionen nicht jedermanns Geschmack sind, folgen noch einige Hinweise und Beispiele, die mehr Klarheit verschaffen sollen:

Es kann nicht ein Register mit einem Registerpaar verglichen werden. Die Vergleiche werden immer vorzeichenlos durchgeführt. Es wird nur das Register A verändert. A wird nicht verändert, wenn es in einem Vergleich benutzt wird. (DE), (BC), R und I können nicht miteinander verglichen werden. Ebenso können diese Register nicht mit A verglichen werden. Im Ausdruck 'expression' darf kein AND und kein OR vorkommen. Statt <, <=, <>, =, > und >= ist LT, LE, NE, EQ, GT und GE zu verwenden. Bei Verwendung von <opcode> darf kein Label definiert werden. Bei register IN < list > wird überprüft, ob das Register einen der angegebenen Werte annimmt. Z.B. wird bei

```
A in <5,10,"A".."Z">
```

überprüft, ob A 5, 10 oder einen Wert zwischen "A" und "Z", je einschließlich, annimmt. Statt spitzen Klammern < > können auch eckige verwendet werden. Bei IN < list > können außerdem auch runde Klammer benutzt werden.
Die Reihenfolge der Abarbeitung ist folgende (bei gleichwertigen Operatoren wird von links nach rechts gearbeitet):

```
IN   <   <=   <>   =   >   >=
NOT
AND
OR
```

Tabelle 9-1: Reihenfolge der Operatoren beim Compiler

Hier zwei Statements, die nicht zur Strukturierung beitragen, aber Gebrauch von condition machen:

9.2 Das RETIF- und ENDWHEN-Statement

RETIF condition RETWHEN condition

Falls condition erfüllt ist, erfolgt ein RET.

Beispiele:

Diese Compiler-Statements	entsprechen diesen Assembler-Statements
RETIF C OR Z	RET C RET Z
RETIF A>5 OR HL=0	CP 6 RET NC LD A,L OR H RET Z

```
RETIF BC=0FF34H          LD   A,C
                         SUB  35H
                         AND  B
                         INC  A
                         RET  Z

RETIF HL<>7000H          LD   A,H
                         SUB  70H
                         OR   L
                         RET  NZ

RETIF NOT C IN ("A".."Z","a".."z")
                         LD   A,C
                         CP   "A"
                         JR   C,L1
                         CP   "Z"+1
                         JP   C,L2
                    L1:  CP   "a"
                         RET  C
                         CP   "z"+1
                         RET  NC
                    L2:

RETIF (DE) IN <0..1FH>   LD   A,(DE)
                         CP   20H
                         RET  C

RETWHEN A>E              CP   E
                         JR   Z,L3
                         RET  NC
                    L3:

RETWHEN TRUE             RET
```

9.3 Das JPIF- und JPWHEN-Statement

```
JPIF condition,destination
JPWHEN condition,destination
```

Falls 'condition' erfüllt ist, erfolgt ein Sprung nach 'destination'. Der Sprung ist immer ein absoluter.

Beispiele:

```
JPIF C,1234H             JP   C,1234H

JPWHEN A=5 OR A=6,LBL    CP   5
                         JP   Z,LBL
                         CP   6
                         JP   Z,LBL
```

9.4 Das DO-Statement

```
DO register,initial value
 instructions
ENDDO
```

Die Instruktionen zwischen DO und ENDDO werden so oft wiederholt, wie 'initial value' angibt. Das Register 'register' wird zum Zählen der Schleifendurchläufe verwendet. Die Angabe von 'initial value' ist optional. Wenn 'initial value' nicht angegeben wird, dann werden die Instruktionen so oft wiederholt, wie das spezifizierte Register zu Beginn der Schleife angibt. Für Werte von 'initital value', die größer als 255 sind, ist ein Registerpaar zu verwenden. Ist 'initial value' 0, dann wird die Schleife bei einem 8-bit-Register 256-mal, bei einem Registerpaar 65536-mal wiederholt. Statt DO kann auch COUNTDOWN, statt ENDDO ENDCD geschrieben werden. Falls möglich, wird JR bzw. DJNZ benutzt. Für 'initial value' können auch Register und Speicheradressen / Labels benutzt werden, nicht jedoch <opcode>.

Beispiele:

```
DO C,20                     LD   C,20
 LD  (HL),"-"         L1:   LD   (HL),"-"
 INC HL                     INC  HL
ENDDO                       DEC  C
                            JR   NZ,L1

DO DE,1000!ENDDO            LD   DE,1000
                     L2:    DEC  DE
                            LD   A,D
                            OR   E
                            JR   NZ,L2

countdown hl,(len)          LD   HL,(LEN)
 call xxx            L3:    CALL XXX
endcd                       DEC  HL
                            LD   A,H
                            OR   L
                            JR   NZ,L3

do b,10                     LD   B,10
enddo                       DJNZ $

DO B,10                     LD   B,10
DEFS 1000            L4:    DEFS 1000
ENDDO                       DEC  B
                            JP   NZ,L4
```

9.5 Das REPEAT-Statement

```
REPEAT
 instructions
UNTIL condition
```

Die Instruktionen zwischen REPEAT und UNTIL werden solange wiederholt, bis condition erfüllt ist. Die Überprüfung findet am Ende der Schleife statt.

Beispiele:

```
REPEAT
 CALL SUBR          L1:  CALL SUBR
UNTIL HL=0               LD   A,L
                         OR   H
                         JP   NZ,L1

repeat
 ld a,(hl)          L2:  LD   A,(HL)
 ld (de),a               LD   (DE),A
 inc hl,de               INC  HL
until a>10               INC  DE
                         CP   11
                         JP   C,L2

repeat
 ld c,6             L3:  LD   C,6
 ld e,255                LD   E,255
 call 5                  CALL 5
until a<>0               OR   A
                         JP   Z,L3

repeat
 call xxx
until a in ("0".."9","A".."Z")
                    L4:  CALL XXX
                         CP   "0"
                         JR   C,L5
                         CP   "9"+1
                         JP   C,L6
                    L5:  CP   "A"
                         JP   C,L4
                         CP   "Z"+1
                         JP   NC,L4
                    L6:

repeat
 call xxx
until a=2 or a=5 or a=100
                    L7:  CALL XXX
                         CP   2
                         JP   Z,L8
                         CP   5
                         JP   Z,L8
                         CP   100
                         JP   NZ,L7
                    L8:
```

```
repeat
 call xxx
until po and z and c
```

```
                    L9:  CALL XXX
                         JP   PE,L9
                         JP   NZ,L9
                         JP   NC,L9
```

```
REPEAT
   -- Endlosschleife --
UNTIL %FALSE        L10:  JP   L10
```

9.6 Das WHILE-Statement

```
WHILE condition
 instructions
ENDWHILE
```

Solange condition erfüllt ist, werden die Instruktionen zwischen WHILE und ENDWHILE wiederholt. Die Überprüfung findet am Anfang der Schleife statt.

Beispiele:

```
WHILE (HL)<>0          L1:  LD   A,(HL)
 LD (DE),(HL)               OR   A
 INC HL,DE                  JP   Z,L2
ENDWHILE                    LD   A,(HL)
                            LD   (DE),A
                            INC  HL
                            INC  DE
                            JP   L1
                       L2:

while HL<DE            L3:  LD   A,H
 add hl,bc                  CP   D
 ex  af,af'                 JR   NZ,L4
 inc a                      LD   A,L
 ex  af,af'                 CP   E
endwhile               L4:  JP   NC,L5
                            ADD  HL,BC
                            EX   AF,AF'
                            INC  A
                            EX   AF,AF'
                            JP   L3
                       L5:

WHILE A<>0             L6:  OR   A
 DEC A                      JP   Z,L7
ENDWHILE                    DEC  A
                            JP   L6
                       L7:

while z and c and p
 call xxx              L8:  JP   NZ,L9
endwhile                    JP   NC,L9
                            JP   M,L9
                            CALL XXX
                            JP   L8
                       L9:
```

9.7 Das LOOP-Statement

```
LOOP
 instructions      EXITIF condition      EXIT
ENDLOOP
```

Die Instruktionen zwischen LOOP und ENDLOOP werden solange wiederholt, bis bei einem EXITIF zwischen LOOP und ENDLOOP condition erfüllt ist, oder ein EXIT durchgeführt wird. Es können mehrere EXITIF vorkommen (das ist der Sinn der Sache). Statt ENDLOOP kann auch ENDL verwendet werden. Statt EXITIF kann EXITWHEN benutzt werden. Mit EXIT wird die Schleife ohne Bedingung abgebrochen. Je nach gewählter Option (siehe *OPT) erfolgt der Sprung aus der Schleife mit JP (Option LJ) oder mit RET (Option LR), wobei in letzterem Falle beachtet werden muß, daß die Adresse des Schleifenendes in den Stack geschoben wurde und somit der Stack innerhalb von LOOP nicht verändert werden sollte. Standardeinstellung ist die Beendigung mit JP. Wenn jedoch sehr viele Abfragen vorhanden sind, so ist es günstiger, die Option LR zu wählen.

Beispiele:

```
*OPT LR
LOOP
 LD A,(HL)
EXITIF A=0
 LD (DE),A
 INC DE
EXITIF DE>BC
 INC HL
ENDLOOP
```

```
        PUSH HL
        LD   HL,L3
        EX   (SP),HL
L1:     LD   A,(HL)
        OR   A
        RET  Z
        LD   (DE),A
        INC  DE
        LD   A,B
        CP   D
        JR   NZ,L2
        LD   A,C
        CP   E
L2:     RET  C
        INC  HL
        JR   L1 ; Dieser Sprung wird optimiert
L3:
```

Das Gleiche nochmal, aber...

```
*OPT LJ
LOOP
 LD A,(HL)
EXITIF A=0
 LD (DE),A
 INC DE
EXITIF DE>BC
 INC HL
ENDLOOP
```

```
L4:     LD   A,(HL)
        OR   A
        JP   Z,L6
        LD   (DE),A
        INC  DE
        LD   A,B
        CP   D
        JR   NZ,L5
        LD   A,C
        CP   E
L5:     JP   C,L6
        INC  HL
        JR   L4
L6:
```

9.8 Das WHEN-Statement

```
WHEN condition
 instructions
ELSEWHEN condition
 instructions
OTHERWISE
 instructions
ENDWHEN
```

Wenn condition (nach WHEN) erfüllt ist, werden die Instruktionen nach WHEN ausgeführt. Andernfalls wird condition beim nächsten ELSEWHEN überprüft. Der OTHERWISE-Teil wird ausgeführt, wenn keine der Bedingungen erfüllt wird. ELSEWHEN kann beliebig oft (auch keinmal) angegeben werden. OTHERWISE kann keinmal oder einmal auftreten, muß aber nach dem letzten ELSEWHEN stehen. Statt ELSEWHEN kann auch ELSEIF oder CASE geschrieben werden. Statt ENDWHEN ist auch ENDW erlaubt. Wurde .IF (s.o.) angegeben, so werden WHEN, OTHERWISE und ENDWHEN durch IF, ELSE und ENDIF ersetzt.

Beispiele:

```
.if
if b=0
 ld a,"0"
elseif b>c
 ld a,"X"
elseif hl=de
 ld a,"*"
else
 ld a," "
endif
```

```
        LD   A,B
        OR   A
        JP   NZ,L1
        LD   A,"0"
        JP   L4
L1:     LD   A,C
        CP   B
        JP   NC,L2
        LD   A,"X"
        JP   L4
L2:     LD   A,H
        CP   D
        JP   NZ,L3
        LD   A,L
        CP   E
        JP   NZ,L3
        LD   A,"*"
        JP   L4
L3:     LD   A," "
L4:
```

10 Listing

Jede Seite wird mit einer Überschrift versehen. In die Überschrift wird gegebenenfalls der mit TITLE definierte Text eingesetzt. Unter diese Überschrift kommt der bei SUBTTL angegebene Text. Die Seitenlänge läßt sich u.a. mit PAGE einstellen (s.o.). Der Standardwert für die Seitenlänge beträgt 50. Wenn eine neue Seite angefangen wird, wird die Überschrift neu ausgegeben. Wird im Quelltext ein CTRL-L angetroffen, so wird eine neue Seite angefangen. Die Kapitelnummer wird um Eins erhöht, die Seitennummer wird gelöscht. Das CTRL-L wird beim weiteren Assemblieren ignoriert. Wenn bei PAGE 0 angegeben wurde, so werden vom Assembler selbst keine Seitenumbrüche erzeugt.
Die Seitennummernangabe besteht aus der Kapitelnummer und der Seitennummer. Die beiden werden durch einen Bindestrich getrennt. Beim Anfangen einer neuen Seite wird die Seitennummer um Eins erhöht. Zu Beginn eines neuen Kapitels (CTRL-L, erste Seite) ist die Seitennummer gelöscht. Auf den Seiten mit der Symbol table wird die Kapitelnummer durch ein 'S' ersetzt.
Das Listing besteht aus dem Adreßfeld, dem Datenfeld, dem Zeilennummernfeld und dem Quelltextfeld. Alle Zahlenangaben werden mit der mit .LRADIX definierten Zahlenbasis ausgegeben (Standard: 16). Je nac' Modus wird eines der folgenden Zeichen an eine Zahl angehängt:

Absolut	
Relativ zum Programmsegment	'
Relativ zum Datensegment	"
COMMON	!
Extern	*
Global	I

Tabelle 10-1: Kennzeichnung der Modi im Listing

In der Symbol table wird bei globalen Labels zusätzlich ein I angegeben.

Bei einem Statement, das keinen Code erzeugt, bleibt das Adreß- und Datenfeld leer. Ausnahme: Beim EQU-Statement wird ins Adreßfeld der zugewiesene Wert und ins Datenfeld ein '=' zur Markierung eingesetzt.

Ins Zeilennummernfeld wird, falls vorhanden, die der Quelldatei entnommene Zeilennummer eingesetzt. Die gelesene Zeilennummer ersetzt die aktuelle Zeilennummer. Die Zeilennummern in der Quelldatei müssen mit gesetztem Bit 7 stehen. Wenn keine Zeilennummer in der Quelldatei steht und der L-Switch angegeben wurde, wird die aktuelle Zeilennummer erhöht und eingetragen. Ohne L-Switch bleibt das Zeilennummernfeld leer. Bei der Expansion eines Macros wird, falls der L-Switch gesetzt ist, statt der Zeilennummer ein '+' eingetragen.

Im Datenfeld erscheint der erzeugte Object-Code, wobei Worte als solche ausgegeben werden (d.h. in der Reihenfolge H, L). Bei einem EQU-Statement wird ein '=' eingetragen.

Weitere Hinweise siehe .LALL, .SALL und .XALL bzw. *MACLIST

A Zusammenstellung der Direktiven und Compiler-Statements

DEFB	DB			Define byte
DEFW	DW			Define word
DEFM	DM			Define message
DEFC	DC			Define character
DEFH	DH			Define message, Bit 7 setzen
FILL	BYTES			Mehrmaliges DEFB
WORDS				Mehrmaliges DEFW
DEFS	DS			Define space (FILL)
DEFL	DL	SET	ASET	Define label
EQU				Wert einem Label zuweisen
ORG				PC auf Adresse setzen
END				Ende des Programmes
MODEND				Ende des Moduls
ASEG				Absolutes Segment
CSEG				Code Segment
DSEG				Data Segment
COMMON				COMMON-Block
LOCAL				Lokale Labels deklarieren
GLOBAL	PUBLIC	ENTRY		Globale Labels deklarieren
EXTERNAL	EXTRN	EXT		Externals deklarieren
.REQUEST				Bibliotheken für Linker
NAME				Modulname
.SYMLEN				Länge der Labels einstellen
.EXTLEN				Dito, aber externals/globals
INCLUDE	MACLIB	*INCL		
*INCLUDE	$INCLUDE			Quelldatei einfügen
.CHAIN	LINK			Quelldatei wechseln
COMLIB				Binäre Daten aus Datei einfügen
SYMLIB				Labels aus Datei lesen
IF	IFT	COND		Bedingte Assemblierung: Wenn wahr
IFE	IFF			Wenn falsch
IF1				Wenn erster Durchlauf
IF2				Wenn zweiter Durchlauf
IFDEF				Wenn Label definiert
IFNDEF				Wenn Label nicht definiert
IFIDN				Wenn beide Argumente gleich
IFDIF				Wenn beide Argumente verschieden
IFB				Wenn Argument leer
IFNB				Wenn Argument nicht leer
REPT				Wiederholen
IRP				Dito, mit verschiedenen Parametern
IRPC				Dito, mit einzelnen Zeichen
MACRO				Macro definieren
EXITM				MACRO/REPT/IRP/IRPC verlassen
ELSE				Bedingte Assemblierung: Andernfalls
ENDIF	ENDC			Ende der bedingten Assemblierung
ENDM				Ende MACRO/REPT/IRP/IRPC
.RADIX				Zahlenbasis setzen
.RLD	.MZAL			RLD- und CMD-Datei erzeugen
.CMD				CMD-Datei erzeugen
.COM				COM-Datei erzeugen
.HEX				HEX-Datei erzeugen
.CRL				CRL-Datei erzeugen
.PHASE				Verschiebung: Anfang
.DEPHASE				Verschiebung: Ende
*RELTAB				Verschiebungstabelle
*DATA HERE				Daten von *Byte-Liste hier einfügen

.COMMENT			Kommentar bis Begrenzer
NOTICE			Kommentar in CMD-Datei schreiben
.EJECT	$EJECT	*EJECT	
PAGE			Seitenvorschub im Listing erzeugen
TITLE	*TITLE		Überschrift definieren
SUBTTL	$TITLE	*TITLE	Überschrift definieren
*SPACE			Leerzeile(n) einfügen
.LRADIX			Listing-Zahlenbasis
.LIST	*LIST ON		Listing an
.XLIST	*LIST OFF		Listing aus
.CREF			Keine Funktion
.XCREF			Keine Funktion
.LALL	*MACLIST ON		Macro-Listing: an
.SALL	*MAXLIST OFF		nur Object-Code
.XALL			Object, Source wenn Object-Code
*PAUSE			Halten, warten bis <RETURN>
.LFCOND			Bedingte Assemblierung: Listing an
.SFCOND			Listing aus
.TFCOND			An/aus umschalten
.Z80			Z80-Programm
.8080			8080/8085-Programm
.PRINTX			Text ausgeben (mit Delimiter)
.PRINTL			Text ausgeben (bis Zeilenende)
.PRINTN			Zahl ausgeben (siehe .LRADIX)
.SLI			Special-LINK-Item erzeugen
.AG			Alle Labels global
.IF			IF <-> WHEN
EXTS			A -> HL (mit Vorzeichen)
DO	COUNTDOWN		Wiederholungsschleife
ENDDO	ENDCD		Ende derselben
REPEAT			Wiederholen
UNTIL			bis...
RETWHEN	RETIF		RET, wenn...
LOOP			Schleife, bis EXITIF erfüllt
EXITIF	EXITWHEN		Bedingter Schleifenabbruch
EXIT			Unbedingter Schleifenabbruch
ENDLOOP	ENDL		Schleifenende
WHILE			Wiederholung solange...
ENDWHILE			Ende Wiederholung
WHEN			Verzweigung
OTHERWISE			Verzweigung: Andernfalls
ELSEIF	ELSEWHEN	CASE	Verzweigung: Fallunterscheidung
ENDWHEN	ENDW		Ende der Verzweigung
FUNCTION			Anfang einer Funktion (CRL-Datei)
ENDFUNC			Ende einer Funktion (CRL-Datei)
*OPT			Optionen einstellen
*ERROR			Fehlermeldung ausgeben ein-/ausschalt

B Sonderzeichen im Quelltext und ihre Bedeutungen

! Trennung von Statements in einer Zeile
Ermöglicht die Verwendung von *Direktive nach 'Label:'
Bewirkt die direkte Übernahme eines Zeichens als Macro-Parameter

" Beginnt und beendet einen Text
Am Ende einer Zahl: Relativ zum Datensegment
"" Erzeugt ein einzelnes " innerhalb eines mit " begrenzten Textes

Ist ein in Labels erlaubtes Zeichen
Markiert ein Label als extern

$ Variable, enthält den aktuellen PC-Wert
Ist ein in Labels erlaubtes Zeichen
Leitet bestimmte Direktiven ein
Ist zwischen den Ziffern einer Zahl erlaubt

% Ist ein in Labels erlaubtes Zeichen (nicht am Anfang)
Dient zur Unterscheidung von HIGH, NUL u.a. von Labels
Bewirkt das Ausrechnen von Macro-Parametern

& Dient zur Zusammensetzung von Labels durch Macro-Parameter
Übernahme von Macro-Parameter und lokalen Labels in Texte

' Beginnt und beendet einen Text
X'nnnn' interpretiert nnnn als Hexadezimalzahl
Am Ende einer Zahl: Relativ zum Programmsegment
Bezeichnet Zweitregister bei EX AF,AF'
'' Erzeugt ein einzelnes ' innerhalb eines mit ' begrenzten Textes

( ) Zeigt indirekte Adressierung an: (DE), (1234H)
Bewirkt Änderung der Berechnungsreihenfolge in Ausdrücken
Markiert *RELTAB-Nummer, falls nicht mit Ziffer beginnend
Begrenzt Liste bei IN
Begrenzt Parameter von z.B. NAME

* Am Zeilenanfang oder nach !: Direktive oder Kommentar
Multiplikation in Ausdrücken (nach Operand)
*Byte-Liste (am Anfang von Ausdrücken, nach Operatoren)
Bezeichnet beliebiges Ende bei Masken von SYMLIB

+ Addition in Ausdrücken
Angabe eines Displacements bei (IX+)/(IY+)

, Trennung von Parametern/Operanden von Statements
Trennung von Listen-Elementen (IN)

- Subtraktion in Ausdrücken
Vorzeichenänderung in Ausdrücken
Angabe eines Displacements bei (IX-)/(IY-)
-- Markiert Kommentar

. Ist ein in Labels erlaubtes Zeichen
Leitet bestimmte Direktiven ein
.. Bezeichnet Bereich bei IN
Markiert Bereich bei *RELTAB HERE

/ Division in Ausdrücken
Begrenzung des COMMON-Namens

: Begrenzt Label-Definition
Bewirkt Zeilenwiederholung (5:ADD HL,HL)
:: Begrenzt globale Label-Definition

; Leitet Kommentar ein
;; Leitet Kommentar ein, der nicht im Macro gespeichert wird

< Bedeutet Vergleich auf kleiner in Ausdrücken/condition
Bewirkt Übergabe einer Liste als Macro-Parameter
Begrenzt Parameter von IRP/IRPC/IFB/IFNB/IFDIF/IFIDN
Markiert <opcode>-Argument
Leitet Liste bei IN ein
Bewirkt Änderung der Berechnungsreihenfolge bei condition (Anfang)
<= Bedeutet Vergleich auf kleiner/gleich in Ausdrücken/condition
<> Bedeutet Vergleich auf ungleich in Ausdrücken
Leerer Parameter bei IRP/IRPC

= Bedeutet Vergleich auf gleich in Ausdrücken/condition

> Bedeutet Vergleich auf größer in Ausdrücken/condition
Beendet Listen-Übergabe als Macro-Parameter
Begrenzt Parameter von IRP/IRPC/IFB/IFNB/IFDIF/IFIDN
Beendet <opcode>-Argument
Beendet Liste bei IN
Bewirkt Änderung der Berechnungsreihenfolge bei condition (Ende)
>= Bedeutet Vergleich auf größer/gleich in Ausdrücken/condition

? Ist ein in Labels zulässiges Zeichen

@ Ist ein in Labels zulässiges Zeichen

Eckige Klammern:
Bewirken Änderung der Berechnungsreihenfolge bei condition
Begrenzen Liste bei IN

_ Ist ein in Labels zulässiges Zeichen

C Die Fehlermeldungen

*DATA HERE missing
Es wurde *Byte-Liste benutzt, ohne daß eine REL-Datei erzeugt oder *DATA HERE angegeben wurde, oder es wurde noch nach dem letzten *DATA HERE ein *Byte-Liste-Operand verwendet.

.DEPHASE without .PHASE
Zu diesem .DEPHASE fehlt das zugehörige .PHASE.

8085 instruction not allowed in Z80 mode
RIM und SIM sind nur im 8080-Modus erlaubt.

<...> not allowed here
An dieser Stelle darf kein <opcode>-Argument verwendet werden.

Bad number
Es ist ein Fehler in einer Zahl, z.B. ergeben die verwendeten Ziffern einen Konflikt mit dem nachgestellten Kennzeichen für die Zahlenbasis oder der mit .RADIX eingestellten Zahlenbasis. Falsches Zeichen innerhalb von X' '.

Bad operand
Diese Fehlermeldung erscheint in den Fällen, in denen ein anderes Argument als das angegebene erwartet wurde, z.B. ein Register statt einer Zahl z.B. bei ORG.

Branch out of range
Die Distanz ist zu weit für einen relativen Sprung. Der relative Sprung ist durch einen absoluten zu ersetzen.

Constant expected
An dieser Stelle sollte eine Konstante stehen.

Division by zero
Es wurde ein Argument durch 0 dividiert.

ENDDO expected
Ein DO/COUNTDOWN-Statement wurde nicht richtig abgeschlossen.

ENDDO/ENDLOOP/ENDWHILE/UNTIL surplus
Es wurde eines dieser Statements zuviel angegeben.

ENDIF/ELSE/ENDM without IF/REPT/IRP/IRPC/MACRO
Es wurde ein ENDC, ENDIF, ELSE oder ENDM zuviel angegeben.

ENDLOOP expected
Ein LOOP-Statement wurde nicht richtig abgeschlossen.

ENDWHEN/ENDIF expected
Ein WHEN/IF-Statement wurde nicht richtig abgeschlossen, eventuell wurde die Vertauschung der Mnemonics durch .IF nicht konsequent beachtet oder IF mit WHEN verwechselt.

ENDWHILE expected
Ein WHILE-Statement wurde nicht richtig abgeschlossen. ENDW entspricht nicht ENDWHILE !

Error in <...>
Innerhalb von <opcode> ist ein Fehler. Z.B. wurden <opcode>-Operanden zu tief geschachtelt oder es wird eine Direktive innerhalb eines solchen Operanden verwendet.

EXITM/LOCAL outside REPT/IRP/IRPC/MACRO
EXITM und LOCAL dürfen nur innerhalb eines REPT-, IRP-, IRPC- oder Macro-Blöcken verwendet werden.

Field overflow
Das Argument ist zu groß, es ist nicht mit 8 Bits darstellbar. Mit 8 Bits sind Zahlen im Bereich von -128 bis 255 darstellbar. Argumente, die nicht vom absolut sind, können ebenfalls nicht mit 8 Bits dargestellt werden.

IF/REPT/IRP/IRPC/MACRO unterminated
Es fehlt ein ENDM oder ENDIF. Oder END wurde zu früh angetroffen (M oder IF nach END vergessen).

Illegal addressing mode
Diese Adressierungsart ist nicht erlaubt (z.B. LD LX,HY).

Illegal conditional expression
In einem Compiler-Statement mit Bedingung ist ein Fehler vorhanden, z.B. sind die Register ausgegangen: WHEN (2)=(3). Siehe Definition von condition.

Illegal construction
Die Verwendete Konstruktion sollte aus einzelnen Z80-Instruktionen zusammengesetzt werden. Bei erweiterten Z80-Instruktionen kann <opcode> nicht verwendet werden!

Internal error
Bei Verwendung eines Compiler-Statements fehlt der erste Durchlauf oder die Unterschiede zwischen den Durchläufen sind zu groß (IF1 und IF2) oder es liegt wirklich ein Fehler im Assembler vor.

Invalid operation on external label
Es wurde eine andere Operation als die Addition auf ein externes Label angewandt.

Invalid use of relocatable labels
Hier wurden relozierbare Labels falsch verwendet, z. B. multipliziert. Möglicherweise mehrfach-relozierbares Ergebnis.

Label expected
Hier erwartet der Assembler die Angabe eines Labels (z.B. Maske bei *SYMLIB).

Line too long
Eine Zeile ist zu lang (d.h. länger als 128 Zeichen). Der Rest wurde abgeschnitten.

LINK mutually exclusive with MODEND
Es kann in einer Quelldatei entweder LINK oder MODEND benutzt werden, aber nicht beide. Weitere Hinweise siehe MODEND.

Lower bound exceeds upper bound
Die untere Grenze des Intervalls bei IN ist größer als die obere.

Missing ')'
Es fehlt eine geschlossene Klammer, oder es wurde eine Klammer zuviel geöffnet.

Missing '>'
Es fehlt ein '>' oder es wurde ein '<' zu viel angegeben (man beachte die verschiedenen Bedeutungen der spitzen Klammern, wie z.B. Vergleichsoperationen, Macro-Parameter und <opcode>-Opernanden).

Missing operand
Es folgt ein falscher oder kein Operand nach einem Operator oder einer Instruktion.

Multiple definition
Ein Label wird unerlaubterweise mehrmals definiert. Label-Länge überprüfen, oder falls Absicht: DEFL verwenden.

Nested .PHASE not allowed
Innerhalb eines PHASE-Blockes darf kein .PHASE erfolgen. Möglicherweise wurde ein .DEPHASE vergessen oder durch .PHASE ersetzt.

Non-8080 instruction
Diese Instruktion kann von einem 8080-Prozessor nicht durchgeführt werden.

Not allowed after *RELTAB HERE
Nach *RELTAB HERE dürfen keine Labels mehr definiert und $ nicht mehr benutzt werden.

Not allowed in current mode
Es wurde ein Feature benutzt, das nicht beim derzeitigen Object-Dateityp oder Segment oder bei Verwendung von *RELTAB und .PHASE vorkommen darf.

Not allowed within .PHASE/.DEPHASE
Innerhalb eines PHASE-Blockes wurde ein Statement benutzt, das nicht in einem solchen Block stehen darf (z.B. CSEG, ORG).

Number expected
An dieser Stelle kann nur eine Zahl stehen.

Number overflow
Eine Zahl ist zu groß, um mit 16 Bits darstellbar zu sein.

Option selected too late
.COM, .HEX, .CMD, .RLD, .MZAL, oder .CRL wurde angegeben, nachdem schon Code erzeugt wurde.

Overflow
Wie 'Number overflow', tritt aber auch bei internem Überlauf auf, d.h. wenn bei assembler-internen Operationen ein Überlauf entsteht.

Questionable: Illegal termination of line or addressing mode
Nach einem Statement stehen noch Zeichen, die keinen Sinn ergeben. Diese Fehlermeldung erfolgt oft zusätzlich zu einer anderen Fehlermeldung. Wenn am Ende eines Statements kein Semikolon oder Ausrufezeichen steht und das Ende der Zeile noch nicht erreicht ist, wird diese Fehlermeldung ausgegeben.

Segment Error
Konflikt zwischen zwei verschiedenen Modi, z.B. ORG 5" nach CSEG.

Syntax error
Der Assembler erwartet ein spezielles Zeichen und hat es nicht gefunden. Diese Fehlermeldung wird häufig ausgegeben, wenn der Fehler nicht näher beschrieben werden kann.

Too many functions
Bei der Erzeugung einer CRL-Datei wurde das Funktions-Directory voll.

Too many local labels
Ein Macro darf nur 64 lokale Labels haben.

Unbalanced brackets
Es fehlt eine schließende eckige (oder spitze) Klammer: Mehrfachbedeutung beachten!

Undefined
Ein Label ist nicht definiert oder falsch geschrieben worden oder es wurde kein oder kein gültiges Argument benutzt (z.B. DEFB ,,). Unbekannte Instruktion.

UNTIL expected
Zu einem REPEAT fehlt das zugehörige UNTIL.

Value different in 2nd pass
Ein Label erhält im zweiten Durchlauf einen anderen Wert als im ersten Durchlauf oder wird mehrfach definiert (DEFL verwenden, falls erwünscht).

Value error
Der angegebene Wert liegt außerhalb des erlaubten Bereichs.

D Die Fehlernummern (für *ERROR)

```
 1: Division by zero
 2: Number overflow
 3: Overflow
 4: Bad number
 5: Missing ')'
 6: Missing operand
 7: Syntax error
 8: Field overflow
 9: Illegal addressing mode
10: Undefined
11: Multiple definition
12: Bad operand
13: Value error
14: Missing '>'
15: Number expected
16: Questionable: Termination of line, addressing mode
17: Branch out of range
18: <...> not allowed here
19: Error in <...>
20: Value different in 2nd pass
21: Option selected too late
22: 8085 instruction not allowed in Z80 mode
23: Invalid use of relocatable labels
24: Invalid operation on external label
25: Label expected
26: Not allowed in current mode
27: LINK mutually exclusive with MODEND
28: Non-8080 instruction
29: Nested .PHASE not allowed
30: Not allowed within .PHASE/.DEPHASE
31: .DEPHASE without .PHASE
32: Segment error
33: IF/REPT/IRP/IRPC/MACRO unterminated
34: ENDIF/ELSE/ENDM without IF/REPT/IRP/IRPC/MACRO
35: EXITM/LOCAL/EXIT outside REPT/IRP/IRPC/MACRO/LOOP
36: Too many local labels
37: Illegal construction
38: ENDDO expected
39: UNTIL expected
40: Illegal conditional expression
41: Internal error
42: ENDLOOP expected
43: ENDWHILE expected
44: ENDWHEN/ENDIF expected
45: Unbalanced brackets
46: Lower bound exceeds upper bound
47: Constant expected
48: ENDDO/ENDLOOP/ENDWHEN/UNTIL surplus
49: *DATA HERE missing
50: Not allowed after *RELTAB HERE
51: Line too long
52: Too many functions
```

E Beispiele

E.1 Bibliothek für Zahlenausgabe

```
; Unterprogrammbibliothek zur Ausgabe von Zahlen
;
;
; Hexdezimale Ausgabe
;
NAME    ('HEXOUT')
GLOBAL  HEXWORD,HEXBYTE,HEXNIB
;
; HL ausgeben
HEXWORD:LD      A,H
        CALL    HEXBYTE
        LD      A,L
;
; A ausgeben
HEXBYTE:PUSH    AF
        4:RRCA                      -- Bits 4..7 ausgeben
          CALL    HEXNIB
          POP     AF                -- Bits 0..3 ausgeben
;
; Nibble ausgeben (Bits 0..3 von A)
HEXNIB:   AND     0FH               -- Nur Bits 0..3
          ADD     A,90H             -- Umwandlung in ASCII
          DAA
          ADC     A,40H
          DAA
          LD      E,A
          JP      OUTPUT##

          MODEND
------------------------ Fortsetzung folgt ---------
```

```
;
;
; Binäre Ausgabe
;
          NAME ('BINOUT')
          GLOBAL  A$BIN,HL$BIN
;
; HL ausgeben
HL$BIN:   LD      A,H
          CALL    A$BIN
          LD      A,L
;
; A ausgeben
A$BIN:    DO      B,8               -- 8 Bits
           RLA                      -- MSB -> Carry
           LD      E,18H            -- "0" SHR 1
           RL      E                --> "0" oder "1"
           CALL    OUTPUT##
          ENDDO
          RET

          MODEND
------------------------ Fortsetzung folgt ---------
```

```
;
;
; Dezimale Ausgabe von A
;
          NAME    ('DECOA')
          GLOBAL  ADECU,ADECS
          EXT     OUTPUT
;
; A mit Vorzeichen ausgeben
ADECS:    OR      A
          JP      P,ADECU
          LD      E,"-"             -- A negativ
          CALL    OUTPUT
          NEG
;
; A ohne Vorzeichen ausgeben
ADECU:    LD      BC,100
          CALL    ADECU1
          LD      C,10
          CALL    ADECU1
          ADD     A,"0"
          LD      E,A
          JP      OUTPUT
ADECU1:   LD      E,"0"-1
          REPEAT
           INC    E
           SUB    C
          UNTIL   C
          ADD     A,C
          PUSH    AF
          WHEN    E<>"0" OR B<>0
           INC    B
           CALL   OUTPUT
          ENDWHEN
          POP     AF
          RET

          MODEND
------------------------ Fortsetzung folgt -------
```

```
;
;
; Dezimale Ausgabe von HL
;
          NAME    ('DECOHL')
          GLOBAL  HLDECS,HLDECU
          EXT     OUTPUT
;
; HL mit Vorzeichen ausgeben
HLDECS:   BIT     7,H
          JR      Z,HLDECU
          LD      E,"-"             -- HL negativ
          CALL    OUTPUT
          NEG     HL
;
; HL ohne Vorzeichen ausgeben
HLDECU:   LD      D,0
          LD      BC,-10000
          CALL    HLDEC1
          LD      BC,-1000
          CALL    HLDEC1
          LD      BC,-100
          CALL    HLDEC1
          LD      BC,-10
          CALL    HLDEC1
          LD      A,L
          ADD     A,"0"
          LD      E,A
          JP      OUTPUT
HLDEC1:   LD      E,"0"-1
          REPEAT
           INC    E
           ADD    HL,BC
          UNTIL   NC
          SBC     HL,BC
          WHEN    DE<>"0"           -- D<>0 OR E<>"0"
          INC     D
          CALL    OUTPUT
          ENDWHEN
          RET

          MODEND
------------------------ Fortsetzung folgt -------
```

```
;
;
; Ausgaberoutine
;
          NAME    ('OUTPUT')
          GLOBAL  OUTPUT
;
TRS80     EQU     1         ; TRS-80, Bildschirmausgabe
CPMCON    EQU     2         ; CP/M, CON:
CPMLST    EQU     3         ; CP/M, LST:
MEMORY    EQU     4         ; -> (IX)
-------------------------------------------------------------
TYPE      EQU     CPMCON              --> Ausgabe auf CON:
-------------------------------------------------------------
OUTPUT:   PUSH    AF,DE,BC,HL
          IF      TYPE = TRS80
          LD      A,E
          CALL    33H
          ELSE
          IF      TYPE = CPMCON
          LD      C,2
          CALL    5
          ELSE
          IF      TYPE = CPMLST
          LD      C,5
          CALL    5
          ELSE
          IF      TYPE = MEMORY
          LD      (IX),E
          INC     IX
          ELSE
          .PRINTL - Modul OUTPUT -
          .PRINTL Unerlaubte Option
          ENDC
          ENDC
          ENDC
          ENDC
          POP     HL,BC,DE,AF
          RET
          END
-------------------------------------------------------------
```

E.2 Schnelle 16x16-bit-Multiplikation ohne Schleife

Beispiel für REPT mit LOCAL

```
; Multiplikation
; HL := BC * DE    (mit Vorzeichen)
;
; Diese Routine berechnet das Produkt sehr schnell, hat aber
; den Nachteil, daß sie recht länglich ist (REPT 15).
;
;
MUL:        LD      HL,0
            REPT    15              -- 15 Bits (Bit 15: siehe unten)
            LOCAL   NO$ADD
            ADD     HL,HL           -- HL SHL 1
            EX      DE,HL
            ADD     HL,HL           -- DE SHL 1
            EX      DE,HL
            JR      NC,NO$ADD
            ADD     HL,BC
NO$ADD:
            ENDM
; Sonderbehandlung von Bit 15 zur Einsparung eines Sprunges
            ADD     HL,HL           -- HL SHL 1
            EX      DE,HL
            ADD     HL,HL           -- DE SHL 1
            EX      DE,HL
            RET     NC              -- Einsparung des Sprunges
            ADD     HL,BC
            RET
```

F Unterschiede zu MS-MACRO (Microsoft MACRO Assembler)

F.1 Allgemeine Unterschiede

- Beim Aufruf mit =DATEI wird weder Listing noch Object-Code erzeugt.
- Der Aufruf mit Laufwerksbezeichnungen ohne Dateinamen ist nicht zulässig.
- Die maximale Zeilenlänge beträgt 128 Zeichen.
- Es gibt (noch) keine BYTE EXTERNALs und es dürfen noch keine Operationen außer Addition und Subtraktion auf relozierbare Argumente angewandt werden.
- PAGE u.a. erzeugen keinen Seitenvorschub wenn damit die Seitenlänge eingestellt wird.
- Es wird kein Seitenvorschub am Anfang des Listings erzeugt.
- 8080-Mnemonics werden nicht unterstützt.
- Bei DEFW wird im Listing pro Zeile nur ein Wert ausgegeben.
- .CREF und .XCREF nicht implementiert (Cross-Reference wird in einer der nächsten Versionen vom Assembler selbsttätig erstellt).
- Bei der Ersetzung von Dummy-Parametern bei der Macro-Expansion werden '&' entfernt.
- Nach .SALL wird nur der Object-Code gelistet, MS-MACRO listet hingegen überhaupt keine Macro-Expansion mehr.

F.2 Erweiterungen gegenüber MS-MACRO

Hier werden nur kleinere Erweiterungen aufgeführt, die beim Lesen des Handbuchs leicht übersehen werden können.

- Bei .RADIX können Werte zwischen 2 und 36 angegeben werden.
- Die Parameter-Ersetzung durch % geht bei allen Zahlenbasen.
- Es können mehrere Instruktionen in einer Zeile stehen.
- REPT,IRP,IRPC können lokale Labels haben.
- IRP kann mehrere Variable haben.
- EQU wird im Listing mit = markiert.
- Geschachteltes INCLUDE.
- Weitere Zeichen in Labels erlaubt: % #
- Mehrere Assemblierungen in Kommandozeile angebbar.
- Überprüfung auf 8080-Kompatibilität (bis auf OV/NV).
- Stop bei Fehler möglich.
- Mehrere Labels können in einer Zeile definiert werden.
- Zwischen Label und : können beliebig viele Leerzeichen stehen.
- Ausgabe von Zahlen während Assemblierung möglich.
- Einsparung von zwei Seiten Papier wenn Seitenlänge eingestellt wird (PAGE n).

G Hinweise

G.1 Markenzeichen

BDS C	Leor Zolman, BD Software
CP/M	Digital Research, Inc.
M-ZAL	Computer Applications Unlimited
Microsoft MACRO Assembler	Microsoft Corporation
MS	Microsoft Corporation
MS-LINK, Link-80	Microsoft Corporation
MS-MACRO, Macro-80	Microsoft Corporation
NEWDOS/80	APPARAT, Inc
SCRIPSIT	Tandy Radio Shack, Inc.
TRS-80	Tandy Radio Shack, Inc.
WordStar	MicroPro International, Inc.
Z80	Zilog, Inc.

H Stichwortverzeichnis