SOFTWARE

WEGA

WEGA-Cross-Software

EAW *electronic*

P8000

Version 1.2 (2008-03-30)

EAW-electronic P8000

W E G A - S o f t w a r e

Cross-Software

Nutzerhandbuch

Diese Dokumentation wurde von einem Kollektiv des Kombinates

VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"

erarbeitet.

Herausgeber:
Kombinat
VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Hoffmannstrasse 15-26
BERLIN
1193

Verantwortlicher Bearbeiter: J. Zabel

WAE/03-0302-01

Ausgabe: 5/87

Die vorliegende Dokumentation unterliegt nicht dem Aenderungsdienst.

Spezielle Hinweise zum aktuellen Stand der Softwarepakete befinden sich in README-Dateien auf den entsprechenden Vertriebsdisketten.

Dieser Band enthaelt folgende Unterlagen:

- Teil 1: WEGA
  U881 PLZ/ASM Cross-Assembler
  Nutzerhandbuch

- Teil 2: WEGA
  U880 Cross-Assembler
  Nutzerhandbuch

- Teil 3: WEGA
  U880 C-Cross-Compiler
  Nutzerhandbuch

- Teil 4: WEGA
  U880 Turbo-Cross-Assembler
  Nutzerhandbuch

- Teil 5: WEGA
  K1810WM86 Cross-Assembler
  Nutzerhandbuch

- Teil 6: WEGA
  LOAD/SEND-Kommunikation
  Nutzerhandbuch

- Teil 7: WEGA
  Prom-Programmer
  Nutzerhandbuch

Gesamtinhaltsverzeichnis Seite

Hinweise des Lesers zu diesem Dokumentationsband

Wir sind staendig bemueht, unsere Unterlagen auf einem qualitativ hochwertigen Stand zu halten. Sollten Sie deshalb Hinweise zur Verbesserung dieses Dokumentationsbandes bzw. zur Beseitigung von Fehlern haben, so bitten wir Sie, diesen Fragebogen auszufuellen und an uns zurueckzusenden.

Titel des Dokumentationsbandes:

Ihr Name / Tel.-Nr.:

Name und Anschrift des Betriebes:

Genuegt diese Dokumentation Ihren Anspreuchen? ja / nein
Falls nein, warum nicht?

Was wuerde diese Dokumentation verbessern?

Sonstige Hinweise:

Fehler innerhalb dieser Dokumentation:

Unsere Anschrift: Kombinat VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Abteilung Basissoftware
Hoffmannstrasse 15-26
BERLIN
1193

Teil 1

W E G A

U881 PLZ/ASM Cross-Assembler

Nutzerhandbuch

Inhaltsverzeichnis Seite

1. Einfuehrung

1.1. Allgemeine Beschreibung

Der U881 PLZ/ASM Assembler (aufgerufen durch das WEGA-Kommando "u8as") ist der Cross-Assembler fuer den Mikrocomputer U881 im WEGA. Er verarbeitet eine Quelldatei (eine symbolische Darstellung eines Programms in U881 Assembler) und uebersetzt sie in einen Objektmodul. Er kann auch eine Listingdatei erzeugen, die die Quelle und den assemblierten Kode enthaelt.
In diesem Handbuch ist nicht die PLZ/ASM Assemblersprache beschrieben. Hinweise dazu sind im Band "UDOS-Software, Mikroprozessorsoftware" der P8000-Dokumentation, Abschnitt "U8000/U881 PLZ/ASM" zu finden.

1.2. Verschieblichkeit

Verschieblich bedeutet, dass ein Programmmodul und seine Daten nach der Assemblierung in einen bestimmten Speicherbereich gebunden werden koennen. Der Assemblerausgang ist ein Objektmodul mit hinreichenden Informationen, damit ein Linker oder Lader diesem Modul einen Speicherbereich zuweisen kann (s. auch Beschreibung des WEGA Linker/Lader "ld(1)" im Band "WEGA-Programmierhandbuch" der P8000-Dokumentation).

1.3. Abbruchbedingungen fuer den Assembler

Fuer den Assembler existieren zwei Abbruchbedingungen:

1. Wenn waehrend eines Systemaufrufs ein Fehler auftritt, wird die Assemblierung abgebrochen und eine Fehlermeldung auf dem Bildschirm ausgegeben.

2. Wenn der Assembler infolge eine Fehlers abstuerzt, wird die Assemblierung abgebrochen und ein Assemblerabruchfehler 255 auf dem Bildschirm angezeigt.

2. Eingabe/Ausgabe

2.1. Eingabe

Durch einen Editor wird ein U881 PLZ/ASM Quellprogramm erzeugt. Der Name der Quelldatei muss mit .s enden (Gross- oder Kleinbuchstabe). Befehle fuer den Auruf des Assemblers sind im Abschnitt 3 angegeben.

2.2. Ausgabe

Der Assembler erzeugt eine Listingdatei. Standardmaessig erhaelt sie den Namen der Quelldatei dessen Endung .s, durch .l ersetzt wird . Die Listingdatei beinhaltet die Quellanweisungen und die entsprechenden Zeilennummern. Die Nummer der Fehlermeldung befindet sich in der nachfolgenden

Zeile, in der der Fehler auftritt (s. Abschnitt 8).
Der Assembler erzeugt ausserdem eine Objektdatei mit der Standardbezeichnung "a.out" (s. "a.out(5)"). Zur Erzeugung dieser Datei benutzt der Assembler eine temporäere Zwischendatei, die nach abgeschlossener Assemblierung geloescht wird.

3. Aufruf des Assemblers

3.1. Kommandozeile

Der Assembler wird durch folgende Kommandozeile aufgerufen:

```
u8as filname.s [options]
```

Der Dateiname muss die Endung .s haben. Die Endung gibt an, dass die Datei die Quelle fuer einen U881 PLZ/ASM Modul enthaelt.

3.2. Optionen

Folgende Optionen koennen in beliebiger Reihenfolge, getrennt durch ein Leerzeichen oder Tabulator, verwendet werden:

-d string	In Verbindung mit der Option -l wird das Datum in der Kopfzeile des Listing angeordnet (Zeichenkette max. 19 Zeichen).
-i	Die vom Assembler benutzte Zwischendatei wird gespeichert und erhaelt den Namen der Quelldatei ergaenzt durch die Endung .i anstelle von .s.
-l	Erstellt eine Listingdatei, die den Namen der Quelldatei, ergaenzt durch die Endung .l anstelle von .s, erhaelt. Wird die Option nicht verwendet, wird kein Listing erzeugt.
-o objfile	Die Objektdatei erhaelt den Namen "objfile" anstelle von "a.out".
-p	Das Listing wird zur Standardausgabe ausgegeben. Wird die Option nicht verwendet, werden nur die Quellzeilen ausgegeben, in denen Fehler enthalten sind.
-r	Die Datei mit den Verschieblichkeitsinformationen wird abgespeichert und erhaelt den Namen der Quelldatei ergaenzt durch die Endung .r anstelle von .s.
-u	Alle nicht definierten Symbole werden als extern betrachtet.

-v Einschalten der Bildschirminformation (Name und Versionsnummer, Meldung Durchlauf 1 und Assembler komplett).

-^ Start des Assemblerdurchlaufs 1

4. Listing-Format

4.1. Formatbeschreibung

Der Assembler erzeugt ein Listing des Quellprogramms entsprechend dem erzeugten Objektkode. In diesem Abschnitt werden die einzelnen Felder im Listing-Format beschrieben.

HEADING Der Kopf der ersten Seite enthaelt die Assembler-Versionsnummer und die Spaltennummer. Zusaetzlich kann der Kopf anwenderspezifische Zeichenketten, wie z.B. das Datum der Assemblierung, enthalten (s. Option Datum, Abschnitt 3.2.).

LOC Diese Spalte enthaelt einen Speicherbereichsbezeichner und einen Referenzzaehler. Der Bereichsbezeichner lautet P fuer Programm, R fuer Register und D fuer Daten. Der Zaehler startet fuer jeden Bereich bei Null.

OBJ CODE Diese Spalte enthaelt den erzeugten Objektkode. Sie ist leer, wenn eine Anweisung keinen Code erzeugt.
Jedes Byte oder Wort wird entweder von einem Apostroph ('), einem Stern (*) oder Leerzeichen gefolgt. Ein Apostroph bezeichnet eine relativen Wert. Ein Stern bedeutet, dass der Wert von einem externen Symbol abhaengt. Ein Leerzeichen bedeutet, dass der Wert nicht mehr veraendert wird. Ein Wert, der relativ oder extern abhaengig ist, kann leicht durch den Linker oder Lader veraendert werden. Der Listingwert kann sich vom Wert waehrend der Programmabarbeitung unterscheiden. Drei Punkte (...) bedeuten, dass das vorhergehende Byte, Wort oder Langwort wiederholt wird (nur bei Dateninitialisierung).

STMT Die Spalte der Anweisungsnummer enthaelt die laufende Nummer jeder Quellzeile

SOURCE Die uebrigen Zeilen enthalten den Quelltext.

5. Minimalformulierung eines Programms

Das in diesem Abschnitt gezeigte Beispiel zeigt die minimale PLZ/ASM-Programmstruktur fuer ein lauffaehiges Programm. Das erste Beispiel zeigt die absolut minimal erforderliche Struktur: eine Moduldefinition, eine Vereinbarungsklasse und eine Prozedurdefinition. Das zweite Beispiel zeigt das gleiche Programm unter Benutzung von symbolischen Konstanten und Datenvereinbarungen.

Beispiel 1:

```
  anyname MODULE

  GLOBAL     !oder  INTERNAL,  wenn Zwischenmodule gelinkt!
             !werden sollen!

  somename PROCEDURE
  ENTRY

       !Programmablauf!
       RET

  END somename

  END anyname
```

Beispiel 2:

```
  anayname MODULE

  CONSTANT   !Vereinbarung symbolischer Konstanten!

       one := 1
       hexten := %10


  GLOBAL     !oder INTERNAL,  wenn Zwischenmodule  gelinkt!
             !werden sollen!

       a BYTE ! Datenvereinbarung !
       b WORD
       buffer ARRAY [100 BYTE]

  GLOBAL     !Nochmaliges  Festsetzen  der   Vereinbarungs!
             !klasse [optional]!

  somename PROCEDURE
  ENTRY

       ! Programmablauf !
       RET

  END somename

  END anyname
```

6. Implementierungsmerkmale und -einschraenkungen

Fuer den U881 PLZ/ASM Assembler gelten folgende Implementierungsmerkmale und -einschraenkungen.

1. Der U881 PLZ/ASM Assembler benutzt die ASCII-Zeichenmenge. Gross- und Kleinbuchstaben werden unterschieden. Schluesselwoerter muessen vollstaendig aus Klein- oder Grossbuchstaben bestehen. (GLOBAL oder global, aber nicht Global). Hexadezimalzahlen oder spezielle Zeichenkettenzeichen koennen klein oder gross geschrieben werden.(%Ab, '1st line%R2nd line%r').

2. Quellzeilen mit mehr als 132 Zeichen werden akzeptiert, bei Fehlermeldungen werden aber nur 132 Zeichen gedruckt. Kommentare und Zeichenketten koennen ueber eine beliebige Anzahl von Zeilen verteilt sein.

   WARNUNG
   Trennzeichen fuer Kommentare (!) und Zeichenketten (') sind unzulaessig.

3. Zeichenketten der Laenge Null sind nicht zugelassen (z.B. '').

4. Konstanten werden intern als vorzeichenlose 32-bit-Worte dargestellt. Jeder Operand in einem konstanten Ausdruck wird als Typ LONG ausgewertet. Zum Beispiel ist 4/2 =2, aber 4/-2 = 0, da -2 als eine sehr grosse vorzeichenlose Zahl dargestellt wird. Bei der Brechnung konstanter Ausdruecke wird kein Ueberlauftest durchgefuehrt. Weil die Konstanten durch 32-bit-Worte dargestellt werden, werden nur die ersten vier Zeichen einer Zeichenfolge benutzt ('ABCD' = 'ABCDE'). Eine Ausnahme bilden die Zeichenketten, die zur Array-Initialisierung benutzt werden. Sie koennen eine Laenge bis zu 127 Zeichen haben.

5. Namen koennen aus maximal 127 Zeichen bestehen.

6. Nach einem Fehler innerhalb einer CONSTANT-, TYPE- oder Variablen-Vereinbarungen springt der Assembler zum naechsten Schluesselwort, mit dem eine neue Anweisung gestartet wird (z.B. ein Operationscode, IF, DO, EXIT, REPEAT oder END). Dieser Sprung muss eventuell mehrmals wiederholt werden, bevor alle Fehler bei einer Assemblierung bemerkt und korrigiert werden.

7. Das Initialisierungssymbol (?) und die rechteckigen Klammern ([und]) sind jetzt implementiert. Die Wiederholausfuehrung (...) ist noch nicht implementiert.

8. Namen existieren optional in der PLZ/ASM Datenvereinbarung. Der folgende Programmauszug zeigt ein Beispiel:

```
        .
        .
        .
BYTE    :=%10
WORD
WORD    :=%1010
LONG    :=%10101010
        .
        .
        .
```

Namen sind auch in den folgenden Faellen erforderlich:

a. Konstanten-Vereinbarungen
b. Typ-Vereinbarungen
c. Record-Feld-Vereinbarungen
d. Variable, die als vom Anwender festgelegt, vereinbart werden.

7. Anwendungshinweise

7.1. P8000 Objektkodedatei

Der "u8as" erzeugt eine Objektdatei im Standard-P8000-Lademodul-Format (s. "a.out(5)"). Die Magic-Nummer wird auf X_MAGIC3 gesetzt und stellt getrennte 8-bit Befehls- (I) und Datendatei (D) dar. Das SF_U8 Bit im Flag-Wort wird gesetzt. Die Bereiche der Objektdatei Kode, Daten und "bss" entsprechen den drei Adressraeumen des U881: Programmspeicher, externer Datenspeicher und Registerspeicher. Im Kodebereich der U881 Objektdatei sind die Informationen enthalten, die im Programmspeicher des U881 abgelegt sind. Der Datenbereich der U881 Objektdatei enthaelt Informationen, die im externen Programmspeicher angeordnet sind. Der "bss"-Bereich der Objektdatei bezieht sich auf nichtinitialisierte Daten. Alle "bss"-Referenzen in einer U881-Objektdatei sind Referenzen im Registerspeicher des U881.

7.2. Unterstuetzende Hilfsmittel

Die folgenden Programme sind Hilfsmittel fuer die Bearbeitung von Objektkode-Dateien im P8000. Mit ihrer Hilfe erhaelt man Informationen ueber die vom "u8as" erzeugten Dateien. Die vollstaendige Beschreibung dieser Hilfsmittel erfolgt im Abschnitt 1 des Bandes "WEGA-Programmierhandbuch" der P8000-Dokumentation.

1. file -- Festlegen der Dateitype fuer den P8000.

2. nm -- Ausgabe der Symboltabelle einer Objektdatei.

3. objhdr -- Ausgabe der Kopfes einer Objektdatei

4. objdu -- Ausgabe des Inhalts einer Objektdatei in lesbarer Form

5. size -- Ausgabe der in einer Objektdatei enthaltenen Groesse des Code-, Daten-, und "bss"-Bereichs

6. strip -- Entfernen der verschieblichen Information, der Symboltabelle und wahlweises des Dateikopfes aus einer Objektdatei.

7.3. P8000 Linker

Der P8000-Linker "ld" verbindet verschiedene U881 Objektdateien. Die entstehende Datei enthaelt entweder kombinierte I- und D-Bereiche oder getrennte I- und D-Bereiche. Die Kode- und Datenstartadressen koennen zum Zeitpunkt des Linkens gesetzt werden.
Eine vollstaendige Beschreibung des Linkers befindet sich in "ld(1)".
Wenn "ld" mit der Option -i zur Erzeugung getrennter I- und D-Lademodule verwendet wird, ist jede Information im Datenbereich fuer den externen Speicher des U881 Anwendersystems vorgesehen.
Wird die Option -i nicht verwendet, ist die entstehende Datei eine kombinierte I- und D-Datei. Es wird angenommen, dass im Anwenderzielsystem kein externer Datenspeicher vorhanden ist. Jede Information im Datenbereich der Datei wird in den Kodebereich des Imagespeichers anstelle des U881 Programmspeichers uebernommen.
Die Datenspeicher des U881 starten bei 800 Hex. Wenn mit "ld" eine getrennte I- und D-Datei erzeugt wird, muss die Startadresse des Datenbereiches auf 800 Hex gesetzt werden. Das wird erreicht mit der Option -b von "ld". Um die Startadresse des Datenbereiches auf 800 Hex zu setzen, wird in der Kommandozeile -bd 2048 angegeben.

Als zusaetzliche Leistung wurde das Programm "slink" in das U881 Cross-Assembler-Paket einbezogen.
Eine vollstaendige Beschreibung dieses Linkers befindet sich in "slink(1)".
"Slink" kann benutzt werden, um Kode- oder Datenbereiche waehrend der Zeit des Linkens unabhaengig voneinander in absoluten Adressraeumen abzulegen.

7.4. Beispiele fuer "ld" uns "slink"

Die folgende Kommandozeile bindet die Dateien test1.o und test2.o zusammen und erzeugt die Datei test. Das Zielsystem benutzt den externen Datenspeicher. Die entstehende Objektdatei ist ein Lademodul mit getrennten I- und D-Bereichen; der Datenbereich startet bei 800 Hex.

```
ld -i -bd 2048 -o test test1.o test2.o
```

Die folgende Kommandozeile bindet die Dateien test1.o und test2.o zusammen und erzeugt die Datei test. Das Zielsystem benutzt keine externen Speicher. Die entstehende Datei ist ein Lademodul mit kombinierten I- und D-Bereichen; jede Information des Datenbereiches wird in den Codebereich des Imagerspeichers uebernommen.

```
ld -o test test1.o test2.o
```

Soll die Datei data.o ab der Adresse %200 liegen und der Rest des Programms ab Adresse 0, ist folgende "slink"-Kommandozeile einzugeben:

```
slink file1.o file2.o file3.o -S 0x100 data.o
```

7.5. LOAD/SEND Programme

Sobald der Lademodul erzeugt ist, kann er durch das Programm "LOAD" in den Adressraum des Programmspeichers des U881-Entwicklungsmoduls geladen (download) werden (s. "load(1)"). Wenn die geladene Datei getrennte I- und D-Bereiche hat, wird der Codebereich in den Programmspeicher geladen. Wenn die geladene Datei kombinierte I- und D-Bereiche hat, wird der geladene Kodebereich durch den Datenbereich uebernommen. WEGA bietet auch die Moeglichkeit Informationen vom Entwicklungsmodul in das P8000 zu laden (upload). Es existiert eine Version des SEND-Programmes fuer den U8000-Entwicklungsmodul ("SEND") und eine Version fuer den U881-Entwicklungsmodul ("U8SEND"). Siehe dazu "send(1)" und "u8send(1)", sowie den Teil "LOAD/SEND-Kommunikation" des vorliegenden Dokumentationsbandes.

8. PLZ/ASM - Fehlermeldungen

Fehler	Erlaeuterung
	Warnungen
1	Fehlendes Trennzeichen zwischen den Zeichen
2	Array enthaelt kein Element
3	Keine Felder in Recordvereinbarung
4	Unpassender Prozedurname
5	Unpassender Modulname
8	Absolute Adresswarnung fuer P8000
	Zeichenfehler
10	Dezimalzahl zu gross
11	Unzulaessiger Operator
12	Unzulaessiges Sonderzeichen nach %
13	Unzulaessige Hexadezimalziffer
14	Zeichenkette der Laenge 0
15	Unzulaessiges Zeichen
16	Hexadezimalzahl zu gross
	DO-Schleifenfehler
20	Unangepasstes OD
21	OD erwartet
22	Unzulasssige REPEAT-Anweisung
23	Unzulaessige EXIT-Anweisung
24	Unzulaessige FROM-Marke
	IF-Anweisung-Fehler
30	Unangepasstes FI
31	FI erwartet
32	THEN oder CASE erwartet
33	Unzulaessiges Selektorregister
	Erwartete Symbole
40	) erwartet
41	(erwartet
42	] erwartet
43	[erwartet
44	: = erwartet
	Nichtdefinierte Namen
50	Nichtdefiniertes Kennzeichen
51	Nichtdefinierter Prozedurname

Vereinbarungsfehler

60 Typkennzeichen erwartet
61 Unzulaessige Modulvereinbarung
62 Unzulaessige Vereinbarungsklasse
63 Unzulaessige Verwendung von array [*] Vereinbarung
64 Array [*] Vereinbarung ohne Initialisierung
65 Unzulaessige Dimensionsgroesse
66 Unzulaessiger Komponententyp eines Array
67 Unzulaessige Record-Feld-Vereinbarung

Prozedur-Vereinbarungsfehler

70 Unzulaessige Prozedurvereinbarung
71 ENTRY erwartet
72 Prozedurname nach END erwartet

Initialisierungsfehler

80 Unzulaessiger Anfangswert
81 Zu viele Initialisierungselmente fuer die vereinbarten Variablen
82 Unzulaessige Initialisierung
83 Array[*] Vereinbarung wurde mit einer leeren Zeichenfolge initialisiert
84 Initalisierungsversuch eines nichtinitialisierten Datenbereichs

Spezielle Fehler

90 Unzulaessige Anweisung
91 Unzulaessiger Befehl
92 Unzulaessiger Operand
93 Operand zu gross
94 Relativadresse ausserhalb des Bereiches
95 : erwartet
97 Record-Feldname doppelt
98 Case-Konstante doppelt
99 Mehrfache Namensvereinbarung

Unzulaessige Variable

100 Unzulaessige Variable
101 Unzulaessiger Operand fuer # oder SIZEOF
102 Unzulaessiger Feldname
103 Indizierung einer Nicht-Array-Variablen
104 Unzulaessige Verwendung des Punktes (.)

Fehler in Ausdruecken

110 Unzulaessiger arithmetischer Ausdruck
111 Unzulaessiger bedingter Ausdruck
112 Unzulaessiger konstanter Ausdruck
113 Unzulaessiger Auswahlausdruck
114 Unzulaessiger Indexausdruck
115 Unzulaessiger Ausdruck in einer Zuordnung

Konstanten ausserhalb der Grenzen

120 Konstante zu gross fuer 8 Bits
121 Konstante zu gross fuer 16 Bits
122 Konstanter Array-Index ausserhalb der Grenzen

Typ-Inkompatibilitaet

140 Array [*]-Initialisierung mit Zeichenfolgen vom Typ ungleich 8 Bit
141 Typ-Inkompatibilitaet bei Initialisierung

Segmentierungsfehler

170 Unzulaessiger Operator im nichtsegmentierten Modus
171 Unpassender Adressoperator
172 Unpassender Segmentbezeichner

Anweisungsfehler

180 Falsche Sektionsspezifizierung
181 Unzulaessige Sektionszuordnung
182 Unpassende bedingte Assembleranweisung
183 Unzulaessiger Ausdruck bei bedingter Assemblierung
184 Versuch, segmentierten und nichtsegmentierten Kode zu mischen
185 Anweisung muss allein auf einer Zeile stehen
186 Unzulaessige $CODE oder $DATA Anweisung

Datei-Fehler

198 EOF erwartet
199 Unerwartetes EOF in der Quelle, eventuell durch unpassendes ! oder ' in der Quelle

Implementierungseinschraenkungen

224 Zu viele Symbole - Ueberlauf der Hash-Tabelle
226 Kurzes Segment-Offset ausserhalb des Bereiches
227 Ueberlauf der Objekt-Symbol-Tabelle
228 Vorgegebener Speicherbereich wird ueberschritten
229 Merkmal nicht implementiert
230 Zeichenkette des Merkmals zu lang
231 Zu viele Symbole - Ueberlauf der Symboltabelle
234 Zu viele Initialisierungswerte
235 Stack-Ueberlauf
236 Operand zu kompliziert

Teil 2

W E G A

U880 Cross-Assembler

Nutzerhandbuch

Inhaltsverzeichnis Seite

1. Einfuehrung

Das Handbuch beschreibt den U880-Assembler, der unter dem Betriebssystem WEGA laeuft. Das Entwicklungspaket fuer die Erstellung des U880-Kodes beinhaltet den U880-Assembler und andere Hilfsmittel, wie den Linker "u80ld".
Programme in der U880-Assemblersprache koennen assembliert, gelinkt und dann einer U880-Peripherie uebergeben werden. Der ebenfalls fuer den Nutzer verfuegbare U880 C-Compiler erzeugt die Assemblersprache fuer den U880. Das Handbuch beschreibt die fuer den "u80as"-Assembler erforderliche Syntax.
Es ist zu beachten, dass der "u80as"-Assembler fuer P8000 nicht identisch ist mit dem U880-Assembler, der unter UDOS verwendet wird. Ein Anhang des Handbuches ist den Unterschieden gewidmet.

Der "u80as"-Assembler ist ein verschiebbarer Makroassembler fuer den Mikroprozessor U880, der die symbolische Assemblersprache in einen verschiebbaren Objektkode uebersetzt. Der erstellte Objektkode hat "a.out"-Format (siehe hierzu auch "a.out(5)" im WEGA-Programmierhandbuch). Der verschiebbare Objektkode kann durch den Linker "u80ld" verarbeitet werden, der einen nichtverschiebbaren Objektkode erzeugt.
Der Assembler ist ein Zweistufenprogramm, das unterstuetzende Fehlernachrichten ausgibt und sowohl ein uebersichtliches Programmlisting, als auch eine Symbol-Cross-Referenzliste erstellt. Der Assembler fuehrt Verschiebung, Makrobearbeitung, bedingte Assemblierung, symbolische und relative Adressierung, Adressbezuege und komplexe Ausdrucksberechnung aus, erstellt eine Cross-Referenzliste und fuehrt umfangreiche Uebersetzungsanweisungen aus.
Der Linker ("u80ld") verarbeitet ein oder mehrere vorher vom Assembler erzeugte verschiebbare Objektdateien und verbindet sie zu einer nichtverschiebbaren (absoluten) Datei.

Nach Assemblierung ("u80as") und Linken ("u80ld") eines Programms steht der U880 Objektkode im P8000 im "a.out"-Format zur Verfuegung. Er kann mittels "objfx(1)", das ein Teil dieses Entwicklungspaketes ist, ueber einen seriellen Port im "I-Hex"- oder "T-Hex"-Format ausgegeben werden.
Bei Kommunikation eines UDOS-Systems mit einem P8000 ueber eine serielle Schnittstelle mittels dem "Remote"-Kommunikationspaket kann der U880-Objektkode im "a.out"-Format zum UDOS-System mittels "putfile(1)" uebertragen werden (Bsp.: putfile -b a.out). Desgleichen ist die Ausgabe des Objektkodes an das UDOS-System des P8000 ueber "putud(1)" moeglich (Bsp.: putud -b a.out). Anschliessend kann der U880-Objektkode im "a.out"-Format in eine UDOS-Datei mittels dem Programm "UDOSCNVT", das ebenfalls Bestandteil dieses Entwicklungspaketes ist und unter UDOS laeuft, konvertiert und durch Emulation abgearbeitet werden
(Bsp.: UDOSCNVT a.out udosfile).

2. Kommandozeile

NAME

u80as - U880 Cross-Assembler

SYNTAX

u80as [-options] [outfile] infile [-m makrofile(s)]

BESCHREIBUNG

Der U880-Assembler assembliert die Datei mit dem Namen "infile.s" und erzeugt eine verschiebliche Objektdatei mit dem Namen "infile.o", falls "outfile" nicht angegeben wurde. Makrodateien werden, falls sie angegeben wurden, zusammen mit "infile.s" assembliert. Die folgenden Optionen werden von "u80as" interpretiert:

-u Bewirkt, dass alle nichtaufgeloesten symbolischen Namen als externe Namen gekennzeichnet werden.

-i Bewirkt, dass der Assembler verschiedene Fehlerarten ignoriert, die in der Regel vom Cross-Compiler erzeugt wurden (z.B. fehlende "end"-Anweisung).

-o Bewirkt, dass der darauffolgende Dateiname "outfile" den Standarddateinamen fuer die Objektdatei "infile.o" ersetzt.

-x Bewirkt, dass durch den Assembler eine Cross-Referenzliste des Programms erstellt wird. Heisst die Eingabedatei "infile.s", wird die Liste in der Datei "infile.l" abgespeichert.

-l Erstellt ein Listing, das den Objektkode und dessen Lokalisation enthaelt. Heisst die Eingabedatei "infile.s", wird das Listing in der Datei "infile.l" abgespeichert.

-lx Erstellt ein Listing, das den Objektkode und dessen Lokalisation, sowie die Cross-Referenzliste enthaelt. Heisst die Eingabedatei "infile.s", wird das Listing in der Datei "infile.l" abgespeichert.

3. U880 Assemblervereinbarungen

3.1. Assembleruebersicht

Der Zweck eines verschiebbaren Makroassemblers ist das Uebersetzen eines U880-Assemblerprogramms in einen verschiebbaren binaeren Objektkode. Ein in Assembler geschriebenes Programm enthaelt Anweisungen. Eine Anweisung ist entweder ein symbolischer Befehl, eine Uebersetzungsanweisung, eine Makroanweisung oder ein Kommentar. Kommentare werden ignoriert, ausser bei Listings. Die Makroanweisungen werden in Befehle, Uebersetzungsanweisungen oder Kommentare uebersetzt. Die Maschinenbefehle sind direkt in den entsprechenden Binaerkode uebersetzbar. Die Assembleruebersetzungsanweisungen uebermitteln spezielle Informationen und koennen die Binaerkodeerzeugung veranlassen oder unterbinden.

Die Uebersetzung wird in zwei Hauptschritten vorgenommen:
Beim ersten wird die Eingabequelle durchgemustert, die Syntaxanalyse vorgenommen, die Adressen aller Marken notiert und die zugehoerige Zwischensprache auf eine temporaere Datei ausgelagert.
Beim zweiten Schritt wird die Zwischensprache durchmustert, der Binaerkode erzeugt und dieser neben einer Kopie der Original-Zeileninhalte aufgelistet.

Die durch den Assembler erzeugte Ausgabe beinhaltet die bereits genannte Druckausgabe und einen verschiebbaren Modul, der den Binaerkode enthaelt, der die Uebersetzung des Assemblerprogramms ist. Jeder Fehler, der vom Assembler erkannt wird, wird im Listing direkt unter dem fehlerhafte Befehl notiert.

3.2. Assembleranweisungsformat

Generelle Anweisungssyntax:
- Anweisungen werden durch ein "New Line"-Zeichen oder den Beginn eines Kommentares abgeschlossen

Programmarken und Identifikatoren (Bezeichner):
- Es gibt keine Fallunterscheidung. Ein grossgeschriebener Buchstabe ist einem kleingeschriebenen gleich (ausser in bewerteten Zeichenketten). Somit ist ein "A" das gleiche wie ein "a"; ferner ist "LABEL" das gleiche wie "label" oder "laBel".
- Die Markenlaenge ist bei diesem Assembler auf 80 Zeichen begrenzt.
- Externe Symbole, wie z.B. der Einsprungpunkt, duerfen nur einmal innerhalb der ersten acht Zeichen auftreten.

Befehle: Ein Befehl ist die mnemonische Assemblersprachen-Beschreibung eines bestimmten durchzufuehrenden Vorganges. Er muss von seinen Operanden durch ein Trennzeichen getrennt sein.

Operanden:
In Abhaengigkeit vom auszufuehrenden Befehl kann das Feld keinen oder mehr Operanden haben. Zwei oder mehr Operanden sind durch Komma zu trennen. Ein Zwischenraum (Space) ist nicht zulaessig.
Operanden enthalten die Informationen des Befehls, um ihn auszufuehren.
Ein Operand kann sein:
- Daten, die unmittelbar verarbeitet werden
- Die Adresse eines Speicherplatzes, von dem die Daten entnommen wurden (Quelladresse)
- Die Adresse eines Speicherplatzes, zu dem die Daten abgelegt werden (Zieladresse)
- Die Adresse eines Programmspeicherplatzes, zu welchem die Programmsteuerung Zugriff hat.
- Eine bedingte Anweisung, die die Programmsteuerung beeinflusst.

Obwohl es eine Reihe von zulaessigen Operandenkombinationen gibt, sei an eine Hauptvereinbarung erinnert: der Zieloperand hat stets Vorrang vor dem Quelloperand.

Kommentare:
Kommentare werden im Listing zur besseren Darstellung der Programmlogik verwendet und vereinfachen damit eine sofortige oder spaetere Fehlersuche. Sie koennen ueberall eingefuegt werden. Kommentare beginnen mit einem Semikolon (;) und gehen bis zum Zeilenende. Sie koennen jedes beliebige Zeichen beinhalten.

3.3. Arithmetische Operanden

Ein unaeres Minus oder ein unaeres Plus sind auf allen Ebenen erlaubt, ausser bei unmittelbar folgendem Uebersetzungszeit-Operatorsymbol. In diesem Fall ist der gewuenschte Ausdruck in runde Klammern zu setzen. Es sollte auch beachtet werden, dass die Verwendung von Adressbezugsmarken (definiert durch Verwendung von "equ"- oder "def"-Anweisungen) in Verbindung mit einigen anderen Operatoren zu Fehlermitteilungen fuehrt.
Wenn Elemente durch Verwendung von Uebersetzungszeit-Operatoren kombiniert werden, ist der resultierende Wert verschiebbar oder absolut. Absolute Elemente sind Werte, die nicht durch die Verschiebung eines Assemblerprogramms veraendert werden. Verschiebbare Elemente sind Werte, z.B. Adressen, die innerhalb oder ausserhalb des Assemblerprogramms definiert sind und bis zur Verschiebung des Assemblerprogramms nicht aufgeloest sind.
Die gesamte Uebersetzungszeit-Arithmetik wird durch eine 32-Bit-Arithmetik realisiert. Eine Fehlernachricht wird generiert, wenn ein Wert groesser ist, als das angegebene Befehlsfeld es zulaesst und der Wert deshalb auf die entsprechende Bitanzahl gekuerzt wird.

Beachte: Im Ausdruck zwischen Operand und Operator ist kein Leerzeichen erlaubt und die Operatoren sind klein zu schreiben.

+	unaeres Plus
-	unaeres Minus
.not.	logisches Nicht
*	Multiplikation
/	Division
.mod.	Modulo
.shr.	log. Rechtsverschiebung
.shl.	log. Linksverschiebung
+	Addition
-	Subtraktion
.and.	logisches UND (bitweise)
.or.	logisches ODER (bitweise)
.xor.	logisches XOR (bitweise)
=	gleich
<>	ungleich
>	groesser als
>=	groesser oder gleich
<	kleiner als
<=	kleiner oder gleich

Beachte: Der Gebrauch von symbolischen Ausdruecken, bei denen der zweite Operand einen U880-Befehl darstellt, ist zu vermeiden. Manchmal berechnet der Assembler den ersten Teil und verwirft den Rest. Zum Beispiel:

```
  fuenf:   equ    5
  zwei:    equ    2
           ld     a,fuenf+zwei  ; nicht empfohlen
  sieben:  equ    fuenf+zwei
           ld     a,sieben      ; empfohlen
```

Der maximal zu berechnende Ausdruck kann nur 20 "Push"-Ebenen in umgekehrter polnischer Notation haben.

3.4. U880 Adressierungsarten

Ein Offsetwert Null ist festzulegen. Das Offset-Feld ist nichtoptional.

4. Befehlsvorrat der Assemblersprache

Der U880-Befehlsvorrat umfasst 158 Befehle. Sie beinhalten:

- Adressierungsarten
- Flags und Statussymbole
- Interruptarten
- Lade- und Austauschbefehle
- Blocktransfer- und Suchbefehle
- arithmet. und logische Befehle
- Rotations- und Verschiebebefehle
- Bit-Manipulationsbefehle
- Sprung-, Aufruf- und Ruecksprungbefehle
- Ein-/Ausgabebefehle
- Sonstige Befehle

Siehe hierzu auch Anhang A.

5. Strukturierung eines U880-Programms

5.1. Programmstruktur

Einige Assembler haben hoehere Strukturen. Der "u80as" Assembler hat keine der nachfolgend aufgefuehrten Strukturelemente:

- Module
- Prozeduren
- Schleifen
- IF-Laufanweisung

Die einzigsten implementierten Elemente sind globale, externe und interne Variable.

5.2. Verschiebbare Programmabschnitte

Die Assembleranweisungen $ABS, $REL, $DEFAULT bzw. $SECTION sind nicht implementiert.
Der Adresszaehler erlaubt das Zerlegen eines Programms in Teile, die in verschiedenen Speicherbereichen untergebracht sein koennen, wenn das Programm fuer die Abarbeitung gelinkt ist. Der Assembler beginnt im Normalfall die Programmeinteilung ab Adresszaehlerstand Null, sofern der Adresszaehler durch die "BEGIN"-Anweisung nicht geaendert wurde.
Teile eines Programms sind Befehle, Daten und "bss" (nicht initialisierte Daten). Abschnitt 0 ist der Kode (BEGIN 0); Abschnitt 1 sind Daten (BEGIN 1) und es gibt einen "bss"-Abschnitt fuer "COMMON"-Symbole. Der U880 C-Compiler teilt die Programme selbstaendig in einen ausfuehrbaren Abschnitt mit Adresszaehler 0 und einen Datenabschnitt mit Adresszaehler 1 ein. Die Adresszaehler sind verschiebbar; am Beginn der Assemblierung gilt der Standardwert 0.

Man beachte: Es sollten keine Adresszaehlwerte ausser 0 und 1 verwendet werden. Der "u80as"-Assembler erkennt zwar 32 Adresszaehlerwerte (von 0 bis 31), aber im Objektkode und im "u80ld"- Linker lassen sich nur Adresszaehlerwerte von 0 und 1 unterbringen. Ferner hat die "COMMON"-Anweisung den Bereich 30. Somit benutze man die "COMMON"-Anweisung und nicht "BEGIN 30".

5.3. Arten arithmetischer Ausdruecke

Alle arithmetischen Ausdruecke entsprechen einer der drei Arten: absolut, verschiebbar oder extern.

- Ein absoluter Ausdruck besteht aus einer oder mehreren Konstanten, Bezeichnern oder absoluten Marken, die mit arithmetischen oder logischen Operatoren kombiniert sind. Die Differenz zwischen zwei verschiebbaren Ausdruecken ist immer absulut.
- Ein verschiebbarer Ausdruck enthaelt genau einen Bezeichner, der von der Verschiebung nach der Assemblierung abhaengt. Der Ausdruck kann durch Addition oder Subtraktion eines absoluten Ausdrucks erweitert werden. Plus oder Minus sind jedoch nur als Operatoren erlaubt.
- Ein externer Ausdruck hat genau einen externen Bezeichner, ggf. erweitert durch Addition oder Subtraktion eines absoluten Ausdrucks. Ein externer Bezeichner ist eine Variable die im gerade laufenden Programmteil benutzt, aber in einem anderen definiert ist. Sein Wert ist vor dem Linken der Programmteile nicht bekannt.

Man beachte: Die maximale Anzahl externer Variablen ist in einer beliebigen Programmeinheit auf 1023 begrenzt.

6. Assemblersteueranweisungen und erweiterte Befehle

Die Arbeit eines Assemblers wird durch die im folgenden erlaeuterten Assemblersteueranweisungen geregelt. Eine Steueranweisung ist ihrem Wesen nach ein Befehl zum Assembler, mit dem Ziel, eine Hilfsfunkion auszufuehren. Die Syntax der Steueranweisung gleicht der eines Befehls mit den dargestellten Ausnahmen. Das Markenfeld und das Operandenfeld koennen je nach aufgerufener Steueranweisung erforderlich oder verboten sein. Die explizite Syntax und die Wirkung einer angegebenen Steueranweisung sind im folgenden Abschnitt beschrieben.

6.1. Steueranweisungen fuer Symboldefinitionen (equ, def)

equ Gleichsetzung eines Symbols fuer einen Ausdruck

Zweck der "equ"-Steueranweisung ist die Gleichsetzung einer Marke fuer einen Ausdruckswert; somit erlaubt sie die Verwendung von Marken anstelle von Ausdruecken. Bei einer "equ"-Anweisung wird dem Symbol im Markenfeld Laenge, Wert, Typ und Verschiebeangaben des Ausdrucks im Operandenfeld zugewiesen. Sowohl Markenfeld als auch der Ausdruck im Operandenfeld muss in jeder "equ"-Steueranweisung enthalten sein. Der Ausdruck muss zum Zeitpunkt der Abarbeitung berechenbar sein (d.h. keine Adressbezuege). Das Symbol im Markenfeld darf nicht in einem anderen Markenfeld erscheinen, mit Ausnahme einer "def"-Steueranweisung und es muss vor Gebrauch im Assemblerprogramm definiert sein.
Diese Steueranweisung kann auch verwendet werden, um neue Schluesselwoerter zu definieren, die anstelle vordefinierter Assembler-Schluesselworte benutzt werden. Das erlaubt dem Anwender, den Registern des Prozessors sachbezogene Namen zuzuweisen.
Beispiele:

```
2   0014   3 zwanzig:   equ   20
3   0015   4 cntrg:     equ   hl
4   000f   5 dif:       equ   zwanzig - 5
```

def "def"-Steueranweisung

Das Ziel dieser Anweisung ist aehnlich dem der "equ"-Steueranweisung: die Zulaessigkeit eines Symbols fuer einen Ausdruck darzustellen, dessen Berechnung im Programm erfolgt. Bei einer "def"-Anweisung wird dem Symbol im Markenfeld Laenge, Wert, Typ und Verschiebeangaben des Ausdrucks im Operandenfeld zugewiesen. Sowohl Markenfeld als auch der Ausdruck im Operandenfeld muessen in jeder "def"-Anweisung enthalten sein, aber im Gegensatz zur "equ"-Anweisung braucht der Ausdruck zum Abarbeitungszeitpunkt nicht berechenbar zu sein. Das Symbol kann durch eine nachfolgende "def"-Anweisung neu definiert werden. Es hat dann den Wert des Ausdruckes, der ihm durch die letzte "def"-Anweisung nach Abarbeitung durch den Assembler zugewiesen wurde. Der Verwendung von Mehrfachdefinitionen mit Adressbezuegen sollte Aufmerksamkeit geschenkt werden, wenn

fuer den Anfangswert einer adressbezogenen, verschiebbaren Marke Null angeommen wurde und mehrere Assemblerlaeufe bereits stattgefunden haben.
Beispiele:

```
1 ffff   1   st1:   def   st2-st3
2 0009   2   st2:   def   9
3 000a   3   st3:   def   10
```

6.2. Steueranweisungen fuer Datendefinitionen (db, dw, text)

db Byte-Datendefinition

Zweck der Definition ist die Einbringung von Byte-Konstanten in ein Assemblerprogramm. Bei der Anwendung wird die Berechnung des Ausdruckes im Operandenfeld veranlasst und der Wert auf dem naechsten verfuegbaren Speicherplatz abgelegt. Das Markenfeld der Anweisung ist optional und, wenn vorhanden, bezieht es sich auf den Anfangsspeicherplatz der Konstanten. Bei dieser Assemblerversion darf nur ein Ausdruck im Operandenfeld enthalten sein.
Beispiel:

```
0 000000   1 00ff   1 db   0ffh
```

dw Wort-Datendefinition

Zweck der Definition ist die Einbringung von Wort-Konstanten in ein Assemblerprogramm. Bei der Anwendung wird die Berechnung des Ausdrucks im Operandenfeld veranlasst und der Wert, beginnend an einer Wortbegrenzung, auf den naechsten verfuegbaren Speicherplatz abgelegt. Das Markenfeld der Anweisung ist optional und, wenn vorhanden, bezieht es sich auf den Anfangsspeicherplatz des Wortes. Das fuehrt dazu, dass dem Symbol im Markenfeld ein Wert entsprechend der Begrenzung des konstanten Wortes zugewiesen wird, wenn dies noetig ist. Bei dieser Assemblerversion darf nur ein Ausdruck im Operandenfeld enthalten sein.
Beispiel:

```
0 000000   1 ffff   1 dw   0ffffh
```

text Textdefinition

Damit wird dem Nutzer das Assemblieren einer Zeichenkette als Daten erlaubt, bei denen die Daten mehr als zwei Worte Speicherplatz belegen. Bei der Anwendung wird die Zeichenkette linksbuendig in soviel Worten eingetragen, wie fuer die Zeichenkette notwendig sind. Wenn die umgesetzte Zeichenkette nicht an einer Wortbegrenzung endet, werden die restlichen Zeichen des letzten Wortes mit Leerzeichen aufgefuellt. Die gesamte Textanweisung muss in einer Zeile untergebracht werden koennen. Das Markenfeld ist optional und, wenn vorhanden, bezieht es sich auf das erste gespeicherte Wort des Textes. Eine Zeichenfolge ist als Operandenfeld erforderlich.

Beispiel:
```
 0 000000   1 7468    1 string:  text  'this is an example'
 0 000002     6973
 0 000004     2069
 0 000006     7320
 0 000008     616e
 0 00000a     2065
 0 00000c     7861
 0 00000e     6d70
 0 000010     6c65
```

6.3. Steueranweisungen zum Programmlinken (public, extrn)

public Steueranweisung fuer Einsprungpunkt

Zweck der Anweisung ist das Festlegen von Symbolen fuer den Linker, die in diesem Assemblersprachenprogramm definiert sind, aber auch auf andere Programme bezogen sein koennen. Bei der Verwendung wird veranlasst, dass jedes Symbol im Operandenfeld als ein Einsprungpunkt fuer den "u80ld"-Linker deklariert ist, der diese dann zur Aufloesung externer Bezuege, aufgefordert durch anderer Programme, benutzen kann. Mindestens ein Symbol ist im Operatorenfeld erforderlich und eine Symbolliste ist zulaessig, vorausgesetzt die Symbole sind durch Komma getrennt. Jedes Symbol im Operandenfeld der Anweisung muss in einem Markenfeld an irgendeiner Stelle des Assemblerprogramms definiert sein. Da der "u80ld"-Linker nur die ersten 8 Zeichen einer Marke zur Erkennung der Eindeutigkeit benutzt, erfolgt die Angabe einer Fehlerwarnung fuer Namen, die laenger als 8 Zeichen sind.
Beispiel:
```
            1         1         public   item,value,start
            2 000a    2 value:  equ      10
 0 000000   3 0014    3 item:   db       20
 0 000001   4 a121    4 start:  ld       r1, r2
 0 000003   5         5         end
```

extrn Extern-Steueranweisung

Zweck ist das Erkennen von Linker-Symbolen, die in diesem Assemblerprogramm verwendet werden, aber in einem anderen Programm definiert sind. Bei der Anwendung wird veranlasst, dass jedes Symbol im Operandenfeld als externer Bezug fuer den "u80ld"-Linker festgelegt ist. Dieser loest die externen Bezuege auf und modifiziert die Befehle, auf die sie gerichtet sind.
In Verbindung mit der "public"-Steueranweisung in anderen Programmen, erlaubt die "extrn"-Steueranweisung das Verzweigen auf Adressen oder den Zugriff auf Daten, die in diesen Programmen definiert sind. Das Markenfeld ist optional und bezieht sich auf den Wert des aktuellen Adresszaehlers. Im Operandenfeld ist mindestens ein Symbol erforderlich; eine Symbolliste ist zulaessig, wenn die Symbole durch Komma getrennt sind. Maximal 1023 Symbole duerfen als externe Bezuege im Assemblerprogramm deklariert sein. Die

auftretenden Symbole im Operandenfeld duerfen nicht im Markenfeld an einer anderen Programmstelle existieren und jedes externe Symbol muss als solches deklariert sein.
Es sollte beachtet werden, dass nur Symbole mit 8 oder weniger Zeichen als externe Bezuege oder Linker-Einsprungpunkte erkannt werden. Symbole mit mehr als 8 Zeichen koennen im Operandenfeld der "public"- oder "extrn"-Steueranweisung verwendet werden, aber nur die ersten 8 Zeichen bestimmen deren Eindeutigkeit. Bei Operanden mit mehr als 8 Zeichen erfolgt die Ausgabe einer Warnung.
Beispiel:

```
            1        1        public   item, value, start
            2        2        extrn    exlab
            3 000a   3 value: equ      10
 0 000000   4 0014   4 item:  db       20
 0 000001   5 a121   5 start: ld       r1, r2
 0 000003   6 e8fD   6        jr       nc,exlab
 0 000005   7        7        end
```

6.4. Steueranweisungen fuer Assemblerlisting

title Steueranweisung fuer Ueberschrift

Zweck ist die Kennzeichnung des Assemblerlistings fuer den Nutzer durch Drucken der festgelegten Zeichenkette am Anfang und Ende jeder Seite des Listings. Bei Verwendung wird ein Papiervorschub ausgefuehrt und die Zeichenkette auf allen folgenden Seiten gedruckt. Das Markenfeld der "title"-Anweisung muss leer sein, das Operandenfeld muss aus einer in Apostrophen gesetzten Zeichenkette bestehen. Diese darf bis zu 55 Zeichen beinhalten. Man beachte, dass die Zeilenabbildung auf der die Anweisung steht, im Listing nicht gedruckt wird.

eject Steueranweisung fuer Papiervorschub

Zweck ist es, zu realisieren, dass die naechste Listing-Zeile am Anfang einer neuen Seite gedruckt wird. Marken- und Operandenfeld sind nicht erlaubt und die Zeilenabbildung wird ebenfalls nicht gedruckt.

space Steueranweisung fuer Zeilenvorschub

Zweck ist, die Moeglichkeit des Einfuegens einer oder mehrerer Leerzeilen in das Listing. Bei Anwendung wird veranlasst, dass eine Anzahl von Leerzeilen, die durch den Wert des Ausdruckes im Operandenfeld angegeben wird, eingefuegt werden. Das Markenfeld ist nicht erlaubt, waehrend das Operandenfeld, das aus einem einzigen absoluten Wert bestehen soll, erforderlich ist. Stimmt der Wert des Ausdruckes mit der auf der betreffenden Seite noch vorhandenen Leerzeilen ueberein, hat diese Anweisung die gleiche Wirkung wie die Steueranweisung fuer den Papiervorschub.

list Steueranweisung zum Starten des Listings

Zusammen mit der Anweisung zum Beenden des Listings, erlaubt diese Anweisung dem Nutzer die Listing-Ausgabe zu steuern. Bei Anwendung wird die Listing-Ausgabe aktiviert, sofern sie nicht schon anderweitig aktiviert war. Marken- und Operandenfeld sind nicht erlaubt. Die Zeilenabbildung wird nicht gedruckt, unabhaengig vom Aktivierungsstatus. Bei Aufruf der Anweisung im Rahmen einer Makro-Ausdehnung betrifft sie nur die Zeilenabbildungen in diesem und den mitwirkenden Makros.

unlist Steueranweisung zum Beenden des Listings

Sie dient der Deaktivierung der Listing-Ausgabe und stellt das Gegenstueck zur "list"-Anweisung dar. Zur Wirkung gilt das unter "list" gesagte.

6.5. Steueranweisungen zur Programmeinteilung (begin, common)

begin Steueranweisung fuer Beginn des Programmabschnitts

Mit dieser Anweisung wird dem Assembler angegeben, dass der nachfolgende Block von Daten oder Befehlen unter den genannten Adresszaehler assembliert wird. Der Assembler gestattet die Verwendung bis zu 32 Adresszaehlern (von 0 bis 31).
Beachte: Man sollte aber nur Adresszaehler von 0 fuer Befehle und 1 fuer Daten verwenden, da nur diese dem Objektdatei-Format entsprechen und vom Linker "u80ld" erkannt und verarbeitet werden.

Bei Anwendung der "begin"-Steueranweisung wird der Wert des Adresszaehlers initialisiert, der durch den Ausdruck im Operandenfeld zum naechsten verfuegbaren Speicherplatz unter diesem Adresszaehler vorgegeben ist. Der Wert des Adresszaehlers ist Null, wenn der Adresszaehler vorher im Programm nicht benutzt wurde. Das Markenfeld der Anweisung ist optional, und, wenn enthalten, wird das Symbol dem naechsten verfuegbaren Wert des Adresszaehlers zugewiesen. Das Operandenfeld muss einen ganzzahligen absoluten Ausdruck zwischen 0 und 31 beinhalten. Der vom Assembler am Anfang zugewiesene Adresszaehlwert 0 wird verwendet, bis eine neue "begin"-Anweisung ihn anderweitig bestimmt.
Beispiel:

```
        1        1 ; display module
        2        2         begin 0
        3        3 ; constant
        4 0000   4 rxrdy:   equ 0
```

common Steueranweisung fuer den "bss"-Bereich

Die Anweisung veranlasst, fuer nichtinitialisierte Daten einen "bss"-Bereich zu reservieren. Das erste Argument ist der Variablenname, das zweite Argument ist die Breite des reservierten Bereiches. Nach Benutzen einer "common"-Anweisung zur Platzreservierung kann der Name wie eine Marke weiter verwendet werden. Die Anweisung kann in Daten- oder Befehlselementen erscheinen, in jedem Fall reserviert es Platz im "bss"-Bereich. Im Objektkode-Format ist "bss" effektiv Sektion 30. Man benutze aber nicht "begin 30", sondern die "common"-Anweisung, um Marken in den "bss"-Bereich zu legen.
Beispiel:

```
        1            1         begin  0
        2            2 ; reserve 32 bytes common for uni
        3            3         common uni, 020h
        4 2a 00 00   4         ld     hl, (uni)
        5 22 00 00   5         ld     (uni), hl
```

end Steueranweisung fuer Assemblierungsende

Die Anweisung kennzeichnet das Ende eines Assemblerprogramms. Wird eine solche "end"-Anweisung erreicht, beendet der Assembler das Lesen der Eingabe und initiiert die nachfolgende Programmausfuehrung. Wird das Ende des Quellprogramms erreicht, ehe die "end"-Anweisung erscheint, wird eine Fehlernachricht generiert. Das Markenfeld ist optional, und, wenn enthalten, wird es dem aktuellen Adresszaehlerwert zugewiesen.
Das Operandenfeld ist ebenfalls optional, und, wenn enthalten, besteht es aus einem einzigen Ausdruck. Dieser Ausdruck wird berechnet und als Startadresse des Programms benutzt.

7. Bedingte Assemblerbloecke

Der Zweck bedingter Assemblerbloecke ist die Schaffung der Faehigkeit, dass angegebene Assemblersprachenbloecke wiederholt und assembliert werden. Vorsicht ist geboten, wenn ein bedingter Assemblerblock zwischen Makrobezugs-Niveaus aufgeteilt ist. Alle Steueranweisungen, die mit einem bedingten Assemblerblock im Zusammenhang stehen, sollten innerhalb des Hauptprogramms oder des gleichen Makrobezuges liegen.
Ein bedingter "if"-Block besteht grundsaetzlich aus einer "if"-Anweisung, gefolgt von einem Block von Zeichen, die zu assemblieren sind, wenn der Wert des Operandenfeldes (der "if"-Anweisung) ungleich Null ist; optional gefolgt von einer "else"-Anweisung und einem zweiten Block von Zeichen, die zu assemblieren sind, wenn der Wert des Operandenfeldes Null ist. Das ganze wird durch eine "endif"-Anweisung beendet.
Ausgehend vom Erkennen einer "if"-Anweisung berechnet der Assembler den Ausdruck im Operandenfeld und benutzt den Wert, um festzulegen, ob der nachfolgende Assemblersprachenblock zu assemblieren ist. Ist der Wert ungleich Null, dann wird der Block assembliert, andernfalls wird er nur eingelesen und ignoriert. Alle Assemblersprachen-Quellbefehle, die sich zwischen der "if"-Anweisung und der naechsten "else"- oder "endif"-Anweisung befinden, sind als erster Assemblerblock definiert. Diese Definition ist dahingehend erweiterbar, dass ein Assemblersprachenblock bedingte Assemblerblocks beinhalten kann, die wieder andere bedingte "if"-Blocks enthalten koennen.
Eine "else"-Anweisung kann benutzt werden, um diesen ersten Assemblerblock abzuschliessen und festzulegen, dass der zweite Assemblersprachenblock assembliert wird, wenn die "if"-Anweisung Null war; andernfalls wird der zweite Block nur eingelesen und ignoriert. Die Syntaxfestlegung des zweiten Assemblersprachenblocks ist mit der ersten identisch, ausser dass nur eine "endif"-Anweisung als Abschluss des Blockes zulaessig ist. Ohne Ruecksicht auf das Vorhandensein einer "else"-Anweisung ist eine "endif"-Anweisung zum Abschluss jedes bedingten "if"-Blocks erforderlich.

Beispiele:

```
  126                   182  ;       the following code
  127                   183  ;
  128                   184  ;flag:   set     1
  129                   185  ;        if flag
  130                   186  ;          add   r4, 1(r0)
  131                   187  ;        else
  132                   188  ;          sub   r5, 2(r0)
  133                   189  ;        endif
  134                   190  ;
  135                   191  ;    produces this result
  136                   192  ;
  137           0001    193  flag:    set     1
  138 0 0013B   8404    195          add     r4, 1(r0)
```

8. Makrobezuege

Es kommt oft vor, dass in Assemblerquellanweisungen bestimmte Folgen wiederholt an verschiedenen Stellen auftreten und nur geringfuegig voneinander abweichen. Dem Nutzer ist es moeglich, sich auf vordefinierte Makros zu beziehen, um diese angegebenen Folgen von Assemblerquellanweisungen zu generieren und als Hilfsmittel zum Einfuehren verschiedener Informationen in allgemeine Folgen zu benutzen. Jedes Makro, auf das man sich nachfolgend in einem Assemblerprogramm bezieht, muss zuerst in einer Makrobibliothek definiert sein, indem die zusammengehoerenden Folgen der Assemblerquellanweisung angegeben werden.
Immer wenn der Assembler in einer Operationsfeld-Anweisung ein Symbol herausfindet, das kein gueltiges im Operationsfeld fuer einen Befehl, Assemblierung oder ein bedingter Block ist, wird die Annahme gemacht, dass es sich um einen Makronamen handelt und die Anweisung als Makrobezug interpretiert. Der Assembler sucht dann in den Makro-Bibliotheken nach einer Datei, die den Namen der Anweisung im Operationsfeld traegt. Diejenige Assemblerquellanweisung, die im ersten mit den angegebenen Namen abgelegten Datei gefunden wurde, wird dann in das Assemblerprogramm an dieser Stelle eingefuegt und in ueblicher Weise verarbeitet. Sind alle Assemblerquellanweisungen in dieser Datei assembliert, wird die Verarbeitung mit der Assemblerquellanweisung beendet, die dem Makrobzug unmittelbar folgt.
Um Veraenderungen bei jedem Gebrauch eines Makros zu erlauben, wird ein Makro-Argument vorgesehen. Die in der Assemblerquellanweisung enthaltene Makrodefinition kann eine Vielzahl formaler Makroparameter enthalten. Wenn der Assembler im Rahmen einer Makroausdehnung eine Makro-Argumentbegrenzung erkennt (z.B. ein Anfuehrungszeichen '"'), wird der Ausdruck zwischen dem ersten und dem folgenden Argumentbegrenzer berechnet. Der Wert dieses Ausdrucks bestimmt dann, welche aktuelle Makro-Parameterzeichenkette an dieser Stelle in der Assemblerquellanweisung eingefuegt wird. Ein Wert von 1 bedeutet der erste aktuelle Makro-Parameter, der Wert 2 der zweite usw. Ein Wert 0 kann auch benutzt werden und das zeigt an, dass die Zahl der aktuellen Makroparameter in der zum Makrobezug gehoerenden Parameterliste, die gerade in Bearbeitung befindlichen Makro-Ausdehnung aufgerufen hat.
Bei Aufruf eines Makros werden die aktuellen Makroparameter an dieser Stelle durch die entsprechenden formalen Makroparameter ersetzt. Zwei Arten der Parameteruebergabe sind moeglich: Eine wortgetreue und eine wertgetreue Substitution. Bei der wortgetreuen Parameteruebergabe ist jede zulaessige Zeichenkette als Makroparameter moeglich. Die einzigsten Ausnahmen sind Begrenzungsstriche und linke runde Klammern. Begrenzungsstriche werden zur Kennzeichnung des Endes der Parameterliste verwendet. Kommas werden zur Trennung der Parameter benutzt; runde Klammern dienen zum Einschliessen der Parameter zur wertgetreuen Substitution. Ist ein aktueller Makroparameter in Klammern gesetzt, erfolgt seine Berechnung und eine Zeichenkette, die das Aequivalent des Resultates darstellt, wird als aktueller Parameter verwendet.

Da ein Makro nach Definition nur aus einer Folge von Assemblersprachen-Quellanweisungen besteht, sind mehrfache Niveaus von Makrobezuegen erlaubt und wiederkehrende Makros moeglich. Es sollte jedoch beachtet werden, dass bei allen wiederkehrenden Makrodefinitionen zu sichern ist, dass die zugehoerigen eigenen bedingten Assemblerblocks enthalten sind, um einen eventuellen Austritt aus einer Reihe von wiederkehrenden Makrobezuegen zu verhindern.

Beispiele:

Definition makro2 - abgelegt in einer separaten Datei

```
    a:    equ      "0"       ; Anzahl der Parameter
    b:    equ      "1"       ; erster formaler Parameter
    c:    equ      "2"       ; zweiter formaler Parameter
```

Makrobezug

```
    a1:   equ      10
          macro2   1,(a1)
          end
```

erstelltes Listing

```
                1 000a    1 a1:    equ     10
  makro2        2 0002    1  a:    equ     2
  makro2        3 0001    2  b:    equ     1
  makro2        4 000a    3  x:    equ     10
    0  000000   5         3        end
```

Definition makro3 - abgelegt in einer separaten Datei

```
        ld r8,"2" *4
        ld r9,"1"
```

Makrobezug

```
        cat:    db      2
        cmda:   equ     0aaaah
        start:  macro3  cat,2
                end
```

erstelltes Listing

```
        0 000000    1 02 00    1 cat:    dw     2
                    2 aa aa    2 cmda:   equ    0aaaah
macro3  0 000002    3 61 08    0 ld      hl,2*4
                      00 08
macro3  0 000006    4 61 09    1 ld      hl,cat
                      00 00
        0 00000a    5          5         end
```

Anhang A

Zusammenfassung Befehlssatz

```
                1                1   ; **** comment ****
                2                2           begin    1
                3                3           common   nn,02h
                4                4           extrn    extda
                5                5           public   index
                6 05             6   index:  equ      5
                7 20             7   n:      equu     20h
                8 02 30          8   dis:    equ      230h
                9 01 80          9   djdis:  def      180h
1  0000        10 01            10   label:  db       01h
1  0001        11 23 00         11           dw       023h
1  0003        12 61 62         12           text     'abcd'
1  0005           63 64
                13               13   ; **** code section ****
                14               14           begin    0
                15               15           if       1
0  0000        16 8e            16           adc      a,(hl)
0  0001        17 dd 8e 05      17           adc      a,(ix+index)
0  0004        18 fd 8e 05      18           adc      a,(iy+index)
                19               19           endif
0  0007        20 8f            20           adc      a,a
0  0008        21 88            21           adc      a,b
0  0009        22 89            22           adc      a,c
0  000a        23 8a            23           adc      a,d
0  000b        24 8b            24           adc      a,e
0  000c        25 8c            25           adc      a,h
0  000d        26 8d            26           adc      a,l
0  000e        27 ce 20         27           adc      a,n
0  0010        28 ed 4a         28           adc      hl,bc
0  0012        29 ed 5a         29           adc      hl,de
0  0014        30 ed 6a         30           adc      hl,hl
0  0016        31 ed 7a         31   adc     hl,sp
0  0018        32 86            32           add      a,(hl)
0  0019        33 dd 86 05      33           add      a,(ix+index)
0  001c        34 fd 86 05      34           add      a,(iy+index)
0  001f        35 87            35           add      a,a
0  0020        36 80            36           add      a,b
0  0021        37 81            37           add      a,c
0  0022        38 82            38           add      a,d
0  0023        39 83            39           add      a,e
0  0024        40 84            40           add      a,h
0  0025        41 85            41           add      a,l
0  0026        42 c6 20         42           add      a,n
0  0028        43 09            43           add      hl,bc
0  0029        44 19            44           add      hl,de
0  002a        45 29            45           add      hl,hl
0  002b        46 39            46           add      hl,sp
0  002c        47 dd 09         47           add      ix,bc
0  002e        48 dd 19         48           add      ix,de
0  0030        49 dd 29         49           add      ix,ix
0  0032        50 dd 39         50           add      ix,sp
0  0034        51 fd 09         51           add      iy,bc
0  0036        52 fd 19         52           add      iy,de
```

```
0  0038    53 fd 29        53          add     iy,iy
0  003a    54 fd 39        54          add     iy,sp
0  003c    55 a6           55          and     (hl)
0  003d    56 dd a6 05     56          and     (ix+index)
0  0040    57 fd a6 05     57          and     (iy+index)
0  0043    58 a7           58          and     a
0  0044    59 a0           59          and     b
0  0045    60 a1           60          and     c
0  0046    61 a2           61          and     d
0  0047    62 a3           62          and     e
0  0048    63 a4           63          and     h
0  0049    64 a5           64          and     l
0  004a    65 e6 20        65          and     n
0  004c    66 cb 46        66          bit     0,(hl)
0  004e    67 dd cb 05     67          bit     0,(ix+index)
                46
0  0052    68 fd cb 05     68          bit     0,(iy+index)
                46
0  0056    69 cb 47        69          bit     0,a
0  0058    70 cb 40        70          bit     0,b
0  005a    71 cb 41        71          bit     0,c
0  005c    72 cb 42        72          bit     0,d
0  005e    73 cb 43        73          bit     0,e
0  0060    74 cb 44        74          bit     0,h
0  0062    75 cb 45        75          bit     0,l
0  0064    76 cb 4e        76          bit     1,(hl)
0  0066    77 dd cb 05     77          bit     1,(ix+index)
                4e
0  006a    78 fd cb 05     78          bit     1,(iy+index)
                4e
0  006e    79 cb 4f        79          bit     1,a
0  0070    80 cb 48        80          bit     1,b
0  0072    81 cb 49        81          bit     1,c
0  0074    82 cb 4a        82          bit     1,d
0  0076    83 cb 4b        83          bit     1,e
0  0078    84 cb 4c        84          bit     1,h
0  007a    85 cb 4d        85          bit     1,l
0  007c    86 cb 56        86          bit     2,(hl)
0  007e    87 dd cb 05     87          bit     2,(ix+index)
                56
0  0082    88 fd cb 05     88          bit     2,(iy+index)
                56
0  0086    89 cb 57        89          bit     2,a
0  0088    90 cb 50        90          bit     2,b
0  008a    91 cb 51        91          bit     2,c
0  008c    92 cb 52        92          bit     2,d
0  008e    93 cb 53        93          bit     2,e
0  0090    94 cb 54        94          bit     2,h
0  0092    95 cb 55        95          bit     2,l
0  0094    96 cb 5e        96          bit     3,(hl)
0  0096    97 dd cb 05     97          bit     3,(ix+index)
                5e
0  009a    98 fd cb 05     98          bit     3,(iy+index)
                5e
0  009e    99 cb 5f        99          bit     3,a
0  00a0   100 cb 58       100          bit     3,b
0  00a2   101 cb 59       101          bit     3,c
0  00a4   102 cb 5a       102          bit     3,d
```

```
0  00a6   103 cb 5b       103          bit      3,e
0  00a8   104 cb 5c       104          bit      3,h
0  00aa   105 cb 5d       105          bit      3,l
0  00ac   106 cb 66       106          bit      4,(hl)
0  00ae   107 dd cb 05    107          bit      4,(ix+index)
0  00b2   108 fd cb 05    108          bit      4,(iy+index)
                  66
0  00b6   109 cb 67       109          bit      4,a
0  00b8   110 cb 60       110          bit      4,b
0  00ba   111 cb 61       111          bit      4,c
0  00bc   112 cb 62       112          bit      4,d
0  00be   113 cb 63       113          bit      4,e
0  00c0   114 cb 64       114          bit      4,h
0  00c2   115 cb 65       115          bit      4,l
0  00c4   116 cb 6e       116          bit      5,(hl)
0  00c6   117 dd cb 05    117          bit      5,(ix+index)
                  6e
0  00ca   118 fd cb 05    118          bit      5,(iy+index)
                  6e
0  00ce   119 cb 6f       119          bit      5,a
0  00d0   120 cb 68       120          bit      5,b
0  00d2   121 cb 69       121          bit      5,c
0  00d4   122 cb 6a       122          bit      5,d
0  00d6   123 cb 6b       123          bit      5,e
0  00d8   124 cb 6c       124          bit      5,h
0  00da   125 cb 6d       125          bit      5,l
0  00dc   126 cb 76       126          bit      6,(hl)
0  00de   127 dd cb 05    127          bit      6,(ix+index)
                  76
0  00e2   128 fd cb 05    128          bit      6,(iy+index)
                  76
0  00e6   129 cb 77       129          bit      6,a
0  00e8   130 cb 70       130          bit      6,b
0  00ea   131 cb 71       131          bit      6,c
0  00ec   132 cb 72       132          bit      6,d
0  00ee   133 cb 73       133          bit      6,e
0  00f0   134 cb 74       134          bit      6,h
0  00f2   135 cb 75       135          bit      6,l
0  00f4   136 cb 7e       136          bit      7,(hl)
0  00f6   137 dd cb 05    137          bit      7,(ix+index)
                  7e
0  00fa   138 fd cb 05    138          bit      7,(iy+index)
                  7e
0  00fe   139 cb 7f       139          bit      7,a
0  0100   140 cb 78       140          bit      7,b
0  0102   141 cb 79       141          bit      7,c
0  0104   142 cb 7a       142          bit      7,d
0  0106   143 cb 7b       143          bit      7,e
0  0108   144 cb 7c       144          bit      7,h
0  010a   145 cb 7d       145          bit      7,l
0  010c   146 dc 00 00    146          call     c,nn
0  010f   147 fc 00 00    147          call     m,nn
0  0112   148 d4 00 00    148          call     nc,nn
0  0115   149 cd 00 00    149          call     nn
0  0118   150 c4 00 00    150          call     nz,nn
0  011b   151 f4 00 00    151          call     p,nn
0  011e   152 ec 00 00    152          call     pe,nn
0  0121   153 e4 00 00    153          call     po,nn
```

```
0  0124   154 cc 00 00   154          call     z,nn
0  0127   155 3f         155          ccf
0  0128   156 be         156          cp       (hl)
0  0129   157 dd be 05   157          cp       (ix+index)
0  012c   158 fd be 05   158          cp       (iy+index)
0  012f   159 bf         159          cp       a
0  0130   160 b8         160          cp       b
0  0131   161 b9         161          cp       c
0  0132   162 ba         162          cp       d
0  0133   163 bb         163          cp       e
0  0134   164 bc         164          cp       h
0  0135   165 bd         165          cp       l
0  0136   166 fe 20      166          cp       n
0  0138   167 ed a9      167          cpd
0  013a   168 ed b9      168          cpdr
0  013c   169 ed a1      169          cpi
0  013e   170 ed b1      170          cpir
0  0140   171 2f         171          cpl
0  0141   172 27         172          daa
0  0142   173 35         173          dec      (hl)
0  0143   174 dd 35 05   174          dec      (ix+index)
0  0146   175 fd 35 05   175          dec      (iy+index)
0  0149   176 3d         176          dec      a
0  014a   177 05         177          dec      b
0  014b   178 0b         178          dec      bc
0  014c   179 0d         179          dec      c
0  014d   180 15         180          dec      d
0  014e   181 1b         181          dec      de
0  014f   182 1d         182          dec      e
0  0150   183 25         183          dec      h
0  0151   184 2b         184          dec      hl
0  0152   185 dd 2b      185          dec      ix
0  0154   186 fd 2b      186          dec      iy
0  0156   187 2d         187          dec      l
0  0157   188 3b         188          dec      sp
0  0158   189 f3         189          di
0  0159   190 10 25      190          djnz     djdis
0  015b   191 fb         191          ei
0  015c   192 e3         192          ex       (sp),hl
0  015d   193 dd e3      193          ex       (sp),ix
0  015f   194 fd e3      194          ex       (sp),iy
0  0161   195 08         195          ex       af,af
0  0162   196 eb         196          ex       de,hl
0  0163   197 d9         197          exx
0  0164   198 76         198          halt
0  0165   199 ed 46      199          im       0
0  0167   200 ed 56      200          im       1
0  0169   201 ed 5e      201          im       2
0  016b   202 ed 78      202          in       a,(c)
0  016d   203 db 20      203          in       a,(n)
0  016f   204 ed 40      204          in       b,(c)
0  0171   205 ed 48      205          in       c,(c)
0  0173   206 ed 50      206          in       d,(c)
0  0175   207 ed 58      207          in       e,(c)
0  0177   208 ed 60      208          in       h,(c)
0  0179   209 ed 68      209          in       l,(c)
0  017b   210 34         210          inc      (hl)
0  017c   211 dd 34 05   211          inc      (ix+index)
```

```
0  017f   212 fd 34 05   212          inc      (iy+index)
0  0182   213 3c         213          inc      a
0  0183   214 04         214          inc      b
0  0184   215 03         215          inc      bc
0  0185   216 0c         216          inc      c
0  0186   217 14         217          inc      d
0  0187   218 13         218          inc      de
0  0188   219 1c         219          inc      e
0  0189   220 24         220          inc      h
0  018a   221 23         221          inc      hl
0  018b   222 dd 23      222          inc      ix
0  018d   223 fd 23      223          inc      iy
0  018f   224 2c         224          inc      l
0  0190   225 33         225          inc      sp
0  0191   226 ed aa      226          ind
0  0193   227 ed ba      227          indr
0  0195   228 ed a2      228          ini
0  0197   229 ed b2      229          inir
0  0199   230 e9         230          jp       (hl)
0  019a   231 dd e9      231          jp       (ix)
0  019c   232 fd e9      232          jp       (iy)
0  019e   233 da 00 00   233          jp       c,nn
0  01a1   234 fa 00 00   234          jp       m,nn
0  01a4   235 d2 00 00   235          jp       nc,nn
0  01a7   236 c3 00 00   236          jp       nn
0  01aa   237 c2 00 00   237          jp       nz,nn
0  01ad   238 f2 00 00   238          jp       p,nn
0  01b0   239 ea 00 00   239          jp       pe,nn
0  01b3   240 e2 00 00   240          jp       po,nn
0  01b6   241 ca 00 00   241          jp       z,nn
0  01b9   242 38 75      242          jr       c,dis
0  01bb   243 18 73      243          jr       dis
0  01bd   244 30 71      244          jr       nc,dis
0  01bf   245 20 6f      245          jr       nz,dis
0  01c1   246 28 6d      246          jr       z,dis
0  01c3   247 02         247          ld       (bc),a
0  01c4   248 12         248          ld       (de),a
0  01c5   249 77         249          ld       (hl),a
0  01c6   250 70         250          ld       (hl),b
0  01c7   251 71         251          ld       (hl),c
0  01c8   252 72         252          ld       (hl),d
0  01c9   253 73         253          ld       (hl),e
0  01ca   254 74         254          ld       (hl),h
0  01cb   255 75         255          ld       (hl),l
0  01cc   256 36 20      256          ld       (hl),n
0  01ce   257 dd 77 05   257          ld       (ix+index),a
0  01d1   258 dd 70 05   258          ld       (ix+index),b
0  01d4   259 dd 71 05   259          ld       (ix+index),c
0  01d7   260 dd 72 05   260          ld       (ix+index),d
0  01da   261 dd 73 05   261          ld       (ix+index),e
0  01dd   262 dd 74 05   262          ld       (ix+index),h
0  01e0   263 dd 75 05   263          ld       (ix+index),l
0  01e3   264 dd 36 05   264          ld       (ix+index),n
                20
0  01e7   265 fd 77 05   265          ld       (iy+index),a
0  01ea   266 fd 70 05   266          ld       (iy+index),b
0  01ed   267 fd 71 05   267          ld       (iy+index),c
0  01f0   268 fd 72 05   268          ld       (iy+index),d
```

```
0  01f3   269 fd 73 05   269          ld       (iy+index),e
0  01f6   270 fd 74 05   270          ld       (iy+index),h
0  01f9   271 fd 75 05   271          ld       (iy+index),l
0  01fc   272 fd 36 05   272          ld       (iy+index),n
                 20
0  0200   273 32 00 00   273          ld       (nn),a
0  0203   274 ed 43 00   274          ld       (nn),bc
                 00
0  0207   275 ed 53 00   275          ld       (nn),de
                 00
0  020b   276 22 00 00   276          ld       (nn),hl
0  020e   277 dd 22 00   277          ld       (nn),ix
                 00
0  0212   278 fd 22 00   278          ld       (nn),iy
                 00
0  0216   279 ed 73 00   279          ld       (nn),sp
                 00
0  021a   280 0a         280          ld       a,(bc)
0  021b   281 1a         281          ld       a,(de)
0  021c   282 7e         282          ld       a,(hl)
0  021d   283 dd 7e 05   283          ld       a,(ix+index)
0  0220   284 fd 7e 05   284          ld       a,(iy+index)
0  0223   285 3a 00 00   285          ld       a,(nn)
0  0226   286 7f         286          ld       a,a
0  0227   287 78         287          ld       a,b
0  0228   288 79         288          ld       a,c
0  0229   289 7a         289          ld       a,d
0  022a   290 7b         290          ld       a,e
0  022b   291 7c         291          ld       a,h
0  022c   292 ed 57      292          ld       a,i
0  022e   293 7d         293          ld       a,l
0  022f   294 3e 20      294          ld       a,n
0  0231   295 ed 5f      295          ld       a,r
0  0233   296 46         296          ld       b,(hl)
0  0234   297 dd 46 05   297          ld       b,(ix+index)
0  0237   298 fd 46 05   298          ld       b,(iy+index)
0  023a   299 47         299          ld       b,a
0  023b   300 40         300          ld       b,b
0  023c   301 41         301          ld       b,c
0  023d   302 42         302          ld       b,d
0  023e   303 43         303          ld       b,e
0  023f   304 44         304          ld       b,h
0  0240   305 45         305          ld       b,l
0  0241   306 06 20      306          ld       b,n
0  0243   307 ed 4b 00   307          ld       bc,(nn)
                 00
0  0247   308 01 00 00   308          ld       bc,nn
0  024a   309 4e         309          ld       c,(hl)
0  024b   310 dd 4e 05   310          ld       c,(ix+index)
0  024e   311 fd 4e 05   311          ld       c,(iy+index)
0  0251   312 4f         312          ld       c,a
0  0252   313 48         313          ld       c,b
0  0253   314 49         314          ld       c,c
0  0254   315 4a         315          ld       c,d
0  0255   316 4b         316          ld       c,e
0  0256   317 4c         317          ld       c,h
0  0257   318 4d         318          ld       c,l
0  0258   319 0e 20      319          ld       c,n
```

```
0  025a   320 56          320          ld       d,(hl)
0  025b   321 dd 56 05     321          ld       d,(ix+index)
0  025e   322 fd 56 05     322          ld       d,(iy+index)
0  0261   323 57           323          ld       d,a
0  0262   324 50           324          ld       d,b
0  0263   325 51           325          ld       d,c
0  0264   326 52           326          ld       d,d
0  0265   327 53           327          ld       d,e
0  0266   328 5e           328          ld       e,(hl)
0  0267   329 dd 5e 05     329          ld       e,(ix+index)
0  026a   330 fd 5e 05     330          ld       e,(iy+index)
0  026d   331 5f           331          ld       e,a
0  026e   332 58           332          ld       e,b
0  026f   333 59           333          ld       e,c
0  0270   334 5a           334          ld       e,d
0  0271   335 5b           335          ld       e,e
0  0272   336 5c           336          ld       e,h
0  0273   337 5d           337          ld       e,l
0  0274   338 1e 20        338          ld       e,n
0  0276   339 66           339          ld       h,(hl)
0  0277   340 dd 66 05     340          ld       h,(ix+index)
0  027a   341 fd 66 05     341          ld       h,(iy+index)
0  027d   342 67           342          ld       h,a
0  027e   343 60           343          ld       h,b
0  027f   344 61           344          ld       h,c
0  0280   345 62           345          ld       h,d
0  0281   346 63           346          ld       h,e
0  0282   347 64           347          ld       h,h
0  0283   348 65           348          ld       h,l
0  0284   349 26 20        349          ld       h,n
0  0286   350 2a 00 00     350          ld       hl,(nn)
0  0289   351 21 00 00     351          ld       hl,nn
0  028c   352 ed 47        352          ld       i,a
0  028e   353 dd 2a 00     353          ld       ix,(nn)
                  00
0  0292   354 dd 21 00     354          ld       ix,nn
                  00
0  0296   355 fd 2a 00     355          ld       iy,(nn)
                  00
0  029a   356 fd 21 00     356          ld       iy,nn
                  00
0  029e   357 6e           357          ld       l,(hl)
0  029f   358 dd 6e 05     358          ld       l,(ix+index)
0  02a2   359 fd 6e 05     359          ld       l,(iy+index)
0  02a5   360 6f           360          ld       l,a
0  02a6   361 68           361          ld       l,b
0  02a7   362 69           362          ld       l,c
0  02a8   363 6a           363          ld       l,d
0  02a9   364 6b           364          ld       l,e
0  02aa   365 6c           365          ld       l,h
0  02ab   366 6d           366          ld       l,l
0  02ac   367 2e 20        367          ld       l,n
0  02ae   368 ed 4f        368          ld       r,a
0  02b0   369 ed 7b 00     369          ld       sp,(nn)
                  00
0  02b4   370 f9           370          ld       sp,hl
0  02b5   371 dd f9        371          ld       sp,ix
0  02b7   372 fd f9        372          ld       sp,iy
```

```
0  02b9   373 31 00 00   373          ld       sp,nn
0  02bc   374 ed a8      374          ldd
0  02be   375 ed b8      375          lddr
0  02c0   376 ed a0      376          ldi
0  02c2   377 ed b0      377          ldir
0  02c4   378 ed 44      378          neg
0  02c6   379 00         379          nop
0  02c7   380 b6         380          or       (hl)
0  02c8   381 dd b6 05   381          or       (ix+index)
0  02cb   382 fd b6 05   382          or       (iy+index)
0  02ce   383 b7         383          or       a
0  02cf   384 b0         384          or       b
0  02d0   385 b1         385          or       c
0  02d1   386 b2         386          or       d
0  02d2   387 b3         387          or       e
0  02d3   388 b4         388          or       h
0  02d4   389 b5         389          or       l
0  02d5   390 f6 20      390          or       n
0  02d7   391 ed bb      391          otdr
0  02d9   392 ed b3      392          otir
0  02db   393 ed 79      393          out      (c),a
0  02dd   394 ed 41      394          out      (c),b
0  02df   395 ed 49      395          out      (c),c
0  02e1   396 ed 51      396          out      (c),d
0  02e3   397 ed 59      397          out      (c),e
0  02e5   398 ed 61      398          out      (c),h
0  02e7   399 ed 69      399          out      (c),l
0  02e9   400 d3 20      400          out      (n),a
0  02eb   401 ed ab      401          outd
0  02ed   402 ed a3      402          outi
0  02ef   403 f1         403          pop      af
0  02f0   404 c1         404          pop      bc
0  02f1   405 d1         405          pop      de
0  02f2   406 e1         406          pop      hl
0  02f3   407 dd e1      407          pop      ix
0  02f5   408 fd e1      408          pop      iy
0  02f7   409 f5         409          push     af
0  02f8   410 c5         410          push     bc
0  02f9   411 d5         411          push     de
0  02fa   412 e5         412          push     hl
0  02fb   413 dd e5      413          push     ix
0  02fd   414 fd e5      414          push     iy
0  02ff   415 cb 86      415          res      0,(hl)
0  0301   416 dd cb 05   416          res      0,(ix+index)
                 86
0  0305   417 fd cb 05   417          res      0,(iy+index)
                 86
0  0309   418 cb 87      418          res      0,a
0  030b   419 cb 80      419          res      0,b
0  030d   420 cb 81      420          res      0,c
0  030f   421 cb 82      421          res      0,d
0  0311   422 cb 83      422          res      0,e
0  0313   423 cb 84      423          res      0,h
0  0315   424 cb 85      424          res      0,l
0  0317   425 cb 8e      425          res      1,(hl)
0  0319   426 dd cb 05   426          res      1,(ix+index)
                 8e
0  031d   427 fd cb 05   427          res      1,(iy+index)
```

```
                    8e
0  0321   428 cb 8f      428          res     1,a
0  0323   429 cb 88      429          res     1,b
0  0325   430 cb 89      430          res     1,c
0  0327   431 cb 8a      431          res     1,d
0  0329   432 cb 8b      432          res     1,e
0  032b   433 cb 8c      433          res     1,h
0  032d   434 cb 8d      434          res     1,l
0  032f   435 cb 96      435          res     2,(hl)
0  0331   436 dd cb 05   436          res     2,(ix+index)
                    96
0  0335   437 fd cb 05   437          res     2,(iy+index)
                    96
0  0339   438 cb 97      438          res     2,a
0  033b   439 cb 90      439          res     2,b
0  033d   440 cb 91      440          res     2,c
0  033f   441 cb 92      441          res     2,d
0  0341   442 cb 93      442          res     2,e
0  0343   443 cb 94      443          res     2,h
0  0345   444 cb 95      444          res     2,l
0  0347   445 cb 9e      445          res     3,(hl)
0  0349   446 dd cb 05   446          res     3,(ix+index)
                    9e
0  034d   447 fd cb 05   447          res     3,(iy+index)
                    9e
0  0351   448 cb 9f      448          res     3,a
0  0353   449 cb 98      449          res     3,b
0  0355   450 cb 99      450          res     3,c
0  0357   451 cb 9a      451          res     3,d
0  0359   452 cb 9b      452          res     3,e
0  035b   453 cb 9c      453          res     3,h
0  035d   454 cb 9d      454          res     3,l
0  035f   455 cb a6      455          res     4,(hl)
0  0361   456 dd cb 05   456          res     4,(ix+index)
                    a6
0  0365   457 fd cb 05   457          res     4,(iy+index)
                    a6
0  0369   458 cb a7      458          res     4,a
0  036b   459 cb a0      459          res     4,b
0  036d   460 cb a1      460          res     4,c
0  036f   461 cb a2      461          res     4,d
0  0371   462 cb a3      462          res     4,e
0  0373   463 cb a4      463          res     4,h
0  0375   464 cb a5      464          res     4,l
0  0377   465 cb ae      465          res     5,(hl)
0  0379   466 dd cb 05   466          res     5,(ix+index)
                    ae
0  037d   467 fd cb 05   467          res     5,(iy+index)
                    ae
0  0381   468 cb af      468          res     5,a
0  0383   469 cb a8      469          res     5,b
0  0385   470 cb a9      470          res     5,c
0  0387   471 cb aa      471          res     5,d
0  0389   472 cb ab      472          res     5,e
0  038b   473 cb ac      473          res     5,h
0  038d   474 cb ad      474          res     5,l
0  038f   475 cb b6      475          res     6,(hl)
0  0391   476 dd cb 05   476          res     6,(ix+index)
```

```
                    b6
0  0395   477 fd cb 05   477          res      6,(iy+index)
                    b6
0  0399   478 cb b7      478          res      6,a
0  039b   479 cb b0      479          res      6,b
0  039d   480 cb b1      480          res      6,c
0  039f   481 cb b2      481          res      6,d
0  03a1   482 cb b3      482          res      6,e
0  03a3   483 cb b4      483          res      6,h
0  03a5   484 cb b5      484          res      6,l
0  03a7   485 cb be      485          res      7,(hl)
0  03a9   486 dd cb 05   486          res      7,(ix+index)
                    be
0  03ad   487 fd cb 05   487          res      7,(iy+index)
                    be
0  03b1   488 cb bf      488          res      7,a
0  03b3   489 cb b8      489          res      7,b
0  03b5   490 cb b9      490          res      7,c
0  03b7   491 cb ba      491          res      7,d
0  03b9   492 cb bb      492          res      7,e
0  03bb   493 cb bc      493          res      7,h
0  03bd   494 cb bd      494          res      7,l
0  03bf   495 c9         495          ret
0  03c0   496 d8         496          ret      c
0  03c1   497 f8         497          ret      m
0  03c2   498 d0         498          ret      nc
0  03c3   499 c0         499          ret      nz
0  03c4   500 f0         500          ret      p
0  03c5   501 e8         501          ret      pe
0  03c6   502 e0         502          ret      po
0  03c7   503 c8         503          ret      z
0  03c8   504 ed 4d      504          reti
0  03ca   505 ed 45      505          retn
0  03cc   506 cb 16      506          rl       (hl)
0  03ce   507 dd cb 05   507          rl       (ix+index)
                    16
0  03d2   508 fd cb 05   508          rl       (iy+index)
                    16
0  03d6   509 cb 17      509          rl       a
0  03d8   510 cb 10      510          rl       b
0  03da   511 cb 11      511          rl       c
0  03dc   512 cb 12      512          rl       d
0  03de   513 cb 13      513          rl       e
0  03e0   514 cb 14      514          rl       h
0  03e2   515 cb 15      515          rl       l
0  03e4   516 17         516          rla
0  03e5   517 cb 06      517          rlc      (hl)
0  03e7   518 dd cb 05   518          rlc      (ix+index)
                    06
0  03eb   519 fd cb 05   519          rlc      (iy+index)
                    06
0  03ef   520 cb 07      520          rlc      a
0  03f1   521 cb 00      521          rlc      b
0  03f3   522 cb 01      522          rlc      c
0  03f5   523 cb 02      523          rlc      d
0  03f7   524 cb 03      524          rlc      e
0  03f9   525 cb 04      525          rlc      h
0  03fb   526 cb 05      526          rlc      l
```

```
0  03fd   527 07          527          rlca
0  03fe   528 ed 6f       528          rld
0  0400   529 cb 1e       529          rr       (hl)
0  0402   530 dd cb 05     530          rr       (ix+index)
                1e
0  0406   531 fd cb 05     531          rr       (iy+index)
                1e
0  040a   532 cb 1f       532          rr       a
0  040c   533 cb 18       533          rr       b
0  040e   534 cb 19       534          rr       c
0  0410   535 cb 1a       535          rr       d
0  0412   536 cb 1b       536          rr       e
0  0414   537 cb 1c       537          rr       h
0  0416   538 cb 1d       538          rr       l
0  0418   539 1f          539          rra
0  0419   540 cb 0e       540          rrc      (hl)
0  041b   541 dd cb 05     541          rrc      (ix+index)
                0e
0  041f   542 fd cb 05     542          rrc      (iy+index)
                0e
0  0423   543 cb 0f       543          rrc      a
0  0425   544 cb 08       544          rrc      b
0  0427   545 cb 09       545          rrc      c
0  0429   546 cb 0a       546          rrc      d
0  042b   547 cb 0b       547          rrc      e
0  042d   548 cb 0c       548          rrc      h
0  042f   549 cb 0d       549          rrc      l
0  0431   550 0f          550          rrca
0  0432   551 ed 67       551          rrd
0  0434   552 c7          552          rst      0
0  0435   553 cf          553          rst      08h
0  0436   554 d7          554          rst      10h
0  0437   555 df          555          rst      18h
0  0438   556 e7          556          rst      20h
0  0439   557 ef          557          rst      28h
0  043a   558 f7          558          rst      30h
0  043b   559 ff          559          rst      38h
0  043c   560 9e          560          sbc      a,(hl)
0  043d   561 dd 9e 05     561          sbc      a,(ix+index)
0  0440   562 fd 9e 05     562          sbc      a,(iy+index)
0  0443   563 9f          563          sbc      a,a
0  0444   564 98          564          sbc      a,b
0  0445   565 99          565          sbc      a,c
0  0446   566 9a          566          sbc      a,d
0  0447   567 9b          567          sbc      a,e
0  0448   568 9c          568          sbc      a,h
0  0449   569 9d          569          sbc      a,l
0  044a   570 de 20       570          sbc      a,n
0  044c   571 ed 42       571          sbc      hl,bc
0  044e   572 ed 52       572          sbc      hl,de
0  0450   573 ed 62       573          sbc      hl,hl
0  0452   574 ed 72       574          sbc      hl,sp
0  0454   575 37          575          scf
0  0455   576 cb c6       576          set      0,(hl)
0  0457   577 dd cb 05     577          set      0,(ix+index)
                c6
0  045b   578 fd cb 05     578          set      0,(iy+index)
                c6
```

```
0  045f   579 cb c7      579          set     0,a
0  0461   580 cb c0      580          set     0,b
0  0463   581 cb c1      581          set     0,c
0  0465   582 cb c2      582          set     0,d
0  0467   583 cb c3      583          set     0,e
0  0469   584 cb c4      584          set     0,h
0  046b   585 cb c5      585          set     0,l
0  046d   586 cb ce      586          set     1,(hl)
0  046f   587 dd cb 05   587          set     1,(ix+index)
                ce
0  0473   588 fd cb 05   588          set     1,(iy+index)
                ce
0  0477   589 cb cf      589          set     1,a
0  0479   590 cb c8      590          set     1,b
0  047b   591 cb c9      591          set     1,c
0  047d   592 cb ca      592          set     1,d
0  047f   593 cb cb      593          set     1,e
0  0481   594 cb cc      594          set     1,h
0  0483   595 cb cd      595          set     1,l
0  0485   596 cb d6      596          set     2,(hl)
0  0487   597 dd cb 05   597          set     2,(ix+index)
                d6
0  048b   598 fd cb 05   598          set     2,(iy+index)
                d6
0  048f   599 cb d7      599          set     2,a
0  0491   600 cb d0      600          set     2,b
0  0493   601 cb d1      601          set     2,c
0  0495   602 cb d2      602          set     2,d
0  0497   603 cb d3      603          set     2,e
0  0499   604 cb d4      604          set     2,h
0  049b   605 cb d5      605          set     2,l
0  049d   606 cb de      606          set     3,(hl)
0  049f   607 dd cb 05   607          set     3,(ix+index)
                de
0  04a3   608 fd cb 05   608          set     3,(iy+index)
                de
0  04a7   609 cb df      609          set     3,a
0  04a9   610 cb d8      610          set     3,b
0  04ab   611 cb d9      611          set     3,c
0  04ad   612 cb da      612          set     3,d
0  04af   613 cb db      613          set     3,e
0  04b1   614 cb dc      614          set     3,h
0  04b3   615 cb dd      615          set     3,l
0  04b5   616 cb e6      616          set     4,(hl)
0  04b7   617 dd cb 05   617          set     4,(ix+index)
                e6
0  04bb   618 fd cb 05   618          set     4,(iy+index)
                e6
0  04bf   619 cb e7      619          set     4,a
0  04c1   620 cb e0      620          set     4,b
0  04c3   621 cb e1      621          set     4,c
0  04c5   622 cb e2      622          set     4,d
0  04c7   623 cb e3      623          set     4,e
0  04c9   624 cb e4      624          set     4,h
0  04cb   625 cb e5      625          set     4,l
0  04cd   626 cb ee      626          set     5,(hl)
0  04cf   627 dd cb 05   627          set     5,(ix+index)
                ee
```

```
0  04d3   628 fd cb 05   628          set     5,(iy+index)
               ee
0  04d7   629 cb ef      629          set     5,a
0  04d9   630 cb e8      630          set     5,b
0  04db   631 cb e9      631          set     5,c
0  04dd   632 cb ea      632          set     5,d
0  04df   633 cb eb      633          set     5,e
0  04e1   634 cb ec      634          set     5,h
0  04e3   635 cb ed      635          set     5,l
0  04e5   636 cb f6      636          set     6,(hl)
0  04e7   637 dd cb 05   637          set     6,(ix+index)
               f6
0  04eb   638 fd cb 05   638          set     6,(iy+index)
               f6
0  04ef   639 cb f7      639          set     6,a
0  04f1   640 cb f0      640          set     6,b
0  04f3   641 cb f1      641          set     6,c
0  04f5   642 cb f2      642          set     6,d
0  04f7   643 cb f3      643          set     6,e
0  04f9   644 cb f4      644          set     6,h
0  04fb   645 cb f5      645          set     6,l
0  04fd   646 cb fe      646          set     7,(hl)
0  04ff   647 dd cb 05   647          set     7,(ix+index)
               fe
0  0503   648 fd cb 05   648          set     7,(iy+index)
               fe
0  0507   649 cb ff      649          set     7,a
0  0509   650 cb f8      650          set     7,b
0  050b   651 cb f9      651          set     7,c
0  050d   652 cb fa      652          set     7,d
0  050f   653 cb fb      653          set     7,e
0  0511   654 cb fc      654          set     7,h
0  0513   655 cb fd      655          set     7,l
0  0515   656 cb 26      656          sla     (hl)
0  0517   657 dd cb 05   657          sla     (ix+index)
               26
0  051b   658 fd cb 05   658          sla     (iy+index)
               26
0  051f   659 cb 27      659          sla     a
0  0521   660 cb 20      660          sla     b
0  0523   661 cb 21      661          sla     c
0  0525   662 cb 22      662          sla     d
0  0527   663 cb 23      663          sla     e
0  0529   664 cb 24      664          sla     h
0  052b   665 cb 25      665          sla     l
0  052d   666 cb 2e      666          sra     (hl)
0  052f   667 dd cb 05   667          sra     (ix+index)
               2e
0  0533   668 fd cb 05   668          sra     (iy+index)
               2e
0  0537   669 cb 2f      669          sra     a
0  0539   670 cb 28      670          sra     b
0  053b   671 cb 29      671          sra     c
0  053d   672 cb 2a      672          sra     d
0  053f   673 cb 2b      673          sra     e
0  0541   674 cb 2c      674          sra     h
0  0543   675 cb 2d      675          sra     l
0  0545   676 cb 3e      676          srl     (hl)
```

```
0  0547   677 dd cb 05   677          srl     (ix+index)
              3e
0  054b   678 fd cb 05   678          srl     (iy+index)
              3e
0  054f   679 cb 3f      679          srl     a
0  0551   680 cb 38      680          srl     b
0  0553   681 cb 39      681          srl     c
0  0555   682 cb 3a      682          srl     d
0  0557   683 cb 3b      683          srl     e
0  0559   684 cb 3c      684          srl     h
0  055b   685 cb 3d      685          srl     l
0  055d   686 96         686          sub     (hl)
0  055e   687 dd 96 05   687          sub     (ix+index)
0  0561   688 fd 96 05   688          sub     (iy+index)
0  0564   689 97         689          sub     a
0  0565   690 90         690          sub     b
0  0566   691 91         691          sub     c
0  0567   692 92         692          sub     d
0  0568   693 93         693          sub     e
0  0569   694 94         694          sub     h
0  056a   695 95         695          sub     l
0  056b   696 d6 20      696          sub     n
0  056d   697 ae         697          xor     (hl)
0  056e   698 dd ae 05   698          xor     (ix+index)
0  0571   699 fd ae 05   699          xor     (iy+index)
0  0574   700 af         700          xor     a
0  0575   701 a8         701          xor     b
0  0576   702 a9         702          xor     c
0  0577   703 aa         703          xor     d
0  0578   704 ab         704          xor     e
0  0579   705 ac         705          xor     h
0  057a   706 ad         706          xor     l
0  057b   707 ee 20      707          xor     n
0  057d   708            708          end
```

Anhang B

Zusammenstellung der Unterschiede zwischen
"u80as" und "U880 ASM"

Man beachte:
Die bereits fuer UDOS geschriebene U880-Assemblersprache ist zum Teil verschieden und erfordert zur Assemblierung mit dem "u80as" einige Aenderungen.

U880 ASM (UDOS)	u80as (WEGA)
Anweisungssyntax: lab: opcode operand(s) ;comment	identisch
Alle Opkodes und Schluesselwoerter muessen entweder alle gross oder alle klein geschrieben sein	keine Fallunterscheidung
Marken werden unterschiedlich behandelt bei Gross-/Kleinschreibung	keine Fallunterscheidung
Zahlenmarkierung: B, O, H, Dezimal unmarkiert	identisch
Begrenzungssymbole: Kommas oder Leerzeichen	nur Kommas
Kommentare	identisch
Externe Marken: die ersten 6 Zeichen sind relevant	8 Zeichen relevant
Programmteilung: nicht implementiert	Lokalisationszaehler 0 fuer Kode, 1 fuer Daten
Assembler-Steueranweisungen:	
ORG nn	nicht implementiert
EQU nn	equ nn
DEFL nn	def nn
END	end
DEFT	nicht implementiert
EXTERNAL name	extrn name
GLOBAL name	public name
DEFB n	db n
DEFB 's'	nicht implementiert
DEFW nn	dw nn
DEFS nn	common name,nn
DEFM 's'	text 's'
MACRO ENDM	Makro sind in separaten Dateien def.

nicht implementiert	begin 0 (f. Kode) begin 1 (f. Daten)
Bedingte Pseudo-Operatoren:	
COND n	if n
nicht implementiert	else
ENDC	endif
Assemblerkommandos:	
*EJECT	eject
*HEADING 's'	title 's'
*LIST OFF	unlist
*LIST ON	list
*MACLIST OFF	nicht implementiert
*MACLIST ON	nicht implementiert
*INCLUDE dateiname	nicht implementiert
nicht implementiert	space nn
Arithmetische/logische Operatoren:	
+ (unaeres Plus)	+
- (unaeres Minus)	-
.NOT.	.not.
.RES.	nicht implementiert
** (Potenzrechnung)	nicht implementiert
* (Multiplikation)	*
/ (Division)	/
.MOD. (modulo)	mod.
.SHR. (log. Rechtsverschiebung)	.shr.
.SHL. (log. Linksverschiebung)	.shl.
+ (Addition)	+
- (Subtraktion)	-
.AND. oder &	.and.
.OR. oder |	.or.
.XOR. (log. XOR)	.xor.
.EQ. oder = (gleich)	=
.GT. oder > (groesser als)	>
.LT. oder < (kleiner als)	<
.UGT. (vorzeichenlos groesser als)	nicht implementiert
.ULT. (vorzeichenlos kleiner als)	nicht implementiert
nicht implementiert	>= (groesser als oder gleich)
nicht implementiert	<= (kleiner als oder gleich)

Anhang C

Fehlerausschriften

Der Assembler bietet eine umfangreiche Fehlersuche, vor allem bei der Syntax-Analyse. Bei Entdecken eines Fehlers wird die Zeilenabbildung angegeben und unter dieser eine Fehlernachricht angezeigt. Die Fehlernachricht besteht aus dem Fehlertyp, dem Fehlerschluessel und einer ausfuehrlichen Angabe, wo der Fehler entdeckt wurde. Die vom Assembler auffindbaren Fehler lassen sich in vier "Schweregrade" einteilen. Das Auffinden eines Fehlergrades ab mittlerem Fehlerniveau (d.h. mit Ausnahme von Warnungen) fuehrt zum Setzen des Fehler-Flag des erstellten verschiebbaren Programmodules.
Nachfolgend ist eine Liste aller auffindbaren Fehler und korrigierender Eingriffe (Handlung genannt) des Assemblers (sofern sie erfolgen) dargestellt. Auch am Ende jedes Assembler-Listings wird eine Zusammenstellung aller gefundenen Fehler ausgedruckt.

Fehlerwarnungen

(Beachte: Aktuelle Fehlernachrichten sind klein geschrieben)

01 = Ausschrift: DUPLICATE DEFINITION OF A SYMBOLIC
moegl.Grund: Der aktuelle Befehl bezieht sich auf ein Symbol, das doppelt vergeben wurde.
Handlung: Die Anfangsdefinition des Symbols wird benutzt.

02 = Ausschrift: OVERLY LONG SYMBOLIC
moegl.Grund: Eine externe Marke oder Einsprungpunkt ist laenger als 8 Zeichen.
Handlung: Der Assembler akzeptiert alle Zeichen, aber nur die ersten 8 Zeichen sind fuer den Linker "u80ld" relevant

03 = Ausschrift: RECEPTION OF DISALLOWED LABEL
moegl.Grund: Eine Marke wurde mit einer Anweisung gefunden, die keine sein darf.
Handlung: Die Marke wird ignoriert.

04 = Ausschrift: ABSENCE OF REQUIRED END DIRECTIVE
moegl.Grund: Die "end"-Steueranweisung fehlt im Assemblereingabestrom.
Handlung: Der Assembler verhaelt sich so, als sei die "end"-Anweisung die letzte Anweisung der Quelldatei.

24 = Ausschrift: POSSIBLE REMAINING SHORT TO LONG CONVERSIONS
moegl.Grund: Die Grenze des Zwischen-Assemblerlaufs wurde erreicht, ohne dass festgestellt wurde, das alle notwendigen Erweiterungen der Kurzform-Befehle ausgefuehrt

wurden.
Handlung: keine
Ist eine der notwendigen Erweiterungen nicht ausgefuehrt worden, werden die Kurzbefehle verwendet und diese auf Fehler untersucht.

mittlere Fehler
---

25 = Ausschrift: INVALID CHARACTER COMBINATION
moegl.Grund: Es ist dem Assembler nicht moeglich, bei der Syntax-Analyse eine Zeichenkette zu finden.
Handlung: Der Rest des aktuellen Feldes wird uebersprungen und das Einlesen beendet.

26 = Ausschrift: INVALID CONSTANT SPECIFIKATION
moegl.Grund: Eine Konstante wurde falsch spezifiziert
Handlung: Der Assembler versucht den Wert richtig-zustellen.

27 = Ausschrift: UNEVALUATABLE EXPRESSION
moegl.Grund: Der Assembler ist nicht in der Lage, einen angegebenen arithmetischen Ausdruck zu berechnen; in der Regel verursacht durch die Verwendung eines Uebersetzungszeit-Operators mit ungeeignetem Operand.
Handlung: Fuer den Ausdruck wird der Wert 0 angenommen
moegl.Grund: Ein Leerzeichen vor einer "(" oder der erste Operand in einem Zwei-Operanden-Feld.
Handlung: Das Leerzeichen ist zu beseitigen (z.B. r15(#6) und nicht r15 (#6))

28 = Ausschrift: USE OF UNDEFINED SYMBOLIC
moegl.Grund: Selbsterklaerend
Handlung: Der Wert des Symbols wird als 0 angenommen.

29 = Ausschrift: INVALID USE OF FORWARD REFERENCE
moegl.Grund: Eine symbolische Marke wurde verwendet bei einer Operation, die im ersten Pass durchgefuehrt wird, ohne dass ihr aktueller Wert bestimmt wurde. Das ist typisch bei der Anwendung der "equ"-Anweisung.
Handlung: Fuer das Symbol wird der Wert Null angenommen.

30 = Ausschrift: RELOCATION ERROR DETECTED IN EXPRESSION
moegl.Grund: Ein verschiebbares Element wurde in einem arithmet. Ausdruck in unzulaessiger Weise benutzt. Beispiel waere die Addition zweier verschiebbarer Elemente.

31 = Ausschrift: INVALID OPERAND FIELD
moegl.Grund: Zu viel numeriert
Handlung: Es wird ein Wert fuer den des fehlerbehafteten Unterfeldes generiert und die Assemblierung fortgesetzt.
Beachte: Dieser Fehler tritt primaer auf, wenn eine Anweisung zwar syntaktisch korrekt ist, aber semantisch fehlerhaft.

ernste Fehler
---

32 = Ausschrift: OMISSION OF A REQUIRED LABEL FIELD
moegl.Grund: selbsterklaerend
Die Anweisungen "equ", "def" sind die einzigsten, die ein Markenfeld erfordern.
Handlung: Die fehlerhafte Anweisung wird ignoriert.

33 = Ausschrift: INVALID MACRO FORMAL PARAMETER
moegl.Grund: Bei der Parameter-Deklaration fehlt das letzte Anfuehrungszeichen oder es sind vom Nutzer zu wenig Parameter vorgesehen.
Handlung: Das Einlesen der Makrozeilen-Kopie wird beendet.

34 = Ausschrift: INVALID MACRO ACTUAL PARAMETERS
moegl.Grund: Eine fehlende rechte Klammer oder ein zu langer aktueller Parameter.
Handlung: Der Aufruf des angegebenen Makros wird beendet und es werden wortfuellende Nullen erzeugt.

35 = Ausschrift: OMISSION OF A REQUIRED OPERAND FIELD
moegl.Grund: Es fehlt die Angabe eines Operandenfeldes in einer Anweisung, die ein solches erfordert.
Handlung: Fuer das nicht angegebene Operandenfeld wird der Wert 0 angenommen.

36 = Ausschrift: RECEPTION OF DISALLOWED OPERAND
moegl.Grund: Zu viele Operanden-Unterfelder sind angegeben oder ein Unterfeld wurde aufgerufen bei einem Befehl, der solche Angaben verbietet.
Handlung: Das fremde Unterfeld wird ignoriert.

39 = Ausschrift: INVALID ENTRY POINT SPECIFICATION
moegl.Grund: Keine Einsprungmarke oder die Marke ist extern definiert.
Handlung: Die angegebene Einsprungmarke wird aus der Einsprungliste gestrichen.

40 = Ausschrift: TRUNCATION ERROR
moegl.Grund: Der angegebene Wert kann nicht in das

vorgeschriebene Feld eingefuegt werden, ohne das Verschieben des signifikanten Bits ueber das "high-order" oder "low-order"-Ende hinaus. Das wird typisch durch einen zu grossen Wert verursacht oder wenn eine ungerade Adresse angegeben ist wo nur eine gerade Adresse erwartet wird.
Handlung: Der beschnittene Wert wird in das Feld eingefuegt.

50 = Ausschrift: INVALID OPERATION FIELD
moegl.Grund: selbsterklaerend
Handlung: Operation wird ignoriert.

51 = Ausschrift: INVALID CONDITIONAL ASSEMBLY DIRECTIVE
moegl.Grund: Es fehlt das Ende des bedingten Assemblerblocks.
Handlung: Die Assemblierung wird beendet.

52 = Ausschrift: INVALID ASSEMBLER DIRECTIVE
moegl.Grund: Eine falsche Anweisung, genaue Angabe (Kontrolle) erforderlich.
Handlung: Anweisung wird ignoriert.

53 = Ausschrift: NEAR-INFINITE MACRO RECURSION
moegl.Grund: Ein falsche Beendigungsbedingung wurde angegeben. (Makros koennen bis zu 1000 Rekursionsniveaus haben)
Handlung: Makroeinschieben wird beendet und alle Makro-Aufrufniveaus werden geloescht.

54 = Ausschrift: TARGET COMPUTER MEMORY OVERFLOW
moegl.Grund: Das Assemblerprogramm war zu gross. Das verschiebbare Programm kann nicht groesser sein als der verfuegbare Speicherplatz.

55 = Ausschrift: VIOLATION OF ASSEMBLER LIMITATIONS
moegl.Grund: Ueberlauf des Puffers der Zwischensprache.
Handlung: Es wird ein aktueller Puffer bereitgestellt und ein Wort bestehend aus Nullen generiert.

56 = Ausschrift: INCORRECT MACRO INVOCATION
moegl.Grund: Das Makro enthaelt eine "end"-Anweisung.
Handlung: Der Assembler beendet die Verarbeitung des Quellkodes und bearbeitet den nachfolgenden.

57 = Ausschrift: ASSEMBLER DIRECTIVE OUT OF CONTEXT
moegl.Grund: Ein bedingte "else"- oder "endif"-Anweisung wurde eingelesen, die nicht zum Aufruf einer bedingten Assemblierung passt.
Handlung: Die Anweisung wird ignoriert und ein Wert bestehend aus Nullen erzeugt.

58 = Ausschrift: RELOCATABLE FIELD CROSSING WORD BOUNDARY
moegl.Grund: Ein verschiebbarer Wert wurde in einem Feld, das auf ein Format bezug nimmt, benutzt, das die Wortbegrenzung ueberschreitet.
Handlung: Das Feld wird erzeugt wie angegeben, mit der beseitigten Verschiebungsinformation.

grundlegende Fehler
---

76 = Ausschrift: INCORRECT SOURCE MODULE SPECIFICATIONS
moegl.Grund: selbsterklaerend
Handlung: Keine; die Assemblierung wird abgebrochen.

81 = Ausschrift: INCORRECT, OUTPUT MODULE SPECIFICATION
moegl.Grund: Die Ausgabedatei wurde nicht richtig eroeffnet
Handlung: Keine; die Assemblierung wird abgebrochen

Anhang D

Druckausgaben

Eine Assemblersprachen-Quellanweisung kann sich ueber mehrere Druckzeilen ausbreiten. Die Zusammenstellung aller Druckzeilen fuer ein Assemblerprogramm nennt man Assembler-Listing. Jede gedruckte Zeile kann bis zu sieben Elemente in folgender Reihenfolge darstellen:

- Name des Makros, das die Quellzeile enthaelt (nur erste 8 Zeichen)

- Zeilennummer der Quellzeile innerhalb des gesamten Assemblerprogramms

- Nummer des aktuellen Adresszaehlers

- Wert des aktuellen Adresszaehlers

- Wert, der durch die Anweisung bestimmt wird

- Zeilennummer der Quellzeile innerhalb der Quelldatei

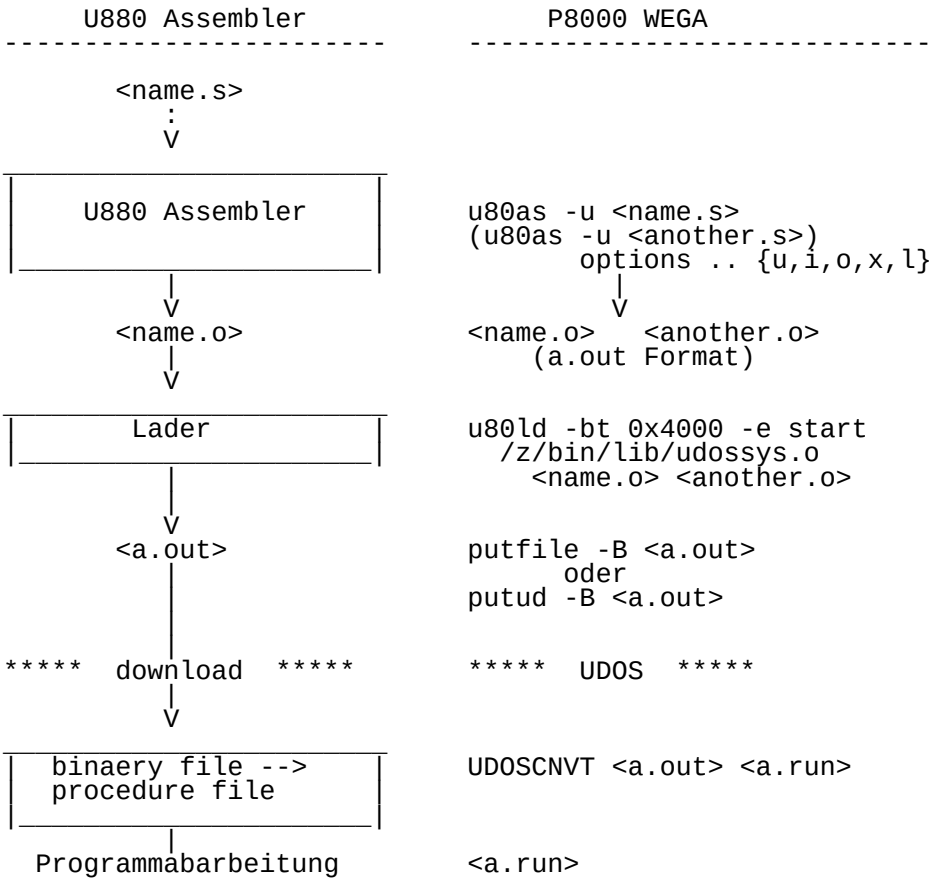
- Kopie der Quellzeile

Praktisch kann, in Abhaengigkeit vom zu verarbeitenden Typ der Quellsprachenanweisung, jede Kombination dieser Elemente auf einer Zeile auftreten. Ferner wird kein Makroname gedruckt, wenn die aktuelle Quelle von der Originaldatei ist.

Anhang E

Konvertierung des U880-Kodes in das UDOS-Format

Der folgende Teil zeigt die Schritte, die erforderlich sind, damit U880-Kode, der mit dem "u80as" assembliert wurde, auf einem UDOS-System laeuft.



Teil 3

W E G A

U880 C-Cross-Compiler

Nutzerhandbuch

Inhaltsverzeichnis Seite

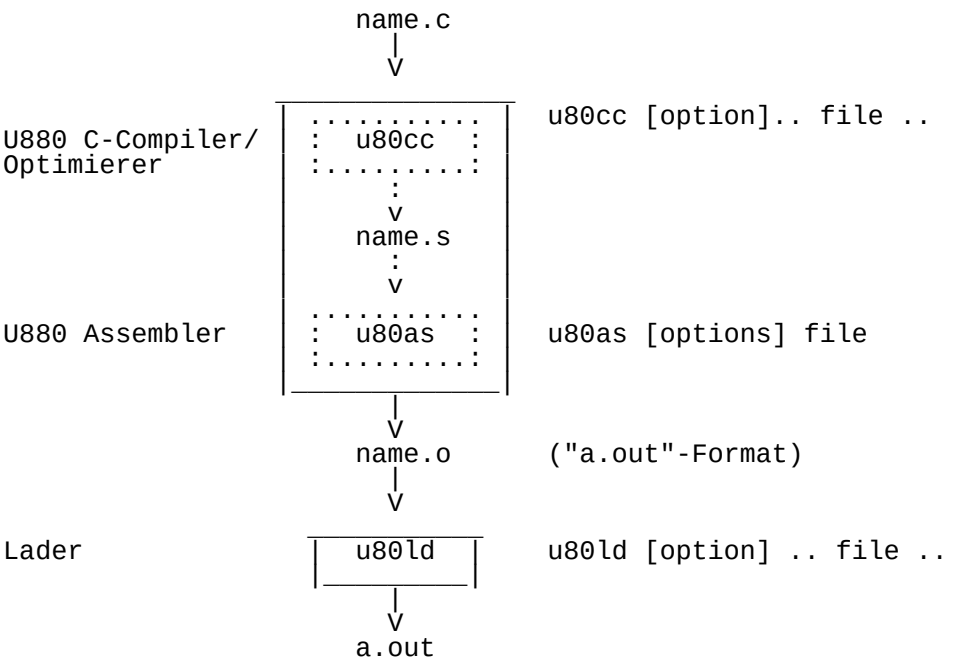
1. Einfuehrung

Der U880 C-Compiler "u80cc" ist ein Cross-Compiler, der unter dem Betriebssystem WEGA laeuft und einen ausfuehrbaren U880-Kode erstellt. Der U880-Kode wird im "a.out"-Objekt-Format erstellt. Alle Merkmale der Sprache C, ausgenommen Gleitkommadaten und -operationen und Registervariable, werden durch den "u80cc" verwendet.

2. Programmbeschreibung

Zur Schaffung eines ausfuehrbaren U880-Kodes sind mehrere Prozessschritte auszufuehren: Vorbehandlung des Quellprogramms mit dem Praeprozessor, Compilierung, Optimierung, Assemblierung und Linken. Ein Aufruf von "u80cc" bedeutet normalerweise eine Compilierung und Assemblierung des Programms. Im Compilierungsprozess werden Assembler-Quelldateien erzeugt und dann intern der "u80as"-Assembler aufgerufen. Der U880-Cross-Assembler "u80as" erstellt eine verschiebbare Datei im "a.out"-Format (siehe hierzu auch "a.out(5)" im WEGA-Programmierhandbuch).
Nach der Assemblierung kann der Linker "u80ld" aufgerufen werden, um mehrere verschiebbare Dateien in einem Lademodul zusammenzufassen, externe Bezuege aufzuloesen und Bibliotheken zu suchen.

Der U880 C-Compilierungsprozess ist nachfolgend gezeigt:



3. Kommandozeile

NAME

u80cc - U880 C-Compiler

SYNTAX

u80cc [option] ... file ...

BESCHREIBUNG

Der "u80cc" laesst verschiedene Typen von Argumenten zu. Argumente, deren Namen mit ".c" enden, werden als C-Quellprogramme betrachtet. Die Programme werden compiliert, assembliert und jedes Objektprogramm wird in der Datei abgespeichert, die den Dateinamen hat, der mit ".o" anstelle von ".c" endet.
In der gleichen Weise werden Argumente, deren Namen mit ".s" enden, als Assembler-Quellprogramme betrachtet. Die Programme werden assembliert und es wird eine Objektdatei mit der Endung ".o" anstelle von ".s" erzeugt.
Die folgenden Optionen werden durch den "u80cc" interpretiert:

-O Aufruf des Objektkodeoptimierers

-S Compiliert die angegebenen C-Quelldateien, aber assembliert sie nicht. Der entstehende Assemblersprachenkode wird in der Datei mit der Endung ".s" anstelle von ".c" abgespeichert.

-P Nur Abarbeitung des Makro-Praeprozessors und Ablage des Ergebnisses fuer jede Datei ".c" in einer entsprechenden Datei ".i".

-C Hindert den Makro-Praeprozessor, Kommentare zu loeschen.

-Dname=def
-Dname
Festlegung des Namens fuer den Praeprozessor, wie durch "#define". Erfolgte keine Festlegung, wird der Name als Wert 1 definiert.

-Uname
Beseitigung irgendeiner Namensfestlegung

-Idir
Veraenderung der Suche nach "#include"-Dateien. "#include"-Dateien, deren Namen nicht mit "/" beginnen, werden zuerst in dem Directory des Dateiarguments gesucht, dann in den Directories, die bei der Option -I angegeben wurden und dann in den Directories der Standardliste.

DATEIEN

file.c	Eingabedateie
file.o	Objektdatei
/tmp/c80?	temporaere Dateie
/lib/cpp	Praeprozessor

SIEHE AUCH

B.W. Kernighan und D.M. Ritchie,
The C Programming Language, Prentice Hill, 1978

B.W. Kernighan, Programming in C-a tutorial

D.M. Ritchie, C Reference Manual

FEHLERMELDUNGEN

Die durch C selbst erzeugten Fehlermeldungen sind selbstbeschreibend. Gelegentlich koennen Meldungen vom Assembler oder Linker ausgegeben werden.

4. Implementationsbeschreibung

Die verschiedenen Datentypen sind in ihrer Breite wie folgt festgelegt:

Typ	Breite in bit
character	8
unsigned character	8
short	8
unsigned short	8
max. Feld (unsigned short)	8
int	16
unsigned int	16
pointer	16
long	16
unsigned long	16
float	nicht implementiert
double	nicht implementiert
Registervariablen	nicht implementiert

Im Rahmen der Bibliothek benutzt der "u80cc" C-Compiler eine Reihe von Routinen, die in U880-Assembler geschrieben sind und ueber den Linker "u80ld" mit einbezogen werden muessen.
Variable vom Typ "character" werden als "unsigned character" in Divisionsoperationen behandelt.
Fuer "printf"- und "putchar"-Aufrufe sollte das "New Line"-Zeichen als "\r" spezifiziert sein.
Die U880 Bibliotheksroutinen befinden sich in der Datei /z/bin/lib/libu80c.a.
Die Felder werden vom Compiler von rechts nach links zugewiesen.
Das Programm sollte mit dem Eintrittspunkt "start" gelinkt sein und die Routine "udossys.o" sollte in den Link-Arbeitsschritt einbezogen sein.
Beispiel:

```
    u80ld -bt 0x4400 -e start
                      /z/bin/lib/udossys.o main.o -lu80c
```

Das Programm udossys.o beinhaltet die Funktionen "start" und "exit" und UDOS Ein-/Ausgabefunktionen. Die Funktion "start" ruft "main" auf, das im U880-Kode definiert ist, der durch den U880 C-Compilers produziert wird. Ein aktueller Parameter der Funktion "exit()" ist erforderlich.

Beachte: Der Compiler kann ueberlaufen, wenn Mengen mit mehr als 150 Werten initialisiert wurden.

5. Aufrufvereinbarungen

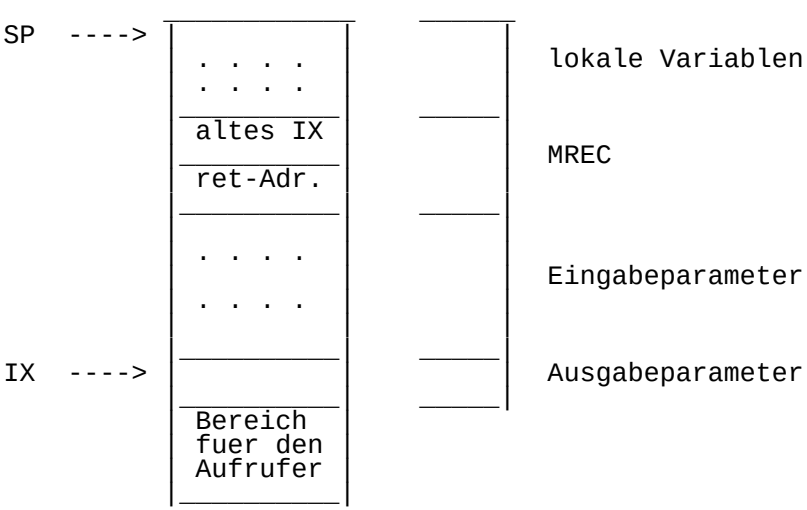
Die vom "u80cc" Compiler benutzten Aufrufvereinbarungen sind kompatibel mit dem PLZ/SYS-Parameteruebermittlungsschema, so dass eine Laufsteuerung zwischen C- und PLZ-Prozeduren erfolgen kann.

- Die Aufruffunktion reserviert Speicherplatz fuer den Rueckkehrwert und fuer die "push"-Parameter und realisiert den Aufruf. Bei variabler Anzahl von Argumenten wird die aktuelle Argumentenzahl in das A-Register geladen, ehe die Funktion aufgerufen wird.

- Die aufgerufene Funktion rettet den alten Frame-Zeiger IX und aktualisiert den laufenden Frame-Zeiger, so dass er auf die Rueckkehradresse im Stack zeigt. Dann werden die lokalen Variablen im Stack eingeladen.

- Nachdem die aufgerufene Prozedur abgearbeitet wurde, wird der alte Frame-Pointer rueckgespeichert und der Rueckkehradresswert erscheint an der Spitze des aufgerufenen Stacks.

Die Stack-Konfiguration ist nachfolgend gezeigt. Im Bild bedeuten SP Stackzeiger und MREC Mark Stack-Record.



6. Programmbeendigung

Ein Aufruf des "u80cc"-Compilers wird in der Regel durch Rueckgabe der Steuerung zum WEGA-Shell beendet.
Im Falle von Uebersetzungsfehlern versucht der "u80cc" die Fortsetzung nachdem eine Fehlernachricht zur Standard-fehlerdatei ausgegeben wurde. Bei fatalen Fehlern erfolgt ebenfalls eine Ausgabe einer Fehlernachricht zur Standard-fehlerdatei und danach erfolgt die Rueckgabe der Steuerung zum WEGA-Shell.

Teil 4

W E G A

U880 Turbo-Cross-Assembler

Nutzerhandbuch

Inhaltsverzeichnis Seite

1. Assembler

1.1. Einfuehrung

Dieser Abschnitt gibt einen Ueberblick ueber den U880 Cross-Assembler. Mit diesem Handbuch soll die Arbeitsweise des Cross-Assemblers beschrieben werden. Es wird vorausgesetzt, dass der Anwender mit der Arbeitsweise und dem Befehlsvorrat des U880 vertraut ist.

Mit Hilfe des U880 Cross-Assemblers kann der Nutzer Programme schreiben, die dann in verschiebbaren Objektkode assembliert und unter Verwendung des Linkers gebunden und auf die gewuenschte Abarbeitungsadresse gebracht werden koennen.

Der Assembler bearbeitet Dateien beliebiger Groesse, solange genuegend Speicherplatz zur Verfuegung steht. Alle vom Assembler verwendeten Puffer werden, wie benoetigt, angefordert und erweitert, mit Ausnahme des Quellkodeeingabepuffers, des Objektkodeausgabepuffers und des Puffers fuer die Listenausgabe.

Der Abschnitt ueber die bedingte Assemblierung befaehigt den Anwender, den Assembler so zu steuern, dass, in Abhaengigkeit vom Ergebnis waehrend der Abarbeitungszeit, unterschiedliche Abschitte der Quelldatei bearbeitet werden koennen. Bedingungen koennen bis zu 248 Ebenen verschachtelt sein. Dabei unterstuetzt der Assembler den Programmierer beim Auffinden von Fehlern bei den bedingten Verschachtelungen, indem er nicht nur nachprueft, ob alle Verschachtelungsebenen abgeschlossen sind, sondern auch dadurch, dass er die momentan aktive Bedingungsebene im Objektkodefeld des Listings anzeigt.

Der Abschnitt ueber die waehrend der Assemblierung durchgefuehrte Berechnung von Ausdruecken zeigt, dass Berechnungen mit bis zu 16 Operanden moeglich sind. Dabei wird eine 80-Bit-Arithmetik verwendet. Die algebraische Hierarchie kann durch die Verwendung von Klammern veraendert werden.

Im Abschnitt ueber die Steuerung des Listendrucks werden Hinweise gegeben, wie das gesamte Programm oder nur Teile davon ausgegeben werden koennen. Das Programmlisting enthaelt brauchbare Hinweise zur Fehlerfindung. Es werden Kommandos beschrieben, mit denen waehrend der Assemblierung der Ausgabemodus veraendert werden kann.

Der Linker gestattet es, Dateien miteinander zu verbinden oder sie auch nur zur Herstellung von externen Adressverbindungen zu benutzen. Genau wie bei dem Assembler werden alle vom Linker verwendeten Puffer so angefordert, wie sie benoetigt werden. Der Linker ist in der Lage, verschiedene unterschiedliche Ausgabeformate zu erzeugen. Das Format kann durch Verwendung einer Assembler-Steueranweisung veraendert werden. Man kann die gewuenschte Ausgabe auch ueber das Linker-Optionenfeld auswaehlen. Programme

koennen bis zu 256 vom Nutzer definierte Abschnitte (sections) enthalten, und der Linker kann dementsprechend bis zu 256 Abschnittsnamen verarbeiten. Eine vollstaendige Beschreibung wird hierzu im Abschnitt ueber den Linker gegeben.

1.2. U880 Cross-Assembler Bedienungsanweisungen

1.2.1. Prompt-Modus

Um den Assembler aufzurufen, wird eingegeben: asm80

Der Assembler antwortet:

```
Listing Destination ? (N, T, D, E, L, <CR> = N) :
(Ziel fuer Listenausgabe?)
```

Dabei haben die Abkuerzungen folgende Bedeutung:

```
N = None (ohne)
T = Terminal
D = Disk
E = Error Only (Nur Fehler)
L = List On/Off (Listenausgabe ein/aus)
```

Soll die Listenausgabe gemaess der Assemblersteueranweisungen LIST ON/OFF entstehen (L), wird jetzt die nachstehende Eingabeanforderung ausgegeben:

```
LIST ON/OFF Listing Destination (T, D, <CR> = T) :
```

Die Abkuerzungen haben dieselbe Bedeutung wie oben.

Die Listenausgabe ueber die LIST ON/OFF-Steueranweisungen gibt dem Nutzer die Moeglichkeit, ausgewaehlte Teile der Quelldatei auszugeben. Weitere Informationen sind dem Abschnitt ueber die Steuerung der Listenausgabe zu entnehmen.

Wurde 'Error Only' (E) gewaehlt, fordert der Assembler den Nutzer wiefolgt zur Angabe des Bestimmungsortes auf.

```
Error Only Listing Destination (T, D, <CR> = T):
```

Wurde 'Disk' (D) gewaehlt, oder soll bei 'L' die Liste an den Disk-Speicher ausgegeben werden, fragt der Assembler, ob eine Cross-Referenz-Liste gewuenscht wird.

```
Generate Cross Reference ? (Y/N <CR> = No) :
```

Danach fordert der Assembler den Bediener zur Eingabe des Quellkodedateinamens auf.

```
Input Filename :
```

Bei der Eingabe des Quelldateinamens kann eine Namenserweiterung angegeben werden. Andernfalls sucht der Assem-

bler nach der Namenserweiterung 'asm'.

Nach Eingabe des Quelldateinamens fordert der Assembler zur Eingabe des Namens der Ausgabedatei auf.

```
Output Filename :
```

Beantwortet der Bediener die Eingabeanforderung nur mit einem <CR> (Wagenruecklauf), so wird die Ausgabedatei denselben Dateinamen wie die Eingabedatei erhalten, aber mit der Namenserweiterung 'obj'. Besteht die Eingabe nur aus einem Dateinamen ohne Namenserweiterung, so erhaelt die Ausgabedatei diesen Namen und die Namenserweiterung 'obj'.

1.2.2. Kommandozeilen-Modus

Der Assembler kann auch unter Verwendung einer Kommandozeile aufgerufen werden. In diesem Fall wird zuerst der Eingabedateiname, dann der Ausgabedateiname und anschliessend eine Liste von Optionen angegeben. Sowohl die Angabe der Ausgabedatei als auch die Angabe des Ziels fuer die Listenausgabe ist optional. Das Kommando hat, die optionalen Angaben in Klammern eingeschlossen, folgende allgemeine Form:

```
asm80 [-q] input_filename [output_filename]
                          [-t,-d,-dx,-et,-ed,-lt,-ld,-ldx]
```

Die Option '-q' bewirkt den 'Quiet'-Mode. Wird diese Option gewaehlt, so werden vom Assembler nur die Fehlermeldungen mit der dazugehoerigen Zeilennummer auf dem Bildschirm ausgegeben. Diese Option muss vor dem Eingabedateinamen angegeben werden.

Es folgen einige Beispiele fuer moegliche Kommandozeilen.

Nur Eingabedateiname :

```
asm80 input_filename
```

Dieses Kommando veranlasst den Assembler, die Quellendatei 'input_filename' zu bearbeiten. Wurde keine Namenserweiterung angegeben, wird angenommen, dass sie '.asm' lautet. Da keine Optionen angegeben wurden, wird standardmaessig nur die Fehlerliste ueber den Bildschirm ausgegeben. Der Ausgabedateiname wird derselbe wie der der Eingabedatei sein, jedoch mit der Namenserweiterung '.obj'.

Eingabe- und Ausgabedateiname :

```
asm80 input_filename output_filename
```

Dieses Kommando ist mit dem vorhergehenden identisch, mit dem Unterschied, dass der Assembler die Objektdatei 'output_filename' nennen wird.

Listenausgabe ueber Bildschirm :

asm80 input_filename output_filename -t

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die Liste wird ueber den Bildschirm ausgegeben. Die optionale Angabe des Ausgabedateinamens veranlasst den Assembler, eine Objektdatei mit dem Namen 'output_filename' zu erzeugen.

Listenausgabe an Disk-Speicher :

asm80 input_filename output_filename -d

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die Liste wird an den Disk-Speicher ausgegeben. Die Listendatei auf dem Disk-Speicher erhaelt denselben Namen wie die Eingabedatei, aber mit der Namenserweiterung '.lst'. Die optionale Angabe des Ausgabedateinamens veranlasst den Assembler, eine Objektdatei mit dem Namen 'output_filename' zu erzeugen.

Listenausgabe mit Cross-Referenztabelle an Disk-Speicher :

asm80 input_filename output_filename -dx

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die Liste und die Cross-Referenztabelle wird an den Disk-Speicher ausgegeben. Die Listendatei auf dem Disk-Speicher erhaelt denselben Namen wie die Eingabedatei, aber mit der Namenserweiterung '.lst'. Die optionale Angabe des Ausgabedateinamens veranlasst den Assembler, eine Objektdatei mit dem Namen 'output_filename' zu erzeugen.

Fehlerlistenausgabe ueber den Bildschirm :

asm80 input_filename output_filename -et

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die Fehlermeldungen werden ueber den Bildschirm ausgegeben. Die optionale Angabe des Ausgabedateinamens veranlasst den Assembler, eine Objektdatei mit dem Namen 'output_filename' zu erzeugen.

Fehlerlistenausgabe auf Disk-Speicher :

asm80 input_filename output_filename -ed

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die Fehlermeldungen werden an den Disk-Speicher ausgegeben. Die Listendatei auf dem Disk-Speicher erhaelt denselben Namen wie die Eingabedatei, aber mit der Namenserweiterung '.lst'. Die optionale Angabe des Ausgabedateinamens veranlasst den Assembler, eine Objektdatei mit dem Namen 'output_filename' zu erzeugen.

'LIST ON/OFF'-Listenausgabe ueber den Bildschirm :

```
asm80   input_filename   -lt
```

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die durch LIST ON/OFF gekennzeichneten Bloecke werden ueber den Bildschirm ausgegeben.

'LIST ON/OFF'-Listenausgabe an den Disk-Speicher :

```
asm80   input_filename   -ld
```

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die durch LIST ON/OFF gekennzeichneten Bloecke werden an den Disk-Speicher ausgegeben. Die Listendatei auf dem Disk-Speicher erhaelt denselben Namen wie die Eingabedatei, aber mit der Namenserweiterung '.lst'.

'LIST ON/OFF'-Listenausgabe an den Disk-Speicher mit Cross-Referenztabelle:

```
asm80   input_filename   -ldx
```

Mit diesem Kommando wird die Eingabedatei 'input_filename' assembliert. Die durch LIST ON/OFF gekennzeichneten Bloecke und die Cross-Referenztabelle werden an den Disk-Speicher ausgegeben. Die Listendatei auf dem Disk-Speicher erhaelt denselben Namen wie die Eingabedatei, aber mit der Namenserweiterung '.lst'.

1.3. Standardfestlegungen des Systems

Die nachfolgend angegebenen Dateinamenserweiterungen werden standardmaessig durch das System festgelegt, sofern der Nutzer keine anderen Festlegungen trifft.

```
asm - Eingabe fuer den Assembler
obj - Ausgabe vom Assembler
lst - Datei fuer die Listenausgabe

obj - Eingabe fuer den Linker

tsk - abarbeitungsfaehiger Objektkode
hex - 'I-Hex'- und 'Extended I-Hex'-Format
tek - 'T-Hex'-Format
s19 - 'Moto-S19'-Format
s28 - 'Moto-S28'-Format
s37 - 'Moto-S37'-Format
```

Es ist zu beachten, dass wegen der zusaetzlichen Informationen, die die Assembler-Ausgabedatei enthaelt, der Linker in jedem Fall benutzt werden muss. Das gilt auch, wenn das Programm schon auf der gewuenschten Startadresse assembliert wurde und keine externen Adressbezuege notwendig sind. Es ist einfach so, dass erst durch den Linker alle zusaetzlichen Informationen entfernt werden und eine Ladedatei erzeugt werden kann.

1.4. Assembler-Abarbeitungskommandos

Die folgenden Kommandos sind waehrend des Assemblierungsvorganges aktiv. Diese Kommandos sind sowohl waehrend des ersten als auch waehrend des zweiten Durchlaufs aktiv. Sie ueberschreiben den beim Aufruf des Assemblers festgelegten Ausgabemodus.

Ctrl S - Unterbrechung des Assemblers
Ctrl Q - Fortsetzung des Assemblers
Del C - Beenden der Assemblierung
Del T - Ausgabe des Listings auf dem Bildschirm
Del D - Ausgabe des Listings auf den Disk-Speicher
Del M - Ausgabe des Listings sowohl auf Bildschirm als auch auf Disk-Speicher
Del N - Abbruch der Listingausgabe

1.5. Assembler-Fehlerbehandlung

Tritt ein Assemblierungsfehler auf, so sind die Massnahmen, die der Assembler trifft, abhaengig von dem Ausgabemodus, unter dem er gerade arbeitet.

Wurde die N-Option (keine Listenausgabe) angegeben, so wird die Anweisung, die den Fehler und die Fehlernachricht verursacht hat, auf dem Bildschirm angezeigt. Die Bildschirmanzeige wird eingeschaltet und der Assembler haelt an, als haette der Nutzer ^S eingegeben. Der Grund dafuer ist, dass dem Nutzer die Moeglichkeit gegeben werden soll, genau zu sehen, wo sich der Fehler befindet. Das kann sowohl beim ersten als auch beim zweiten Durchlauf geschehen. Man beachte, dass manche Fehler, wie z.B. undefinierte Symbole beim ersten Durchlauf nicht gefunden werden koennen. Nachdem der Fehler angezeigt wurde, kann die Bildschirmanzeige mit Del N wieder unterdrueckt werden.

Ist die Ausgabe des Listings auf den Disk-Speicher vorgesehen, so werden im ersten Durchlauf festgestellte Fehler auf dem Bildschirm angezeigt, aber nicht auf den Disk-Speicher ausgegeben. Der Assembler haelt auch nicht an. Beim zweiten Durchlauf wird der Fehler sowohl auf den Disk-Speicher als auch ueber den Bildschirm ausgegeben und der Assembler arbeitet weiter.

Ist die Ausgabe des Listings auf den Disk-Speicher gemaess Steueranweisungen des Assemblers vorgesehen, so werden Fehler, die im ersten Durchlauf festgestellt wurden, ueber den Bildschirm ausgegeben, nicht jedoch auf den Disk-Speicher. Der Assembler arbeitet weiter. Fehler, die im zweiten Durchlauf festgestellt werden, werden sowohl auf den Disk-Speicher als auch ueber den Bildschirm ausgegeben, selbst wenn sie sich in einem Block befinden, der nicht fuer die Ausgabe vorgesehen war.

1.6. Syntaxregeln der U880-Assemblersprache

Dieser Abschnitt beschreibt die vom U880 Cross-Assembler verwendete Syntax.

1.6.1. Bezeichnung der Zahlenbasis

Zahlenbasen werden folgendermassen bezeichnet:

Binary	-	B
Octal	-	O oder Q
Decimal	-	D oder ohne Bezeichnung der Basis
Hex	-	H oder ein vorgestelltes %-Zeichen
ASCII	-	Einfache oder doppelte Hochkommas - "X" oder 'X'

Die nachfolgend angegebenen 2-Zeichenfolgen, die in einfache oder doppelte Hochkommas eingeschlossen sein koennen, sind folgendermassen definiert. Es muss jedoch die TWOCHAR ON-Steueranweisung verwendet werden, um sie wirksam zu machen.

"CR" oder 'CR'	-	Carriage return (Wagenruecklauf)
"LF" oder 'LF'	-	Line feed (Zeilenschaltung)
"SP" oder 'SP'	-	Space (Leerzeichen)
"HT" oder 'HT'	-	Horizontal tab (Tabulator)
"NL" oder 'NL'	-	Null

1.6.2. Programmkommentare

Kommentarzeilen muessen mit einem Semikolon oder einem Stern in Spalte 1 beginnen, falls nicht die .COMMENT-Steueranweisung verwendet wird. Kommentare nach einem Befehl benoetigen kein Semikolon, sofern sie wenigstens durch ein Leerzeichen oder einen Tabulator vom Befehl getrennt sind und der Assembler im 'Spaces Off'-Mode arbeitet. Arbeitet der Assembler im 'Spaces On'-Mode, dann muessen alle Kommentare nach einer Anweisung mit einem Semikolon beginnen. Weitere Informationen dazu und zur Standardeinstellung sind der Beschreibung der SPACE-Steueranweisung zu entnehmen.

1.6.3. Marken

Marken koennen beliebig lang sein, jedoch sind nur 32 Zeichen von Bedeutung. Sie koennen in jeder Spalte beginnen, wenn der Name durch einen Doppelpunkt abgeschlossen wird. Wird kein Doppelpunkt verwendet, muss die Marke in Spalte 1 beginnen. Alle Marken muessen mit einem Buchstaben beginnen. Es wird zwischen Gross- und Kleinbuchstaben unterschieden.

1.6.4. Befehlszaehler ($ oder *)

Die Sonderzeichen '$' oder '*' koennen in einem Ausdruck verwendet werden, um den Befehlszaehler zu bezeichnen. Der Wert, der dem $- oder dem *-Zeichen zugeordnet wird, ist der Wert des Befehlszaehlers bei Beginn des Befehls.

1.6.5. Hoeherwertiges Byte

Um das hoeherwertige Byte eines 16-Bit-Wertes zu laden, muss man das Zeichen fuer 'groesser als', >, verwenden. Damit ist es moeglich, die Bits 8 bis 15 als einen Bytewert, der verschiebbar ist, zu benutzen.

1.6.6. Niederwertiges Byte

Um das niederwertige Byte eines 16-Bit-Wertes zu laden, muss man das Zeichen fuer 'kleiner als', <, verwenden. Damit ist es moeglich, die Bits 0 bis 7 als einen Bytewert, der verschiebbar ist, zu benutzen.

1.6.7. Gross-/Kleinbuchstaben

Bei Marken wird zwischen Gross- und Kleinbuchstaben unterschieden. Ebenso werden Marken, die als Abschnitts- oder Makronamen verwendet werden, als unterschiedlich angesehen, wenn man sie klein statt gross schreibt.

1.7. Adressierungsarten

Die verschiedenen Adressierungsarten, die der U880 gestattet, werden nachfolgend anhand von Beispielen gezeigt.

1.7.1. Unmittelbare (immediate) Adressierung

Der Datenwert ist im Befehl enthalten.

Beispiele:

```
LD   HL,1234H       ;Lade HL mit hexadezimal # 1234
LD   HL,DATA        ;Lade HL mit dem Wert, der mit der
                     Marke 'DATA' verbunden ist
```

1.7.2. Register Adressierung

Der Datenwert ist in einem CPU-Register enthalten.

Beispiele:

```
LD   A,B            ;Lade  den Inhalt des Registers  B
```

```
                        in das Register A
    SUB  D              ;Subtrahiere  den Inhalt des Regi-
                         sters B von dem des Registers A
```

1.7.3. Indirekte Register Adressierung

Ein Register weist auf die Operandenadresse.

Beispiele:

```
    LD   A,(HL)         ;Lade  A mit dem Inhalt des  Spei-
                         cherplatzes,  auf  den  HL  weist
    LD   (HL),B         ;Speichere  den Inhalt von  B  auf
                         dem  Speicherplatz,  auf  den  HL
                         weist
```

1.7.4. Direkte Adressierung

Die Adresse des Operanden ist im Befehl enthalten.

Beispiele:

```
    LD   HL,(1234H)     ;Lade   HL  mit  dem  Inhalt   des
                         Speicherplatzes  mit der  Adresse
                         1234 HEX
    LD   (ADDRESS),HL   ;Speichere HL in dem Speicherplatz
                         'ADDRESS'
    LD   (ABCDH),HL     ;Speichere HL in dem Speicherplatz
                         ABCD hexadezimal
```

1.7.5. Index Adressierung

Die Operandenadresse ist die Summe aus dem 8-Bit-Offset im Befehl und dem Inhalt des Registers IX oder IY.

Beispiele:

```
    LD   A,(IX+4)       ;Lade  A mit dem Inhalt des  Spei-
                         cherplatzes,  dessen  Adresse ge-
                         bildet wird,  indem 4 zu dem  In-
                         halt  des  Registers  IX  addiert
                         wird
    LD   (IX+DATA8),B   ;Speichere B in dem Speicherplatz,
                         dessen   Adresse  gebildet  wird,
                         indem der Wert,  der mit  'DATA8'
                         verbunden  ist,  zum  Inhalt  des
                         Registers IX addiert wird
```

1.7.6. Relative Adressierung

Die Operandenadresse ist relativ zum aktuellen Befehl. Wird die Adresse als numerischer Wert gegeben, so wird am Anfang des naechsten Befehles begonnen zu zaehlen.

Beispiele:

```
DJNZ    LOOP    ;Der Assembler berechnet die Zieladresse, indem die Adresse der
                 Marke 'LOOP' von der Adresse des naechsten Befehls abgezogen wird

JR      4       ;Die Zieladresse befindet sich 4 Bytes hinter dem Beginn des
                 naechsten Befehls
```

1.8. Assembler-Steueranweisungen

Dieser Abschnitt beschreibt die Assembler-Steueranweisungen. Es ist moeglich, den Steueranweisungen einen Dezimalpunkt voranzustellen. Das erleichtert es, sie von Programmbefehlen zu unterscheiden.

1.8.1. Steueranweisungen fuer den Speicher

```
            ORG     VALUE
            ORIGIN
```

Legt die absolute Adresse fuer die Assemblierung fest. Wird diese Steueranweisung nicht gegeben, wird die Assemblierungsadresse auf 0000 voreingestellt.

```
            END     VALUE
```

Diese Steueranweisung definiert das Ende eines Programms oder einer durch eine INCLUDE-Anweisung einbezogenen Datei. Der der END-Anweisung folgende Ausdruck ist optional und kennzeichnet, sofern er angegeben wird, die Startadresse des Programms. Diese Adresse wird kodiert in die Ausgabedatei eingetragen, falls das Format der Ausgabedatei eine Eintragung der Programmstartadresse vorsieht.

```
LABEL:      DB      VALUE
            FCB
            DEFB
            BYTE
            STRING
```

Der Assembler legt den Wert des Ausdrucks in aufeinanderfolgenden Speicherplaetzen ab. Der VALUE-Ausdruck kann eine beliebige Mischung von Operandentypen sein, die voneinander jeweils durch Komma getrennt sind. ASCII-Zeichenketten muessen in Apostrophe eingeschlossen sein. Wenn die Zeichenkette ein Apostroph enthaelt, wird das durch zwei nebeneinander stehende Apostrophe angegeben. Wird kein

Ausdruck angegeben , so wird ein Byte reserviert und zu Null gesetzt. Eine Marke ist optional. Nachfolgend werden ein paar Beispiele fuer die Anwendung der BYTE-Steueranweisung gegeben.

```
     .BYTE                  ;Reserviert 1 zu Null gesetztes
                             Byte
     .BYTE     10           ;Reserviert 1 Byte=10 (dezimal)
     .BYTE     1,2,3        ;Reserviert 3 Bytes, = zu 1,2 &
                             3 gesetzt, in dieser Reihen-
                             folge
     .BYTE     SYMBOL-10    ;Sucht in der Symboltabelle
                             nach SYMBOL, subtrahiert 10
                             (dezim.) von seinem Wert und
                             speichert das Ergebnis
     .BYTE     'Hello'      ;Speichert die der Zeichenkette
                             'Hello' entsprechenden ASCII-
                             Zeichen in aufeinanderfolgen-
                             den Speicherplaetzen
     .BYTE     'Hello', 0DH ;Wie im obigen Beispiel, mit
                             zusaetzlichem CR am Ende.Leer-
                             zeichen vor den Operanden
                             bleiben unberuecksichtigt,
                             aber das Komma ist erforder-
                             lich.
     .BYTE     'Karl''s'    ;Eingeschlossenes Apostroph.
```

```
LABEL:         DW        VALUE
               FDB
               DEFW
               WORD
```

Diese Steueranweisung speichert den Wert des Ausdrucks in einem 16-Bit-Speicherbereich. Es koennen mehrere WORD-Ausdruecke in einer Anweisung angegeben werden. Sie muessen aber voneinander durch Komma getrennt sein. Wird ueberhaupt kein Ausdruck angegeben, so wird 1 WORD reserviert und zu Null gesetzt. Die Verwendung einer Marke ist optional.

```
LABEL:         LONG      VALUE
               LONGW
               LWORD
```

Diese Steueranweisung speichert den Wert des Ausdrucks in einem 32-Bit-Speicherbereich. Es koennen mehrere LONG WORD-Ausdruecke in einer Anweisung angegeben werden, sie muessen aber voneinander durch Komma getrennt sein. Wird ueberhaupt kein Ausdruck angegeben, so wird 1 LONG WORD reserviert und zu Null gesetzt. Die Verwendung einer Marke ist optional.

```
LABEL:          ASCII     STRING
```

Speichert die Zeichenkette STRING, die entweder bis zu einem CR (Wagenruecklauf) oder einem "|" (HEX 7C) reicht, es aber nicht enthaelt. Die Marke ist optional. Nachfolgend werden ein paar Beispiele fuer die ASCII-Steueranweisung gegeben.

```
     ASCII     Hello     ;Speichert die ASCII-Darstellung
                          von Hello in aufeinanderfolgenden
                          Speicherplaetzen. Uebrigens wuerde
                          auch dieser Kommentar mit gespei-
                          chert werden.
     ASCII     Hello|    ;Nun wuerde der Kommentar nicht mit
                          gespeichert werden. Im naechsten
                          Beispiel wurde die Zeichenkette
                          nur mit einem CR abgeschlossen.
     ASCII     Hello
```

```
LABEL:          FCC       STRING
```

Speichert die durch zwei identische Zeichen eingeschlossene Zeichenkette STRING im Speicher. Dabei werden das erste sowie das ihm entsprechende letzte Zeichen nicht mit gespeichert. Die Verwendung einer Marke ist optional. Eine typische Anwendung wuerde folgendermassen aussehen:

```
                FCC       /Das ist eine Test-Zeichenkette/
```

```
LABEL:          BLKB      SIZE, VALUE
```

Reserviert die Anzahl von Bytes, die durch den Ausdruck 'SIZE' festgelegt wird. Wird fuer 'VALUE' ein Wert angegeben, so wird dieser Wert jedem Byte zugewiesen. Andernfalls werden die reservierten Bytes zu Null gesetzt. Die Marke ist optional.

```
     BLKB      20        ;Reserviert 20 Bytes, die zu Null
                          gesetzt werden
     BLKB      20,0      ;Reserviert 20 Bytes, die zu Null
                          gesetzt werden
     BLKB      20,FFH    ;Reserviert 20 Bytes und setzt
                          jedes auf FF Hex
```

```
LABEL:          DS        SIZE
                RMB
                DEFS
```

Diese Steueranweisung reserviert die Anzahl von Bytes, die durch den Ausdruck 'SIZE' festgelegt wird. Es werden keine Werte in den reservierten Bereich gebracht. Diese Steueranweisung unterscheidet sich von der BLKB-Anweisung in folgender Weise: Unter der Bedingung, dass sich die reservierten Speicherplaetze am Ende eines Programmabschnitts befinden, dass die Ausgabe des Linkers ausfuehrbar

ist und dass der Linker keinen weiteren Modul an diesen Abschnitt anschliessen soll, werden die reservierten Bytes nicht in die Ausgabedatei aufgenommen.

```
LABEL:          BLKW      SIZE, VALUE
```

Reserviert die Anzahl von 16-Bit-Worten, die durch den Ausdruck 'SIZE' festgelegt werden. Wird fuer 'VALUE' ein Wert angegeben, so wird dieser Wert jedem WORD zugewiesen. Andernfalls werden die reservierten Worte zu Null gesetzt. Die Marke ist optional.

```
        BLKW      20          ;Reserviert 20 Worte, die zu Null
                               gesetzt werden
        BLKW      20,0        ;Reserviert 20 Worte, die zu Null
                               gesetzt werden
        BLKW      20,FFFFH    ;Reserviert 20 Worte und setzt
                               jedes auf FFFF Hex
```

```
LABEL:          BLKL      SIZE, VALUE
```

Reserviert die Anzahl von 32-Bit-Worten, die durch den Ausdruck 'SIZE' festgelegt werden. Wird fuer 'VALUE' ein Wert angegeben, so wird dieser Wert jedem 'LONG WORD' zugewiesen. Andernfalls wird jedes von ihnen zu Null gesetzt. Die Marke ist optional.

```
        BLKL      20          ;Reserviert 20 'LONG WORD' die zu
                               Null gesetzt werden
        BLKL      20,0        ;Reserviert 20 'LONG WORD' die zu
                               Null gesetzt werden
        BLKL      20,FFFFH    ;Reserviert 20 'LONG WORD' und
                               setzt jedes auf FFFF Hex
```

1.8.2. Steueranweisungen fuer Definitionen

```
LABEL:          EQU       VALUE
                EQUAL
```

Setzt 'LABEL' dem Wert von 'VALUE' gleich, wobei 'VALUE' ein anderes Symbol oder ein gueltiger arithmetischer Ausdruck sein kann.

```
LABEL:          VAR       VALUE
                DEFL
```

Setzt 'LABEL' auf den Wert von 'VALUE'. Diese Zuordnung kann jedoch innerhalb des gesamten Programms immer wieder veraendert werden. Eine Marke, die als Variable definiert wurde, sollte nicht durch eine EQUAL-Steueranweisung neu definiert werden.

```
LABEL:          MACRO     ARGS
```

Kennzeichnet den Beginn einer Makrodefinition.

```
                ENDM
                MACEND
```

Kennzeichnet das Ende einer Makrodefinition.

```
                MACEXIT
```

Diese Steueranweisung veranlasst den unmittelbaren Austritt aus einem Makro. Der Unterschied zwischen MACEXIT und MACEND ist der, dass waehrend des Vorgangs der Makrodefinition das Makro durch MACEXIT nicht beendet wird und dass, wenn sich MACEXIT im Zweig der nicht erfuellten Bedingung eines bedingten Assemblerblocks befindet, es nicht ausgefuehrt wird. Alle bedingungsabhaengigen Assemblerwerte werden in denselben Zustand gebracht, als waere das Makro aufgerufen worden.

```
                XDEF      LABEL
                GLOBAL
                PUBLIC
```

Kennzeichnet die Marke als eine globale Marke, auf die von anderen Programmen zugegriffen werden kann. Es koennen mehrere Marken angegeben werden, die aber durch Komma getrennt sein muessen. Es werden einige Beispiele fuer den korrekten Gebrauch von GLOBAL angegeben.

```
GLOBAL    SYM1         ;Deklariert die Marke SYM1 fuer
                        andere Programme als verfueg-
                        bar. Der Linker wird die exter-
                        nen Adressbezuege aufloesen.
GLOBAL    SYM1, SYM2   ;Mehrfache Deklarationen auf ei-
                        ner Zeile sind zulaessig, wenn
                        sie durch Komma getrennt sind.
                        Leerzeichen werden ignoriert.
```

```
                XREF      LABEL
                EXTERN
                EXTERNAL
```

Macht kenntlich, dass die Marke in einem anderen Programm definiert wurde. Es koennen mehrere Marken angegeben werden. Sie muessen aber durch Komma voneinander getrennt sein.

```
LABEL:          ASK       PROMPT
```

Gibt 'PROMPT' auf das Teminal aus und wartet auf eine 1 Zeichen Eingabe, von der 30H subtrahiert wird. Gewoehnlich ist das Ziel dieser Steueranweisung, ein 0/1 Flag in das Programm einzufuehren. 'LABEL' wird dem Ergebnis gleichgesetzt. Ein CR beendet 'PROMPT'. Beim zweiten Durchlauf wird die Zeile gemeinsam mit der Antwort ausgegeben.

Nachfolgend ein Beispiel fuer 'ASK':

```
DISK_SIZE: ASK  ASSEMBLE FOR 8''(=1) or 5 1/4''(=0) DRIVES?
```

1.8.3. Assemblierungsmodus

```
LABEL:          SECTION
```

Mit dieser Steueranweisung koennen vom Nutzer definierte Abschnittsnamen erzeugt werden. Der Assembler hat zwei vordefinierte Abschnitte, CODE und DATA. Insgesamt sind je Datei 256 Abschnittsnamen moeglich. Jeder Name kann bis zu 32 Zeichen lang sein. Es wird zwischen Gross- und Kleinbuchstaben unterschieden. Nachdem der Abschnitt definiert wurde, kann das Programm von einem zum anderen Abschnitt umschalten, indem es einfach den Namen als Mnemonik benutzt. Der standardmaessig eingestellte Abschnitt ist CODE. Abschnitte koennen verschachtelt werden. Wie allen Steueranweisungen kann dem Abschnittsnamen ein Punkt vorangestellt werden. Im Kapitel ueber die Linker-Bedienungsanweisungen wird beschrieben, wie der Linker mit diesen Abschnittsnamen verfaehrt. Nachfolgend werden ein paar Beispiele fuer das Definieren von Abschnittsnamen und das Umschalten zwischen verschiedenen Abschnitten gegeben.

```
           NOP        ;Diese  Anweisung geht  standardmaessig
                       in den CODE-Abschnitt
           .DATA      ;Schaltet  in den vordefinierten  DATA-
                       Abschnitt um
           .BYTE      ;Dieses Byte geht in den DATA-Abschnitt
SECTION1:  .SECTION   ;Definiert  einen neuen Abschnitt.  Die
                       Definition macht diesen Abschnitt  au-
                       tomatisch aktiv.
           NOP        ;Diese  Anweisung geht in den SECTION1-
                       Abschnitt
           .CODE      ;Schaltet zurueck in den CODE-Abschnitt
           NOP        ;Diese  Anweisung  geht  in  den  CODE-
                       Abschnitt
           .SECTION1  ;Schaltet in den vom Nutzer definierten
                       SECTION1-Abschnitt um
           NOP        ;Diese  Anweisung geht in den SECTION1-
                       Abschnitt
           .BYTE      ;Jeder  beliebige Abschnitt kann sowohl
                       Kode  als auch Daten oder auch  beides
                       enthalten

                ENDS
```

Diese Steueranweisung wird in Verbindung mit der SECTION-Steueranweisung verwendet. ENDS ermoeglicht die Beendigung von verschachtelten Abschnitten in einer Datei.

RADIX VALUE

Setzt die Assembler Zahlenbasis wie folgt:

2 oder B = Binaer
8 oder O oder Q = Oktal
10 oder D = Dezimal
16 oder H = Hexadezimal

Wird kein Ausdruck angegeben, so wird zum voreingestellten Standardmodus, Basis 10, zurueckgekehrt und vorausgesetzt, dass jede andere Basis entsprechend mit B, Q, D oder H hinter der Konstanten gekennzeichnet wird. Man beachte, dass bei eingestellter Basis 16 keine Moeglichkeit besteht, dezimale oder binaere Zahlen zu kennzeichnen, da sowohl D als auch B gueltige hexadezimale Zahlen sind.

INCLUDE filename

Weist den Assembler an, die angegebene Datei in die Assemblierung einzubeziehen. Dateinamen koennen Pfadnamen enthalten. Dateinamenserweiterungen muessen vollstaendig angegeben werden. Include-Anweisungen sollen nicht verschachtelt sein.

SPACES ON

Diese Steueranweisung laesst Leerzeichen zwischen den Operanden zu. Wenn Leerzeichen zugelassen sind, muessen Kommentare mit einem Semikolon beginnen. Das ist der voreingestellte Standardmodus.

SPACES OFF

Diese Steueranweisung verbietet Leerzeichen zwischen Operanden. In diesem Fall muessen Kommentare nicht mit einem Semikolon beginnen. Der voreingestellte Standardmodus ist SPACES ON.

TWOCHAR ON

Diese Steueranweisung gestattet die Verwendung der unten angegebenen Zwei-Zeichen-Abkuerzungen (ASCII). Der voreingestellte Standardmodus ist TWOCHAR OFF.

"CR" oder 'CR' - Wagenruecklauf
"LF" oder 'LF' - Zeilenschaltung
"SP" oder 'SP' - Leerzeichen
"HT" oder 'HT' - horizontaler Tabulator
"NL" oder 'NL' - Null

TWOCHAR OFF

Diese Steueranweisung verbietet die Verwendung der in der vorigen Steueranweisung angegebenen Zwei-Zeichen-Abkuerzungen. Das ist der voreingestellte Standardmodus.

```
COMMENT   X
```

Weist den Assembler an, ausgehend von dieser Zeile alle nachfolgenden Zeilen bis zum naechsten 'X' als Kommentar anzusehen. 'X' kann jedes beliebige Zeichen sein. Der Kommentar muss mindestens zwei Zeilen lang sein.

1.8.4. Bedingte Assemblierung

```
IFZ       VALUE
```

Der Assembler wird die auf diese Steueranweisung folgenden Anweisungen bis zu einer ELSE- oder ENDIF-Steueranweisung uebersetzen, falls der Wert von VALUE gleich Null ist. Bedingte Anweisungen koennen bis zu 248 Ebenen verschachtelt sein. 'VALUE' kann ein arithmetischer Ausdruck, ein anderes Symbol oder eine Zeichenkette sein.

```
IF        VALUE
IFNZ      VALUE
COND      VALUE
```

Die auf diese Steueranweisung folgenden Anweisungen bis zu einer ELSE- oder ENDIF-Steueranweisung werden uebersetzt, wenn der Wert von VALUE ungleich Null ist. Bedingte Anweisungen koennen bis zu 248 Ebenen verschachtelt sein.

```
IFTRUE    VALUE
IFNFALSE  VALUE
```

Diese Steueranweisung ist eigentlich die gleiche wie IFNZ, aber sie ist logischer, wenn man Vergleichsoperatoren verwendet. Wenn die angegebene Bedingung "wahr" ist, werden die nachfolgenden Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Ist die Bedingung "nicht wahr", werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung nicht assembliert.

```
IFNTRUE   VALUE
IFFALSE   VALUE
```

Diese Steueranweisung ist die gleiche wie IFZ und sie ist das Komplement zu IFTRUE. Wenn die angegebene Bedingung "falsch" ist, werden die nachfolgenden Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Ist die Bedingung "wahr", werden die Anweisungen bis zu einer ELSE oder ENDIF-Anweisung nicht assembliert.

```
IFDEF     LABEL
```

Auf diese Steueranweisung hin wird die Symboltabelle durchsucht. Wird LABEL gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Wird LABEL nicht gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE oder ENIF-Steueranweisung nicht assembliert.

```
IFNDEF    LABEL
```

Diese Steueranweisung ist das Komplement zu IFDEF. Die Symboltabelle wird durchsucht und wird LABEL nicht gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE oder ENIF-Anweisung assembliert. Wird LABEL gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE- oder ENIF-Steueranweisung nicht assembliert.

```
IFSAME    STRING1,STRING2
IFNDIFF
```

Diese Steueranweisung vergleicht STRING1 mit STRING2 und assembliert bedingt, d.h. abhaengig vom Ergebnis des Vergleichs, die dieser Anweisung nachfolgenden Anweisungen. Sind die beiden Zeichenketten identisch, dann werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Sind die Zeichenketten nicht identisch, so werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung nicht assembliert. Es sind zwei Typen von Zeichenketten moeglich, entweder enthalten sie Leerzeichen oder sie enthalten keine. Jedoch muessen die miteinander zu vergleichenden Zeichenketten vom gleichen Typ sein. Wenn die Zeichenketten Leerzeichen enthalten, dann muss der Anfang und das Ende jeder Zeichenkette durch ein Apostroph gekennzeichnet werden, wobei in ihr enthaltene Apostrophe durch die Verwendung von zwei Apostrophen dargestellt werden muessen. Enthalten die Zeichenketten keine Leerzeichen, dann sind die Apostrophe nicht erforderlich. Diese Steueranweisung ist sehr nuetzlich fuer den Vergleich von Makro-Parameterargumenten. In beiden Faellen muessen die Zeichenketten voneinander durch Komma getrennt sein. Es folgen ein paar Beispiele fuer die Verwendung von IFSAME.

```
IFSAME    'test string','test string'
IFSAME    'LUDWIG''S','LUDWIG''S'
IFSAME    X,Y
```

Im ersten Beispiel oben enthalten die Zeichenketten Leerzeichen. Deshalb muessen sie in Apostrophe eingeschlossen sein. Beim zweiten Beispiel enthalten die Zeichenketten im Innern ein Apostroph, das durch zwei Apostrophe dargestellt wird. Beim dritten Beispiel koennte es sich um das Testen von Registern in einem Makro handeln, und da die Zeichenketten keine Leerzeichen enthalten, muessen sie auch nicht in Apostrophe eingeschlossen sein.

```
IFNSAME    STRING1,STRING2
IFDIFF
```

Diese Steueranweisung ist das Komplement zu IFSAME. Wenn die beiden Zeichenketten nicht identisch sind, werden die Anweisungen nach dieser Anweisung bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Sind die beiden Zeichenketten identisch, werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung nicht assembliert. Die Syntaxregeln fuer die Bildung dieser Zeichenketten sind

dieselben wie fuer IFSAME. Die Beispiele bei IFSAME zeigen die Anwendung dieser Steueranweisung.

```
IFEXT     LABEL
```

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wurde diese Marke als external deklariert, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

```
IFNEXT    LABEL
```

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wurde diese Marke nicht als external deklariert, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

```
IFABS     LABEL
IFNREL
```

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wenn diese Marke absolut ist (d.h. nicht verschiebbar), werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Externe Marken werden als verschiebbar betrachtet. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

```
IFREL     LABEL
IFNABS
```

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wenn diese Marke verschiebbar ist, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Externe Marken werden als verschiebbar betrachtet. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

```
IFMA      EXP
```

Diese Steueranweisung ist fuer die Verwendung im Innern eines Makros bestimmt. Sie ueberprueft, ob die dem Wert EXP entsprechende Argumentnummer in der Makroaufrufzeile enthalten ist. Wenn das entsprechende Argument vorhanden ist, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Ist das Argument nicht vorhanden, werden die nachfolgenden Anweisungen bis zu einem ELSE oder ENDIF nicht assembliert. Soll das Nichtvorhandensein von Argumenten gepreuft werden, wird EXP = 0 verwendet. In diesem Fall werden, sofern in dem Makroaufruf keine Argumente vorhanden sind, die nachfolgenden Anweisungen assembliert. Sind in der Makroaufrufzeile jedoch Argumente angegeben, werden die nachfolgenden Anwei-

sungen nicht assembliert. Beispiele fuer die Verwendung dieser Steueranweisung sind im Abschnitt ueber Makros in diesem Handbuch zu finden.

IFNMA EXP

Diese Steueranweisung ist das Komplement zu IFMA. Sie ueberprueft, ob die dem Wert EXP entsprechende Argumentnummer in der Makroaufrufzeile enthalten ist. Wenn das entsprechende Argument nicht vorhanden ist, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Ist das Argument vorhanden, werden die nachfolgenden Anweisungen bis zu einem ELSE oder ENDIF nicht assembliert. Soll lediglich das Vorhandensein von Argumenten geprueft werden, wird EXP = 0 verwendet. In diesem Fall werden, sofern in dem Makroaufruf wenigstens ein Argument vorhanden ist, die nachfolgenden Anweisungen assembliert. Sind in der Makroaufrufzeile jedoch keine Argumente angegeben, werden die nachfolgenden Anweisungen nicht assembliert. Beispiele fuer die Verwendung dieser Steueranweisung sind im Abschnitt ueber Makros in diesem Handbuch zu finden.

ELSE

Beginn der Anweisungen, die assembliert werden sollen, falls fuer irgendeine der obigen Steueranweisungen des Typs IF die Bedingung nicht erfuellt ist.

ENDC
ENDIF

Kennzeichnet das Ende eines bedingt zu assemblierenden Blockes. Stellt der Assembler fest, dass nicht fuer jedes IF ein ENDIF vorhanden ist, gibt er eine Fehlermeldung aus. Da rekursive Makros fast immer durch Steueranweisungen vom Typ IF gesteuert werden, ist es moeglich, dass man die IFCLEAR-Steueranweisung verwenden muss. Der Unterschied zwischen den beiden besteht darin, dass ENDIF immer ausgefuehrt wird, waehrend IFCLEAR nicht ausgefuehrt wird, wenn es sich im Innern eines bedingt zu assemblierenden Blockes befindet, fuer den die Bedingung nicht erfuellt ist.

IFCLEAR

Diese Steueranweisung erfuellt genau dieselbe Funktion wie ENDIF, nur wird sie nicht ausgefuehrt, wenn sie sich innerhalb eines bedingt zu assemblierenden Blockes befindet, fuer den die Bedingung nicht erfuellt ist. Diese Steueranweisung kann in rekursiven Makros verwendet werden, um Uebereinstimmung bei den IF-ENDIF Paaren zu erzielen. Auf diese Weise hat man die Moeglichkeit, das Makro schliesslich zu beenden und trotzdem den Vorteil wahrzunehmen, dass der Assembler das paarweise Vorhandensein von IF-ENDIF ueberprueft. Diese Steueranweisung kann zum gleichen Zweck verwendet werden, wenn ein Makro MACEXIT fuer einen fruehzeitigen Makroaustritt verwendet, denn fast immer wird das durch irgendeine IF-Steueranweisung eingeleitet.

1.8.5. Steuerung der Listenausgabe

LIST ON

Schaltet die Listenausgabe ein, falls beim Aufruf des Assemblers die Steueranweisung "List On/Off" (Listenausgabe ein/aus) eingegeben wurde. Diese Steueranweisung muss immer verwendet werden, bevor man LIST OFF benutzt. Mit anderen Worten, beim Start des Programms wird immer LIST OFF vorausgesetzt.

LIST OFF

Schaltet die Listenausgabe aus, falls die Steueranweisung "List On/Off" (Listenausgabe ein/aus) eingegeben und LIST ON ausgefuehrt wurde. Das ist der voreingestellte Standardmodus und deshalb sollte diese Steueranweisung nur verwendet werden, wenn vorher schon ein LIST ON gegeben wurde.

MACLIST ON

Schaltet die Listenausgabe fuer Makroeinfuegungen ein. Das ist der voreingestellte Standardmodus.

MACLIST OFF

Schaltet die Listenausgabe fuer Makroeinfuegungen aus. Durch Voreinstellung ist sie eingeschaltet.

CONDLIST ON

Schaltet die Listenausgabe fuer bedingt zu assemblierende Bloecke, deren Bedingung nicht erfuellt ist, ein. Das ist der voreingestellte Standardmodus.

CONDLIST OFF

Schaltet die Listenausgabe fuer bedingt zu assemblierende Bloecke , deren Bedingung nicht erfuellt ist, aus. Durch Voreinstellung ist sie eingeschaltet.

ASCLIST ON

Schaltet die Listenausgabe von ASCII-Zeichenketten, die mehr als eine Objektkodezeile auf der Assemblerliste benoetigen, ein.

ASCLIST OFF

Schaltet die Listenausgabe von ASCII-Zeichenketten, die mehr als eine Objektkodezeile auf der Assemblerliste benoetigen, aus. Nur die erste Zeile des Objektkodes wird ausgegeben.

```
PW      EXP
```

Legt die Breite der Druckerseite fest. Die voreingestellte Breite einer Seite betraegt 132 Spalten.

```
PL      EXP
```

Legt die Laenge der Druckerseite fest. Die voreingestellte Laenge einer Seite betraegt 61 Zeilen. Der Assembler loest einen Seitenvorschub aus, wenn dieser Grenzwert erreicht oder ueberschritten wird. Wenn ein Fehler auftritt, gibt der Assembler den Seitenvorschub nach der Fehlermeldung aus.

```
TOP     EXP
```

Diese Steueranweisung bestimmt die Zeilenanzahl zwischen dem Seitenbeginn und der Seitenzahl. Der voreingestellte Wert betraegt Null.

```
PASS1   ON
```

Schaltet die Listenausgabe fuer den 1.Durchlauf ein. Man kann diese Moeglichkeit nutzen, um Fehler zu finden, die dadurch entstanden sind, dass der Assembler beim ersten Durchlauf einen anderen Weg genommen hat, als beim zweiten. Gewoehnlich wird dieser Umstand zu dem Fehler 'Symbolwert zwischen den Durchlaeufen gewechselt' fuehren. Die Steueranweisung ist auch von Nutzen, wenn Fehler bei verschachtelter bedingter Assemblierung gefunden werden sollen.

```
PASS1   OFF
```

Schaltet die Listenausgabe fuer den 1.Durchlauf aus, vorausgesetzt, es wurde vorher PASS1 ON ausgefuehrt.

```
PAG
PAGE
EJECT
```

Gibt einen Seitenvorschub an das Listenausgabegeraet aus.

```
NAM     STRING
TTL
TITLE
HEADING
```

Veranlasst, dass 'STRING' am Anfang jeder Seite als Titelzeile gedruckt wird. Wird 'STRING' nicht angegeben, so wird die 'TITLE'-Steueranweisung ausgeschaltet. Der Titel kann so oft wie gewuenscht geaendert und zu jedem Zeitpunkt ausgeschaltet werden. Die maximale Titellaenge betraegt 80 Zeichen. Die ersten beiden Tabulatoren zwischen der 'TITLE'-Steueranweisung und der STRING-Zeichenkette werden, sofern sie vorhanden sind, ignoriert. Alle nachfolgenden Leerzeichen und Tabulatoren bleiben in der Titelzeile enthalten.

```
STTL      STRING
SUBTITLE
```

Veranlasst, dass 'STRING' am Anfang jeder Seite gedruckt wird. Wenn 'TITLE' ausgefuehrt wurde, wird der Untertitel darunter erscheinen. Wenn 'TITLE' nicht ausgefuehrt wurde oder ausgeschaltet wurde, wird der Untertitel dennoch ausgegeben. Wird 'STRING' nicht angegeben, so wird die Steueranweisung ausgeschaltet. Der Untertitel kann so oft wie gewuenscht geaendert und zu jedem Zeitpunkt ausgeschaltet werden. Die maximale Laenge des Untertitels betraegt 80 Zeichen. Wie bei der 'TITLE'-Steueranweisung werden die ersten beiden Tabulatoren zwischen der SUBTITLE-Steueranweisung und dem Anfang der 'STRING'-Zeichenkette, sofern sie vorhanden sind, ignoriert. Alle danach erscheinenden Leerzeichen oder Tabulatoren bleiben in der Untertitelzeile enthalten.

1.8.6. Steueranweisungen fuer den Linker

```
OPTIONS        OPTION LIST
```

Mit dieser Steueranweisung werden die Optionen fuer den Linker ausgewaehlt. Eine Liste der moeglichen Optionen kann dem Abschnitt ueber die Linkeroptionen des Handbuches entnommen werden. Der standardmaessig voreingestellte Ausgabedateityp ist 'I-Hex'. Die Ausgabe vom Linker kann durch die Verwendung des Linker-Optionenfeldes noch veraendert werden.

```
FILLCHAR       VALUE
```

Die durch die Verwendung von Abschnitten (Sections) oder ORG-Steueranweisungen entstandenen Luecken werden vom Linker mit dem fuer 'VALUE' angegebenen Wert ausgefuellt.

```
RECSIZE        VALUE
```

Mit dieser Steueranweisung kann die Recordlaenge fuer 'I-Hex'- und 'Moto-S'-Ausgaberecords veraendert werden. Die standardmaessig festgelegten 32 Datenbytes fuer 'I-Hex' und 131 Datenbytes fuer 'Moto-S' werden durch den fuer 'VALUE' festgelegten Wert ersetzt.

```
SYMBOLS
```

Damit koennen die Symbole an eine Ausgabedatei fuer den Linker gesendet werden. Der Linker gibt die 'Mic'-Symboltabelle aus, wenn diese Steueranweisung verwendet wird.

1.9. Arithmetische und logische Operatoren

Die nachfolgende Liste enthaelt die zulaessigen arithmetischen und logischen Operatoren fuer die Berechnung von Ausdruecken. Dazu wird auch ihre Prioritaetsebene angegeben. Operationen der Prioritaetsebene 7 werden als erste ausgefuehrt. Durch die Verwendung von Klammern kann die Reihenfolge bei der Ausfuehrung von Berechnungen veraendert werden. Alle Berechnungen koennen unter Verwendung einer 16- oder 32-Bit-Integer-Arithmetik ausgefuehrt werden, mit Ausnahme der Potenzierung, die nur einen 8-Bit-Exponenten verwendet. Welche Art der Kontrolle auf Ueberlauf durchgefuehrt wird, haengt von dem Modus ab, in dem der Assembler gerade arbeitet. Der naechste Abschnitt enthaelt hierzu mehr Informationen. Die maximale Anzahl fuer noch andauernde Operationen ist 16.

OPERATION	PRIORITAET	BESCHREIBUNG
Unaeres +	7	Optionale Kennzeichnung eines positiven Operanden
Unaeres -	7	Macht den folgenden Ausdruck negativ
\ oder .NOT.	7	Bildet das Komplement des folgenden Ausdrucks
Unaeres >	7	Verwendet das hoeherwertige Byte der nachfolgenden Adresse. Muss verwendet werden, um verschiebbare Byte-Adresswerte zu erhalten.
Unaeres <	7	Verwendet das niederwertige Byte der nachfolgenden Adresse. Muss verwendet werden, um verschiebbare Byte-Adresswerte zu erhalten.
**	6	Vorzeichenlose Potenzierung
*	5	Vorzeichenlose Multiplikation
/	5	Vorzeichenlose Division
.MOD.	5	Rest
.SHR.	5	Verschiebt den vorhergehenden Ausdruck so oft nach rechts (mit 0 aufgefuellt), wie in dem nachfolgenden Ausdruck angegeben ist.
.SHL.	5	Verschiebt den vorhergehenden Ausdruck so oft nach links (mit 0 aufgefuellt), wie in dem nachfolgenden Ausdruck angegeben ist.
+	4	Addition
-	4	Subtraktion
& oder .AND.	3	Logisches UND
^ oder .OR.	2	Logisches ODER
.XOR.	2	Logisches EXCLUSIVES ODER

1.10. Vergleichsoperatoren

Die folgende Liste enthaelt die zulaessigen Vergleichsoperatoren. Wird die Vergleichsbedingung erfuellt, so ist das Ergebnis des Vergleichs 1. Ist der Vergleich falsch, d.h. die Bedingung wird nicht erfuellt, so ist das Vergleichsergebnis 0 :

```
=  oder .EQ.  -  gleich
>  oder .GT.  -  groesser als
<  oder .LT.  -  kleiner als
   .UGT.      -  vorzeichenlos groesser als
   .ULT.      -  vorzeichenlos kleiner als
```

1.11. Makros

1.11.1. Definition

Ein Makro ist eine Folge von Quellprogrammzeilen, die fuer eine einzelne Quellprogrammzeile eingesetzt werden. Ein Makro muss definiert werden, bevor es verwendet werden kann. Der Assembler speichert die Makrodefinition und setzt dann, beim Auftauchen des Makronamens, die vorher definierten Quellprogrammzeilen ein. Argumente koennen in der Makrodefinition enthalten sein. Argumente koennen in jedem Feld, ausser dem Kommandofeld, erscheinen.

In Makrodefinitionen duerfen die Formalargumente keine Leerzeichen enthalten. In den eigentlichen Makroaufrufen koennen die Argumente von jedem Typ sein: direkt, indirekt, Zeichenketten oder Register. Leerzeichen sind in Argumenten nicht erlaubt, es sei denn, es ist eine ASCII-Zeichenkette. In diesem Fall muss die Zeichenkette in Apostrophe eingeschlossen sein. Enthaelt die Zeichenkette selbst ein Apostroph, so wird dies durch zwei nebeneinander stehende Apostrophe dargestellt. Argumente werden an beliebig verschachtelte Makros uebergeben, wenn die Namen der Formalargumente identisch sind. Die Verschachtelung von Makros wird nur durch den verfuegbaren Speicherplatz begrenzt.

Um ein Makro zu definieren, wird die ".MACRO"-Steueranweisung verwendet. Ein Makro muss, anschliessend an die Makrodefinition, die Steueranweisung ".MACEND" oder ".ENDM" enthalten. Der Name des Makros steht im Markenfeld.

1.11.2. Trennzeichen fuer Argumente

In der Makroaufrufzeile muessen die Argumente voneinander durch Komma getrennt werden. Fuehrende Leerzeichen und Tabulatoren werden ignoriert. Fuer ein nicht vorhandenes Argument kann ein einzelnes Komma als Platzhalter dienen. Das Zeichen '*' als Argument wird nicht als Befehlszaehler, sondern als Multiplikationszeichen verwendet. In einem Makrokoerper sind die folgenden Trennzeichen fuer Argumente zulaessig:

, + - * / ** \ & ^ = < > ( ) [ ] |

.NOT. .AND. .OR. .XOR. .EQ. .GT. .LT. .UGT.

.ULT. .SHR. .SHL.

1.11.3. Verkettung

Der Senkrechtstrich (| = hex 7C) wird als Verkettungsoperator fuer Zeichenketten verwendet. Verkettungen duerfen nur im Innern eines Makros ausgefuehrt werden.

1.11.4. Marken in Makros

Marken sind in Makrodefinitionen zugelassen. Marken koennen auf zwei Arten definiert sein: explizit oder implizit. Explizite Marken werden in der Makrodefinition durch den Assembler nicht veraendert. Implizite Marken haben am Ende ein "#" stehen. Der Assembler ersetzt "#" durch eine dreistellige Zahl, die Makroerweiterungsnummer. In diesem Fall darf die Marke mit der Makroerweiterungsnummer nicht laenger als 32 Zeichen sein. Endet eine Marke, die sich ausserhalb eines Makros befindet, mit dem Symbol "#", so erhaelt sie die Makroerweiterungsnummer des letzten Makros. Damit ist es moeglich, Marken umzubenennen, ohne die tatsaechliche Makroerweiterungsnummer zu kennen. Ein Argument kann als Bezeichnung fuer eine Marke benutzt werden.

1.11.5. Umdefinieren von Mnemoniks

Die Assemblertabellen werden in der folgenden Reihenfolge durchsucht:

1. Mnemoniktabelle
2. Tabelle der Makrodefinitionen
3. Tabelle der Assembler-Steueranweisungen
4. Tabelle der Abschnittsnamen (Sections)

Um ein Mnemonik umzudefinieren, kann man die MACFIRST-Steueranweisung verwenden. Damit wird die Suchreihenfolge vertauscht, so dass zuerst die Tabelle der Makrodefinitionen und dann die Mnemoniktabelle durchsucht wird.

1.11.6. Makro-Beispiel

Mit dem nachfolgenden Makro kann ein Zeichenkettenvergleich durchgefuehrt werden.

```
CMP_STRING:    .MACRO    ARG1
               IFNMA      1
               CMP_STRING NEEDS AN ARGUMENT
               MACEXIT
               ENDIF
               IFSAME     "JANUARY",ARG1
```

```
MONTH           BYTE        1
                MACEXIT
                ENDIF
                IFSAME      "FEBRUARY",ARG1
MONTH           BYTE        2
                MACEXIT
                ENDIF
                IFSAME      "MARCH",ARG1
MONTH           BYTE        3
                MACEXIT
                ENDIF
                IFSAME      "APRIL",ARG1
MONTH           BYTE        4
                MACEXIT
                ENDIF
                IFSAME      "MAY",ARG1
MONTH           BYTE        5
                MACEXIT
                ENDIF
                IFSAME      "JUNE",ARG1
MONTH           BYTE        6
                MACEXIT
                ENDIF
                ARGUMENT ERROR IN MACRO STRING
                ENDM
                CMP_STRING      "APRIL"
                END
```

Das folgende Beispiel zeigt, wie die Substitution von Argumenten, die im Operandenfeld des Makros angegeben werden, durchgefuehrt wird.

```
EMPLOYEE_INFO: .MACRO    ARG1,ARG2,ARG3
NAME:          DB        ARG1
DEPARTMENT:    ASCII     ARG2
DATE_HIRED:    LONG      ARG3
               .ENDM
               EMPLOYEE_INFO  'JOHNN DOE',PERSONNEL,101085
               END
```

Das letzte Beispiel kann so veraendert werden, dass die Argumente in das Markenfeld gebracht werden. Das gestattet es, die Struktur zu veraendern.

```
EMPLOYEE_INFO: .MACRO    ARG1,ARG2,ARG3
ARG1:          DS        30H
ARG2:          DS        10H
ARG3:          LONG
               .ENDM
               EMPLOYEE_INFO  NAME,DEPARTMENT,DATE_HIRED
               END
```

Mit einem Makro koennen auch Substitutionen im Mnemonikfeld vorgenommen werden. Auch ist es moeglich, mit dem Zeichen '#', eine Marke innerhalb eines Makros zu erzeugen.

```
INSTRUCTION:    .MACRO    ARG,VAL
                ARG
LAB#:           DS        VAL
                .MACEND
                INSTRUCTION    NOP,7
                END
```

Um ein Mnemonik neu zu definieren, muss die Steueranweisung MACFIRST ON dem Makro vorausgehen.

```
                MACFIRST ON
NOP:            .MACO     ARG
                DB        ARG
                .ENDM
                NOP       FFH
                END
```

1.11.7. Rekursive Abarbeitung

Wenn Makros verschachtelt werden, so werden alle Informationen, die fuer die Rueckkehr in die vorhergehende Quellenkodeumgebung benoetigt werden, auf dem Disk-Speicher gerettet. Diese Verfahrensweise macht es dem Assembler moeglich, Makros auf dem Disk-Speicher zu speichern und Verschachtelungen bis zu jeder gewuenschten Tiefe zu gestatten. Rekursionen (ein Makro ruft sich selbst auf) sind erlaubt. Sie werden in der Weise realisiert, dass festgestellt wird, dass das Makro bereits aktiv ist und daraufhin das Abspeichern der vorhergehenden Quellenkodeumgebung verhindert wird. Jedoch ist es nicht gestattet, dass ein Makro ein anderes Makro aufruft, welches wiederum das erste Makro aufruft. Wenn das geschieht, wird die urspruengliche Rueckkehrinformation zerstoert und der Assembler ist nicht mehr in der Lage, in die urspruengliche Umgebung zurueckzukehren. Darueber hinaus ist es dem Assembler auch nicht moeglich, diese Bedingung festzustellen, weil er, wenn das zweite Makro das erste, das ja schon aktiv ist, aktiviert, einen rekursiven Makrozustand vorraussetzt und die Informationen, die zur Rueckkehr zum zweiten Makro noetig waeren, nicht aufzeichnet.

Unten folgt ein Beispiel fuer ein rekursives Makro, das eine Anzahl von Datenbytes, die durch die Formalargumente ARG2, ARG3, ARG4, ARG5 und ARG6 gegeben sind, so oft im Speicher ablegt, wie ARG1 es vorsieht.

```
RESERVE:  .MACRO    ARG1,ARG2,ARG3,ARG4,ARG5,ARG6
COUNT:    .VAR      ARG1            ;Speichere Anzahl
          .IFZ      COUNT           ;Kontrolle der Ausfueh-
                                     rung
          .IFCLEAR                  ;Fuer paarweises IF-ENDIF
                                     sorgen
          .MACEXIT                  ;Austritt
```

```
          .ENDIF
COUNT:    .VAR      COUNT-1         ;Else dekrementiere Byte-
                                     zaehler
          .BYTE     ARG2,ARG3,ARG4,ARG5,ARG6
                                    ;Speichert die Bytes
          .RESERVE  COUNT,ARG2,ARG3,ARG4,ARG5,ARG6
                                    ;Erneuter Durchlauf
          .MACEND
```

Dieses Makro wuerde man mit einer Anweisung wie der folgenden aufrufen:

```
          RESERVE      10,AH,BH,CH,DH,EH,
                                    ;Fuelle 50 Bytes mit der
                                     Folge ABCDE
```

Fuer ein rekursives Makro, wie beispielsweise das oben beschriebene, ist es durchaus zulaessig, ein weiteres rekursives Makro aufzurufen usw., bis zu jeder gewuenschten Tiefe. Man beachte auch die Verwendung der IFCLEAR-Steueranweisung, die die Paarigkeit der bedingten IF- ENDIF-Steueranweisungen aufrechterhaelt. Das ist erforderlich fuer den Fall, dass die MACEXIT-Steueranweisung ausgefuehrt wird, da dann die ENDIF-Steueranweisung nicht ausgefuehrt wird.

1.12. Assembler-Fehlernachrichten

Fehler - SYNTAX ERROR
(Syntaxfehler)
Bedeutung - Gewoehnlich ein fehlendes Komma oder eine fehlende Klammer.

Fehler - CAN'T RESOLVE OPERAND
(Operand kann nicht aufgeloest werden)
Bedeutung - Es ist nicht erkennbar, was der Programmierer beabsichtigte.

Fehler - ILLEGAL ADDRESSING MODE
(Unzulaessige Adressierungsart)
Bedeutung - In dieser Form kann der Operand nicht adressiert werden.

Fehler - ILLEGAL ARGUMENT
(Unzulaessiges Argument)
Bedeutung - Der Operand kann hier nicht verwendet werden.

Fehler - MULTIPLY DEFINED SYMBOL
(Mehrfach definiertes Symbol)
Bedeutung - Das Symbol wurde vorher definiert ('.VAR' ist davon ausgenommen).

Fehler - ILLEGAL MNEMONIC
(Unzulaessige Mnemonik)
Bedeutung - Mnemonik existiert nicht und wurde nicht als Makro definiert.

Fehler - # TOO LARGE
(Zahl ist zu gross)
Bedeutung - Der Bestimmungsort ist zu klein fuer den Operanden.

Fehler - ILLEGAL ASCII DESIGNATOR
(Unzulaessige ASCII-Kennzeichnung)
Bedeutung - Schlechte Zeichensetzung bei der Darstellung von ASCII-Zeichen.

Fehler - HEX # AND SYMBOL ARE IDENTICAL
(Hexadezimalzahl und Symbol sind identisch)
Bedeutung - Es existiert eine Marke, die voellig identisch mit einer als Operand verwendeten Hexadezimalzahl ist. Der Fehler tritt nur auf, wenn sich auch das zur Kennzeichnung der Hexadezimalzahl verwendete 'H' oder '%' an der gleichen Stelle befindet.

Fehler - UNDEFINED SYMBOL
(Undefiniertes Symbol)
Bedeutung - Das Symbol wurde waehrend des 1.Durchlaufs nicht definiert.

Fehler - RELATIVE JUMP TOO LARGE
(Relativer Sprung zu gross)
Bedeutung - Zieladresse auf einer anderen Seite

Fehler - EXTRA CHARACTERS AT END OF OPERAND
(Zusaetzliche Zeichen am Ende des Operanden)
Bedeutung - Gewoehnlich ein Syntax- oder Formatfehler.
Hinweis - Dieser Fehler wird bei der letzten Kontrolle, die der Assembler an jedem Befehl vornimmt, bevor er zur naechsten Zeile geht, festgestellt. Er zeigt an, dass nach dem gueltigen Ende des Operanden zusaetzliche Zeichen vorhanden sind.

Fehler - LABEL VALUE CHANGED BETWEEN PASSES
(Markenwert zwischen den Durchlaeufen veraendert)
Bedeutung - Der im ersten Durchlauf ermittelte Symbolwert entspricht nicht dem im zweiten Durchlauf ermittelten.
Hinweis - Dieser Fehler wird gewoehnlich dadurch verursacht, dass der Assembler, infolge von ver-

aenderten Argumentwerten fuer bedingte Steueranweisungen, beim ersten Durchlauf einen anderen Weg waehlt, als beim zweiten. Die Steueranweisung PASS1 ON/OFF kann fuer das Auffinden dieser Art von Fehlern nuetzlich sein.

Fehler - ATTEMPTED DIVISION BY ZERO
(Versuchte Division durch Null)
Bedeutung - Der Operand fuer den Divisor hat den Wert 0.

Fehler - ILLEGAL EXTERNAL REFERENCE
(Unzulaessiger externer Bezug)
Bedeutung - Hier kann kein externer Bezug erfolgen.

Fehler - NESTED CONDITION. ASSEMBLY UNBALANCE DETECTED
(Unpaarigkeit in verschachtelter bedingter Assemblierung festgestellt)
Bedeutung - Beliebiger '.IF'-Typ-Befehl ohne das dazugehoerige '.ENDIF'.

Fehler - MACRO STACK OVERFLOW
(Makro-Stack Ueberlauf)
Bedeutung - Zu tief verschachtelte Makros.
Hinweis - Dieser Fehler kann durch zu viele rekursive Makroaufrufe verursacht werden. Der Stack sieht fuer etwa 700 verschachtelte oder rekursive Makroaufrufe Platz vor. Die Anzahl der Aufrufe ist von der Anzahl der vom Makro verwendeten Argumente abhaengig.

Fehler - ILLEGAL REGISTER
(Unzulaessiges Register)
Bedeutung - Das angegebene Register ist fuer diese Anweisung nicht zulaessig.

Fehler - CANT RECOGNIZE NUMBER BASE
(Zahlenbasis nicht erkennbar)
Bedeutung - Die angegebene Zahlenbasis ist keine von denen, die der Assembler akzeptiert.

Fehler - NOT ENOUGH PARAMETERS
(Nicht genuegend Parameter)
Bedeutung - Es wurden mehr Argumente angegeben als das Makro Parameter hat.

Fehler - ILLEGAL LABEL 1ST CHARACTER
(Unzulaessiges erstes Zeichen einer Marke)
Bedeutung - Marken muessen mit einem Alphazeichen beginnen.

Fehler - MAXIMUM EXTERNAL SYMBOL COUNT EXCEEDED
(Maximalanzahl fuer externe Symbole ueberschritten)
Bedeutung - Der Modul enthaelt zu viele Externals.
Hinweis - Es gibt ein Maximum von annaehernd 5000 Externals je Modul.

Fehler - ILLEGAL REGISTER SIZE
(Unzulaessige Registergroesse)
Bedeutung - Die angegebene Registergroesse ist nicht erlaubt.

Fehler - ILLEGAL SCALE FACTOR
(Unzulaessiger Skalenfaktor)
Bedeutung - Der Skalenfaktor ist nicht zulaessig (Nur 1, 2, 4 oder 8).

Fehler - ILLEGAL INSTRUCTION SIZE
(Unzulaessige Befehlsgroesse)
Bedeutung - Der Befehl erlaubt nicht die Verwendung der angegebene Groesse.

Fehler - ILLEGAL SIZE SPECIFIER
(Unzulaessiger Groessenbezeichner)
Bedeutung - Der Befehl gestattet nicht, eine Groesse anzugeben.

Fehler - Dh:D1 CANNOT BE THE SAME
(Dh:D1 kann nicht dasselbe sein)
Bedeutung - Der Befehl liefert ein 64-Bit-Ergebniss in 2 Registern.

Fehler - MUST BE IN SAME SECTION
(Muss im gleichen Abschnitt liegen)
Bedeutung - Der Operand des Befehls befindet sich in einem anderen Abschnitt.

Fehler - MUST USE LONG OFFSET
(Ein langer Offset muss verwendet werden)
Bedeutung - Die angegebene Offsetgroesse ist nicht gross genug.

Fehler - NON-EXISTANT INCLUDE FILE
(Nicht existierende INCLUDE-Datei)
Bedeutung - Die mit INCLUDE-Steueranweisung einzubeziehende Datei konnte nicht gefunden werden.

Fehler - ILLEGAL NESTED INCLUDE
(Unzulaessige Verschachtelung bei INCLUDE)
Bedeutung - Eine durch INCLUDE-Steueranweisung einbezo-

gene Datei enthaelt eine .INCLUDE-Steueranweisung. Dieser Fehler kann auch darauf hinweisen, dass eine durch INLUDE einbezogene Datei keine END-Anweisung enthaelt.

Fehler - NESTED SECTION UNBALANCE
(Nicht abgeschlossene Verschachtelung von Abschnitten)
Bedeutung - Eine verschachtelte Abschnittsdefinition ohne ein ENDS.

2. Linker

2.1. Beschreibung des Linkers

Der Linker ermoeglicht dem Anwender, Programme in der U880-Assemblersprache zu schreiben, die aus mehreren Modulen bestehen. Der Linker loest externe Adressbezuege auf und fuehrt Adressverschiebungen durch. Der Linker ist in der Lage, alle ueblichen Dateiformate zu erzeugen. Dadurch entfaellt die Notwendigkeit einer zusaetzlichen Formatkonvertierung.

Abgesehen von dem Fall, dass eine ausfuehrbare Ausgabedatei erzeugt werden soll, laeuft der Linker vollstaendig im RAM. Fuer die zu linkende Datei gibt es keine Beschraenkung in der Groesse, solange genuegend Speicherplatz zur Verfuegung steht. Soll eine ausfuehrbare Datei erzeugt werden, legt der Linker so viele temporaere Dateien an, wie erforderlich sind, um die verschiedenen Programmabschnitte in aufsteigender Reihenfolge zu sortieren.

Jede Objektdatei kann bis zu 256 verschiedene, vom Nutzer definierte Abschnitte, enthalten. Insgesamt kann der Linker 256 Eingabedateien und 256 verschiedene Abschnittsnamen bearbeiten. Angenommen, jede Eingabedatei enthaelt 256 Abschnitte (wobei in jeder Datei die gleichen Abschnittsnamen gewaehlt werden muessen), dann betraegt die maximale Kapazitaet des Linkers 65536 verschiedene Abschnitte. Dabei gibt es fuer die einzelnen Abschnitte keine Groessenbeschraenkungen.

Der Linker kann unter Verwendung des "Prompt-Modus", des "Data-File-Modus" oder des "Kommandozeilen-Modus" aufgerufen werden. Das Ausgabeformat wird durch eine Steueranweisung im Quellprogramm oder durch die Optionsliste des Linkers ausgewaehlt. Die Ladetabelle, eine alphabetische Liste aller globalen Symbole, sowie alle Linkfehler koennen in einer Datei auf dem Disk-Speicher abgelegt werden.

Der Linker kann angewiesen werden, mehrere verschiedene Typen von Symboltabellen-Dateien auszugeben. Als Formate sind moeglich: globale Symbole mit einer Laenge von 10 Zeichen, globale Symbole mit einer Laenge von 32 Zeichen sowie das 'Mic'-Format, das alle Symbole einschliesst.

2.2. Linker Bedienungsanweisungen

2.2.1. Prompt-Modus

Um den Linker im "Prompt-Modus" laufen zu lassen, wird 'Link' eingegeben. Das Programm antwortet mit der Anforderung des Namens einer Eingabedatei. Die Standardnamenserweiterung einer Eingabedatei fuer den Linker ist 'obj'. Nachdem der Linker die Objektdatei eroeffnet hat, fordert er die Offsetadresse jedes Programmabschnitts, der eine Laenge ungleich Null hat, an. Dieser Offsetwert wird zum Wert der in der Datei vorhandenen ORG-Steueranweisungen

addiert. Wird auf die Eingabeanforderung nur ein CR eingegeben, so schliesst der Linker den Programmabschnitt unmittelbar an den vorhergehenden an. Ein Minuszeichen veranlasst den Linker, den entsprechenden Abschnitt mit zu binden, aber er nimmt ihn nicht in die Ausgabedatei auf. Wird nach einer Offsetadresse ein Semikolon eingegeben, veranlasst das den Linker, jeden weiteren Abschnitt unmittelbar an den vorhergehenden anzuschliessen. Am besten laesst sich das alles anhand von Beispielen erklaeren. Deshalb sei hier auf den Abschnitt Linker Beispiele (2.5.) verwiesen.

Der Eingabevorgang kann beendet werden, indem auf die Anforderung des Eingabedateinamens nur mit einem CR geantwortet wird. Daraufhin fordert der Linker zur Eingabe des Ausgabedateinamens auf. Wird auch hier nur CR eingegeben, erzeugt der Linker eine Ausgabedatei, die denselben Namen wie die erste Eingabedatei traegt mit einer dem Ausgabedateityp entsprechenden Namenserweiterung.

Nachdem der Ausgabedateiname eingegeben wurde, fordert der Linker zur Eingabe der Optionen auf. Die Linker Optionen werden im Abschnitt 2.3. beschrieben.

2.2.2. Data-File-Modus

Der "Data-File-Modus" ist fuer grosse oder komplexe Linkvorgaenge vorgesehen. Dieser Modus kann als identisch zum "Prompt-Modus" angesehen werden, mit Ausnahme, dass alle Antworten auf Eingabeanforderungen in einer Datendatei untergebracht werden, die dem Linker zur Verfuegung gestellt wird.

Der Aufruf erfolgt mit dem folgenden Kommando:

```
link data_file
```

Das veranlasst den Linker, die Datei 'data_file.lnk' zu lesen und die Antworten aus der Datendatei Zeile fuer Zeile zu verwenden. Fuer die Datendatei erwartet der Linker die Namenserweiterung 'lnk'. Ist als Antwort auf eine Eingabeanforderung nur ein CR vorgesehen, so ist dieses moeglicherweise schwierig in der Datendatei zu erkennen. Deshalb wird in eine Zeile, in der nur ein CR stehen soll, ein Unterstrich ('_') eingetragen. Sollen Linker Optionen angegeben werden, so werden sie am Ende der Datendatei angegeben, genau wie das im "Prompt-Modus" geschieht. Die nachfolgend als Beispiel angegebene Datendatei soll zwei Dateien miteinander verbinden, wobei der Abschnitt CODE bei 2000H und der Abschnitt DATA bei 4000H beginnen soll. Es soll der Standardname fuer die Ausgabedatei des Linkers verwendet werden. Die Optionen 'D' und '3' werden benutzt, um auf dem Disk-Speicher eine Mapdatei und eine Ausgabedatei im Format 'Moto-S37' zu erzeugen (s. Abschn. 2.3.).

```
file1          Name der ersten Eingabedatei
2000           CODE-Abschnitt beginnt bei 2000H
4000           DATA-Abschnitt beginnt bei 4000H
```

file2	Name der zweiten Eingabedatei
_	Der CODE-Abschnitt soll unmittelbar an den ersten CODE-Abschnitt anschliessen
_	Der DATA-Abschnitt soll unmittelbar an den ersten DATA-Abschnitt anschliessen
_	Keine weiteren Eingabedateinamen
_	Fuer Ausgabedatei soll der Standardname verwendet werden
D3	Mapdatei und 'Moto-S37'-Datei auf Disk-Speicher erzeugen

Die leichteste Art und Weise, eine Datendatei zu erzeugen ist die, dass man den Linkprozess im "Prompt-Modus" ausfuehrt und alle Eingaben notiert. Dann erzeugt man mit Hilfe eines Texteditors eine Datei, in der jede Eingabe auf einer eigenen Zeile steht. Die Datei muss die Namenserweiterung 'lnk' erhalten.

2.2.3. Kommandozeilen-Modus

Der Linker kann auch unter Verwendung einer Kommandozeile aufgerufen werden. Die Form des Kommandos wird nachfolgend gezeigt. Optionale Angaben sind in Klammern eingeschlossen.

```
link [-q] -c file1 [-lnnnn] file2 [-lnnnn] ... [-ofile]
                                                [-options]
```

Die Option -q bringt den Linker in den 'Quiet'-Modus. In diesem Fall werden nur die Linkfehler ueber das Terminal ausgegeben.

Die Option -c ist notwendig. Sie informiert den Linker, dass die Abarbeitung im "Kommandozeilen-Modus" und nicht im "Data-File-Modus" erfolgt.

Der Option -c folgt die Liste der Eingabedateien, die in der obigen Kommandozeile aus 'file1' und 'file2' besteht. Nach jeder Eingabedatei kann unter Verwendung der Option -l eine Offsetadresse angegeben werden. Wird keine Offsetadresse angegeben, so wird die Datei unmittelbar an die vorhergehende angeschlossen, wobei eine Zuordnung zu den entsprechenden Abschnitten erfolgt.

Die Option -o kann verwendet werden, um einen Ausgabedateinamen festzulegen. Diese Angabe ist optional. Wird kein Ausgabedateiname angegeben, erzeugt der Linker eine Ausgabedatei mit dem Namen der ersten Eingabedatei und einer dem festgelegten Ausgabeformat entsprechenden Namenserweiterung.

Im Optionenfeld koennen beliebige Linkeroptionen angegeben werden. Der Liste muss ein Minuszeichen vorausgehen. Es koennen beliebig viele Optionen angegeben werden. Im nachfolgenden Abschnitt werden die Linkeroptionen ausfuehrlich beschrieben.

2.3. Linker Optionen

Im "Prompt-Modus" erfolgt die Aufforderung zur Eingabe der Optionen nach Eingabe des Ausgabedateinamens. Die unten angegebenen Optionen koennen auch im "Kommandozeilen-Modus" sowie im "Data-File-Modus" verwendet werden. Werden mehr als eine Option eingegeben, so gilt die zuletzt eingegebene Option vor den frueher eingegebenen Optionen.

Optionen (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=Standard)

D - Erzeugt eine Datei auf dem Disk-Speicher, die aufgetretene Linkfehler, eine alphabetische Tabelle der globalen Symbole und die Ladetabelle (Load Map) enthaelt. Diese Datei traegt den gleichen Namen wie die Ausgabedatei des Linkers mit der Namenserweiterung 'map'.

S - Erzeugt eine Symboldatei, die zur Programmtestung mit einem Debugger verwendet werden kann. Die Datei enthaelt alle globalen Symbole und deren Adressen. Jedes Symbol ist 32 Zeichen lang. Genauere Angaben hierzu werden im Abschnitt ueber das Ausgabeformat der Symboltabelle gemacht.

A - Erzeugt eine Symboldatei, die zur Programmtestung mit einem Debugger verwendet werden kann, begrenzt aber die Symbole auf die ersten 10 Zeichen. Kann aus Kompatibilitaetsgruenden zu frueheren Linkerversionen benutzt werden. Genauere Angaben hierzu werden im Abschnitt ueber das Ausgabeformat der Symboltabelle gemacht.

M - Erzeugt eine Symboltabelle im 'Mic'-Format, die zur Programmtestung mit einem Debugger verwendet werden kann. Diese Datei enthaelt alle Symbole, sowohl die lokalen als auch die globalen. Das Quellprogramm muss die Steueranweisung SYMBOLS enthalten, um dieses Format erzeugen zu koennen.

X - Erzeugt eine ausfuehrbare Ausgabedatei.

H - Erzeugt eine Ausgabedatei im 'I-Hex'-Format.

E - Erzeugt eine Ausgabedatei im Format 'Extended I-Hex'.

T - Erzeugt eine Ausgabedatei im Format 'T-Hex'.

1 - Erzeugt eine Ausgabedatei im Format 'Moto-S19'.

2 - Erzeugt eine Ausgabedatei im Format 'Moto-S28'.

3 - Erzeugt eine Ausgabedatei im Format 'Moto-S37'.

2.4. Adressenverschiebung

Adressen werden verschoben, indem die Offsetadresse zu dem durch die Assemblierung zugewiesenen Adresswert addiert wird. Normalerweise beginnt die Assemblierung eines Programms bei der Startadresse 0. Aber das muss nicht immer der Fall sein. Deshalb wird die Offsetadresse einfach zu jeder Adresse addiert, die der Assembler erzeugt hat.

Der Assembler erzeugt eine Tabelle, in der die Eigenschaften eines jeden im Programm verwendeten Symbols eingetragen werden. Handelt es sich nur um eine Marke, die vor einem Befehl steht, so ist sie als verschiebbar gekennzeichnet. Ist die Marke in einer ".EQUAL"-Steueranweisung definiert, so haengt ihre Verschiebbarkeit vom Typ des Operandenfeldes ab. Enthaelt der Operand keine verschiebbaren Bestandteile, dann ist der Ausdruck nicht verschiebbar. Enthaelt der Operand nur einen verschiebbaren Bestandteil, dann ist der Ausdruck verschiebbar. Enthaelt der Operand zwei oder mehr verschiebbare Bestandteile, so ist der Ausdruck nicht verschiebbar.

Bytewerte sind nur verschiebbar, wenn das unaere groesser als '>'-Zeichen zur Kennzeichnung des hoeherwertigen Bytes und/oder das unaere kleiner als '<'-Zeichen zur Kennzeichnung des niederwertigen Bytes verwendet wird. Diese Operanden folgen denselben Verschiebungsregeln, wie vollstaendige 16-Bit-Adresswerte.

Die nachfolgenden Beispiele sollen die Erklaerungen deutlicher machen.

```
LABEL1:   NOP       ;Die Marke ist so definiert, dass sie
                     gleich der Adresse des Befehls ist.
                     Deshalb ist sie verschiebbar.

LABEL2:   .EQUAL    LABEL1
                    ;Gemaess Definition soll die Marke
                     gleich einem Wert sein, der als ver-
                     schiebbar gekennzeichnet wurde. Des-
                     halb ist LABEL2 ebenfalls verschieb-
                     bar.

LABEL3:   .EQUAL    10
                    ;Gemaess Definition ist die Marke
                     gleich einer Konstanten. Deshalb ist
                     LABEL3 nicht verschiebbar.

LABEL4:   .EQUAL    $+10
                    ;Gemaess Definition ist die Marke
                     gleich einem verschiebbaren Wert plus
                     einem nicht verschiebbaren Wert. Da
                     nur ein Wert verschiebbar ist, ist
                     auch das Symbol LABEL4 verschiebbar.

LABEL5:   .EQUAL    10+$
                    ;Gemaess Definition ist die Marke
                     gleich einem nicht verschiebbaren Wert
```

```
                    plus einem verschiebbaren Wert. Da nur
                    ein Wert verschiebbar ist, ist auch
                    LABEL5 verschiebbar.

LABEL6:   .EQUAL    LABEL5-LABEL2
                    ;Gemaess Definition ist die Marke
                     gleich einem verschiebbaren Wert minus
                     einem anderen verschiebbaren Wert. Das
                     fuehrt zu einem nicht verschiebbaren
                     Ergebnis.
```

An das letzte Beispiel sollte man sich erinnern, wenn man mit Hilfe des Assemblers die Groesse von Datenbereichen berechnen moechte. Hierzu das nachfolgende Beispiel. Es enthaelt eine Tabelle von Datenwerten. Die Anzahl der Bytes wird waehrend der Assemblierung automatisch durch den Assembler berechnet. Das gestattet dem Programmierer, die Tabelle zu erweitern oder sie zu verkuerzen, ohne sich um die Groesse des Datenblocks kuemmern zu muessen.

```
DATA:           .BYTE     0
                .WORD     10
                .BYTE     20
                .BLKB     5

DATA_SIZE:      .EQUAL    $-DATA
```

Der Assembler wird die Groesse des Datenblocks berechnen, und da das Ergebnis nicht verschiebbar ist, wird der Linker die Datenblockgroesse nicht veraendern.

2.5. Linker Beispiele

Dieser Abschnitt besteht aus Beispielen, an denen die Verwendung des Linkers demonstriert werden soll. Das Symbol <CR> kennzeichnet einen Wagenruecklauf und wird nur angegeben, wenn keine andere Antwort auf eine Eingabeanforderung gewuenscht wird. Andernfalls wird angenommen, dass jede Eingabe mit einem Wagenruecklauf abgeschlossen wird. In allen Faellen kann eine Datendatei eingerichtet werden, die genau die gleichen Antworten enthaelt, als wuerde man im "Prompt-Modus" arbeiten.

2.5.1. Einzelne Datei, ab der gewuenschten Startadresse assembliert

Im ersten Beispiel wird nur eine einzige Datei behandelt, die unter Verwendung einer ORG-Steueranweisung fehlerfrei ab der gewuenschten Adresse assembliert wurde. Es wird vorausgesetzt, dass die Ausgabedatei das voreingestellte Standardformat 'ausfuehrbar' haben soll und dass keine Linkeroptionen gewuenscht werden. Wenn keine zusaetzlichen Abschnitte definiert wurden und kein Umschalten zwischen den vordefinierten Abschnitten erfolgte, werden die Eingabeanforderungen des Linkers folgendermassen aussehen:

```
Input   Filename:  filename
        Enter Offset for 'CODE'            :0
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Die obigen Eingaben werden den Linker veranlassen, die Datei 'filename.obj' zu lesen, zu allen verschiebbaren Adressen eine 0 zu addieren, und eine Datei mit dem Namen 'filename.tsk' auszugeben.

Das folgende Beispiel unterscheidet sich von dem vorherigen nur darin, dass die Ausgabedatei im Format 'I-Hex' gewuenscht wird. Es ist zu beachten, dass das Standardausgabeformat des Linkers mit der Assemblersteueranweisung OPTIONS veraendert werden kann.

```
Input   Filename:  filename
        Enter Offset for 'CODE'            :0
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None): H
```

Das letzte Beispiel fuer diesen Typ von Dateien ist das gleiche wie das vorhergehende, wobei zusaetzlich eine Datei fuer die Ladetabelle auf dem Disk-Speicher angelegt werden soll und die Ausgabedatei einen vom Nutzer festgelegten Namen erhaelt. Die Optionen koennen in beliebiger Reihenfolge eingegeben werden.

```
Input   Filename:  filename
        Enter Offset for 'CODE'            :0
Input   Filename:  <CR>

Output  Filename:  user_filename

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None): HD
```

2.5.2. Einzelne Datei mit mehreren Abschnitten

Das Beispiel zeigt, wie der Linker mit mehreren Programmabschnitten verfaehrt. Wenn die vordefinierten Abschnitte CODE und DATA benutzt wurden und der DATA-Abschnitt unmittelbar an den CODE-Abschnitt anschliessen soll, sieht der Eingabedialog mit dem Linker folgendermassen aus:

```
Input   Filename:  filename
        Enter Offset for 'CODE'            : 0
        Enter Offset for 'DATA'            : <CR>
Input   Filename:  <CR>
```

```
Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Wenn anstelle der vordefinierten Abschnitte CODE und DATA die vom Nutzer definierten Abschnitte Program_section1 und Program_section2 benutzt werden und wenn Program_section2 unmittelbar an Program_section1 anschliessen soll, sieht der Eingabedialog folgendermassen aus:

```
Input   Filename:  filename
        Enter Offset for 'Program_section1': 0
        Enter Offset for 'Program_section2' : <CR>
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Eine wichtige Tatsache, der man sich immer bewusst sein sollte, ist, dass Abschnitte in der Reihenfolge angeordnet werden, in der sie definiert wurden. Deshalb gibt es in diesem Beispiel keine Moeglichkeit, Program_section1 hinter Program_section2 anzuordnen. Ist die Notwendigkeit gegeben, die Reihenfolge zu vertauschen, dann muss auch die Reihenfolge der SECTION-Steueranweisungen im Quellprogramm veraendert werden.

Wenn in dem obigen Beispiel Program_section1 auf 2000H und Program_section2 auf 4000H verschoben werden soll, muessen folgende Eingaben gemacht werden:

```
Input   Filename:  filename
        Enter Offset for 'Program_section1': 2000
        Enter Offset for 'Program_section2': 4000
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Man beachte, dass die Adressen immer hexadezimal angegeben werden. Wenn die Ausgabedatei vom Format 'ausfuehrbar' war, wird die Luecke zwischen dem Ende von Program_section1 und dem Anfang von Program_section2 mit dem standardmaessig festgelegten Fuellwert, FF Hex, aufgefuellt. Dieser Fuellwert kann mit der Assemblersteueranweisung FILLCHAR veraendert werden.

2.5.3. Mehrere Dateien mit mehreren Abschnitten

Dieses Beispiel illustriert, wie der Linker mit Abschnittsnamen verfaehrt, wenn mehrere Dateien vorhanden sind. Es wird vorausgesetzt, dass sowohl file1 als auch file2 den CODE- und den DATA-Abschnitt benutzen und dass die Linkstartadresse fuer die Datei 0 sein soll. Es ergibt sich folgender Eingabedialog:

```
  Input   Filename:  file1
          Enter Offset for 'CODE'            : 0
          Enter Offset for 'DATA'            : <CR>
  Input   Filename:  file2
          Enter Offset for 'CODE'            : <CR>
          Enter Offset for 'DATA'            : <CR>

  Output  Filename:  <CR>

  Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Damit wird eine Datei erzeugt, in der beide CODE-Abschnitte hintereinander liegen, gefolgt von beiden DATA-Abschnitten, die ebenfalls hintereinander angeordnet sind. Das zeigt die generelle Verfahrensweise bei der Anordnung von Abschnitten. Abschnitte werden entsprechend dem Namen angeordnet und in der Reihenfolge, in der sie im Quellprogramm definiert wurden. Alle CODE-Abschnitte werden zusammengefasst, dann alle DATA-Abschnitte usw. CODE-Abschnitte werden immer vor DATA-Abschnitten angeordnet, weil das die Reihenfolge ist, in der sie vordefiniert wurden. Sollte es notwendig sein, DATA vor CODE anzuordnen, dann darf CODE nicht benutzt werden, sondern stattdessen ein vom Nutzer definierter Abschnitt. Das gilt nur, wenn die Abschnitte unmittelbar hintereinander angeordnet werden sollen. Sollen alle CODE- und alle DATA-Abschnitte dicht gepackt werden, es aber fuer beide festgelegte Adressen gibt, so geht man folgendermassen vor, falls CODE bei E000H und DATA bei 1000H beginnen sollen:

```
  Input   Filename:  file1
          Enter Offset for 'CODE'            : E000
          Enter Offset for 'DATA'            : 1000
  Input   Filename:  file2
          Enter Offset for 'CODE'            : <CR>
          Enter Offset for 'DATA'            : <CR>

  Output  Filename:  <CR>

  Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Die in diesem Beispiel beschriebenen Regeln gelten unabhaengig davon, wieviel Eingabedateien vorhanden sind. Abschnitte koennen verwendet werden, um Programmteile entsprechend ihrer Funktion zusammenzufassen oder um einen komplexen Linkprozess zu unterstuetzen, da jeder beliebige Abschnitt auf einer eigene Adresse angeordnet werden kann.

2.5.4. Einzelne Datei mit einem Abschnitt fuer 'Reference only'

Ein 'Reference only'-Abschnitt ist ein Abschnitt, der ebenfalls gebunden wird, so dass alle in ihm definierten Globals fuer den Linkprozess genutzt werden koennen. Jedoch wird dieser Abschnitt nicht mit in die Ausgabedatei aufgenommen. 'Reference only'-Abschnitte sind in solchen Faellen sinnvoll, wo sich das Programm im ROM oder EPROM und der

Datenbereich im RAM befinden. Man moechte, dass die Ausgabedatei nur den Teil des Programms enthaelt, der im ROM gespeichert werden soll. Unter Verwendung eines Beispiels, aehnlich den vorhergehenden, wird angenommen, dass das Programm nur die vordefinierten Abschnitte CODE und DATA benutzt, dass CODE bei 1000H beginnen soll, und dass DATA unmittelbar an CODE anschliessen soll, nur fuer den Linkprozess benutzt und dann nicht weiter verwendet wird. Ein Minuszeichen vor einer Abschnittsadresse kennzeichnet den Abschnitt als 'Reference only'-Abschnitt. Ein Minuszeichen vor dem Namen eines Abschnitts kennzeichnet einen 'Reference only'-Abschnitt in der Ladetabelle.

```
Input   Filename:  filename
        Enter Offset for 'CODE'            : 1000
        Enter Offset for 'DATA'            : - <CR>
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Falls der DATA-Abschnitt auf 4000H gebracht werden soll, statt sich unmittelbar an den CODE-Abschnitt anzuschliessen, und falls er ebenfalls nur als 'Reference only'-Abschnitt verwendet werden soll, wird das folgendermassen erreicht:

```
Input   Filename:  filename
        Enter Offset for 'CODE'            : 1000
        Enter Offset for 'DATA'            : -4000
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

Jeder beliebige Abschnitt jeder Datei kann als 'Reference only'-Abschnitt verwendet werden, mit einer Ausnahme. Der erste Abschnitt der ersten Datei darf nicht als 'Reference only'-Abschnitt verwendet werden, da er die Basis fuer alle Berechnungen des Linkers bildet. Deshalb sollte man nicht den CODE-Abschnitt als 'Reference only'-Abschnitt verwenden. Man definiert stattdessen mit der Assemblersteueranweisung SECTION einen Abschnitt, den man zum Reference only'-Abschnitt macht. Benennt man diesen Abschnitt Ref_only, ergibt sich folgender Eingabedialog:

```
Input   Filename:  filename
        Enter Offset for 'DATA'            : 1000
        Enter Offset for 'Ref_only'        : -4000
Input   Filename:  <CR>

Output  Filename:  <CR>

Options (D, S, A, M, X, H, E, T, 1, 2, 3, <CR>=None):<CR>
```

2.6. Linker Ausgabeformate

2.6.1. Ausgabeformat der Tabelle der globalen Symbole

Dieser Abschnitt beschreibt das Format der Tabelle der globalen Symbole, die erzeugt wird, wenn die Linkeroption 'S' gewaehlt wird. Die Symboltabelle erhaelt immer den gleichen Namen wie die Ausgabedatei des Linkers mit der Namenserweiterung 'sym'.

Bytes	0 - 31	Name des globalen Symbols. Der Name ist mit Nullen aufgefuellt, um die 32 Zeichenpositionen zu belegen. Das Ende der Eintragungen kann durch FFH in Byte 0 gefunden werden.
Byte	32	Hoeherwertiges Byte der Adresse des globalen Symbols.
Byte	33	Niederwertiges Byte der Adresse des globalen Symbols.
Byte	34	Nummer der Datei, in der das globale Symbol definiert wurde. Wird vom Linker benutzt, um gemeinsam mit dem Wert den Dateinamen auszugeben. Das Byte kann, falls das gewuenscht wird, geloescht oder fuer andere Zwecke verwendet werden.
Byte	35	Flagbyte. Momentan wird dieses Byte noch nicht verwendet. Zukuenftige Verwendung ist moeglich.

2.6.2. Verkuerztes Ausgabeformat der Tabelle der globalen Symbole

Dieser Abschnitt beschreibt das Format der Tabelle der globalen Symbole, die erzeugt wird, wenn die Linkeroption 'A' gewaehlt wird. Die Symboltabelle erhaelt immer den gleichen Namen wie die Ausgabedatei des Linkers mit der Namenserweiterung 'sym'.

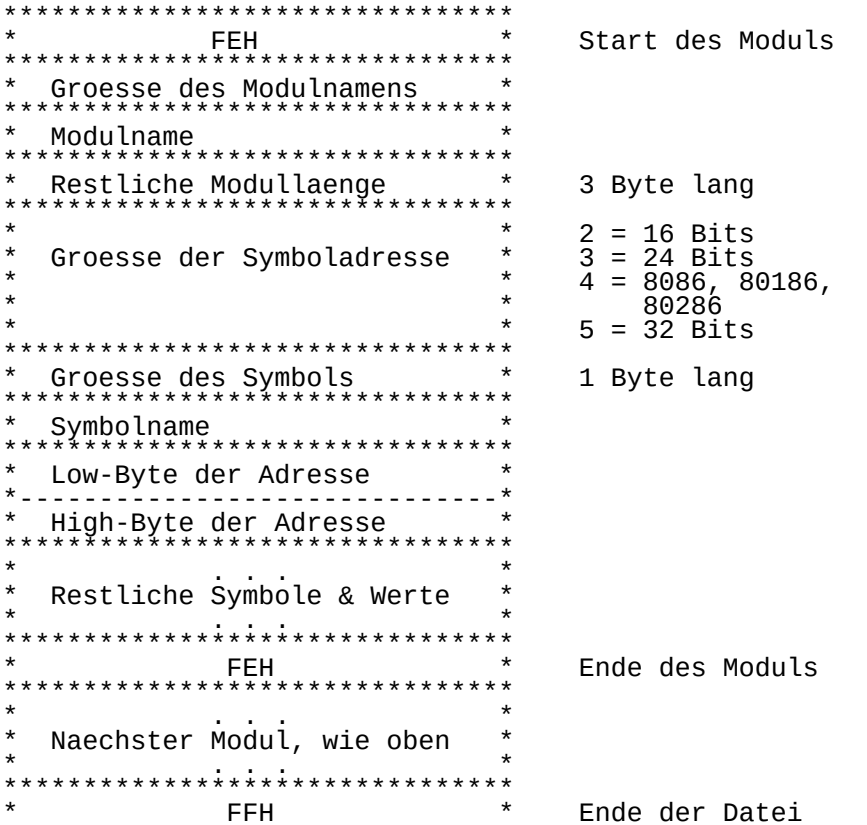
Bytes	0 - 9	Name des globalen Symbols. Der Name ist mit Nullen aufgefuellt, um die 10 Zeichenpositionen zu belegen. Das Ende der Eintragungen kann durch FFH in Byte 0 gefunden werden.
Byte	10	Hoeherwertiges Byte der Adresse des globalen Symbols.
Byte	11	Niederwertiges Byte der Adresse des globalen Symbols.
Byte	12	Nummer der Datei, in der das globale Symbol definiert wurde. Wird vom Linker

benutzt, um gemeinsam mit dem Wert den Dateinamen auszugeben. Das Byte kann, falls das gewuenscht wird, geloescht oder fuer andere Zwecke verwendet werden.

Byte 13 Flagbyte. Momentan wird dieses Byte noch nicht verwendet. Zukuenftige Verwendung ist moeglich.

2.6.3. 'Mic'-Ausgabeformat der Symboltabelle

Dieser Abschnitt beschreibt das 'Mic'-Format der Tabelle der globalen Symbole, das durch die Linkeroption 'M' ausgewaehlt wird. Die Symboltabelle erhaelt immer den gleichen Namen wie die Ausgabedatei des Linkers mit der Namenserweiterung 'sym'.



2.6.4. 'I-Hex' Format

Nachfolgend wird das 'I-Hex'-Format beschrieben:

Record-Markenfeld - Dieses Feld kennzeichnet den Anfang eines Records und besteht aus einem ASCII-Doppelpunkt (:).

Record-Laengenfeld - Dieses Feld besteht aus zwei ASCII-Zeichen, die die Anzahl der Bytes in diesem Record angeben. Die Zeichen sind das Ergebnis der Konvertierung der binaeren Anzahl der Datenbytes in zwei ASCII-Zeichen, wobei die hoeherwertige Zahl zuerst kommt. Ein 'End-of-file'-Record enthaelt zwei ASCII-Nullen in diesem Feld. Die maximale Anzahl von Datenbytes in einem Record betraegt 255. Mit der RECSIZE-Steueranweisung laesst sich dieser Wert veraendern.

Ladeadressenfeld - Dieses Feld besteht aus vier ASCII-Zeichen, die sich aus der Konvertierung des binaeren Wertes der Adresse, ab der dieses Record geladen werden soll, ergeben. Die Anordnung ist folgendermassen:

Hoeherwertige Ziffer des High-Bytes der Adresse.
Niederwertige Ziffer des High-Bytes der Adresse.
Hoeherwertige Ziffer des Low-Bytes der Adresse.
Niederwertige Ziffer des Low-Bytes der Adresse.

In einem 'End-of-file'-Record besteht dieses Feld entweder aus vier ASCII-Nullen oder der Eintrittsadresse fuer das Programm.

Record-Typfeld - Dieses Feld kennzeichnet den Recordtyp. Er ist entweder 00 fuer Datenrecords oder 01 fuer ein 'End-of-file'-Record. Er besteht aus zwei ASCII-Zeichen, mit der hoeherwertigen Ziffer des Recordtyps zuerst, gefolgt von der niederwertigen Ziffer des Recordtyps.

Datenfeld - Dieses Feld besteht aus den eigentlichen Daten, konvertiert in zwei ASCII-Zeichen, die hoeherwertige Ziffer zuerst. In dem 'End-of-file'-Record gibt es keine Datenbytes.

Pruefsummenfeld - Das Pruefsummenfeld ist eine binaere 8-Bit-Summe, gebildet aus dem Record-Laengenfeld, dem Ladeadressenfeld, dem Record-Typfeld und dem Datenfeld. Diese Summe wird dann negiert (2-er Komplement) und in zwei ASCII-Zeichen konvertiert, die hoeherwertige Ziffer zuerst.

2.6.5. 'Moto-S19'-Format

Nachfolgend wird das 'Moto S1-S9'-Format beschrieben:

Record-Typfeld - Dieses Feld kennzeichnet den Anfang eines Records und bezeichnet den Recordtyp folgendermassen:
ASCII S1 - Data Record
ASCII S9 - 'End-of-File'-Record

Record-Laengenfeld - Dieses Feld gibt die Recordlaenge an, die das Adressen-, das Daten- und das Pruefsummenfeld umfasst. Der 8-Bit-Recordlaengenwert wird in zwei ASCII-Zeichen konvertiert, wobei die hoeherwertige Zahl zuerst kommt. Da die kleinste Recordgroesse fuer eine Objektdatei 128 Byte betraegt, besteht das Record-Laengenfeld immer aus 128 Datenbytes, 2 Adressbytes und 1 Pruefsummenbyte, also einer Gesamtrecordlaenge von 131 Bytes. Dieser Wert kann mit der RECSIZE-Steueranweisung veraendert werden.

Ladeadressenfeld - Dieses Feld besteht aus vier ASCII-Zeichen, die sich aus der Konvertierung des binaeren Wertes der Adresse, ab der dieses Record geladen werden soll, ergeben. Die Anordnung ist folgendermassen:

Hoeherwertige Ziffer des High-Bytes der Adresse.
Niederwertige Ziffer des High-Bytes der Adresse.
Hoeherwertige Ziffer des Low-Bytes der Adresse.
Niederwertige Ziffer des Low-Bytes der Adresse.

In einem 'End-of-file'-Record besteht dieses Feld aus vier ASCII-Nullen.

Datenfeld - Dieses Feld besteht aus den eigentlichen Daten, konvertiert in zwei ASCII-Zeichen, die hoeherwertige

Ziffer zuerst. In einem 'End-of-file'-Record gibt es keine Datenbytes.

Pruefsummenfeld - Das Pruefsummenfeld ist eine binaere 8-Bit-Summe, gebildet aus dem Record-Laengenfeld, dem Ladeadressenfeld und dem Datenfeld. Diese Summe wird dann negiert (1-er Komplement) und in zwei ASCII-Zeichen konvertiert, die hoeherwertige Ziffer zuerst.

2.6.6. 'Moto-S28'-Format

Nachfolgend wird das 'Moto S2-S8'-Format beschrieben:

Record-Typfeld - Dieses Feld kennzeichnet den Anfang eines Records und bezeichnet den Recordtyp folgendermassen:
ASCII S2 - Data Record
ASCII S8 - 'End-of-File'-Record

Record-Laengenfeld - Dieses Feld gibt die Recordlaenge an, die das Adressen-, das Daten- und das Pruefsummenfeld umfasst. Der 8-Bit-Recordlaengenwert wird in zwei ASCII-Zeichen konvertiert, wobei die hoeherwertige Zahl zuerst kommt.

Ladeadressenfeld - Dieses Feld besteht aus sechs ASCII-Zeichen, die sich aus der Konvertierung des binaeren Wertes der Adresse, ab der dieses Record geladen werden soll, ergeben. Die Anordnung ist folgendermassen:

Hoeherwertige Ziffer des High-Bytes der Adresse.
Niederwertige Ziffer des High-Bytes der Adresse.
Hoeherwertige Ziffer des mittleren Bytes der Adresse.
Niederwertige Ziffer des mittleren Bytes der Adresse.
Hoeherwertige Ziffer des Low-Bytes der Adresse.
Niederwertige Ziffer des Low-Bytes der Adresse.

In einem 'End-of-file'-Record besteht dieses Feld aus sechs ASCII-Nullen.

Datenfeld - Dieses Feld besteht aus den eigentlichen Daten, konvertiert in zwei ASCII-Zeichen, die hoeherwertige

Ziffer zuerst. In einem 'End-of-file'-Record gibt es keine Datenbytes.

Pruefsummenfeld - Das Pruefsummenfeld ist eine binaere 8-Bit-Summe, gebildet aus dem Record-Laengenfeld, dem Ladeadressenfeld und dem Datenfeld. Diese Summe wird dann negiert (1-er Komplement) und in zwei ASCII-Zeichen konvertiert, die hoeherwertige Ziffer zuerst.

2.6.7. 'Moto-S37'-Format

Nachfolgend wird das 'Moto S3-S7'-Format beschrieben:

Record-Typfeld - Dieses Feld kennzeichnet den Anfang eines Records und bezeichnet den Recordtyp folgendermassen:
ASCII S3 - Data Record
ASCII S7 - 'End-of-file'-Record

Record-Laengenfeld - Dieses Feld gibt die Recordlaenge an, die das Adressen-, das Daten- und das Pruefsummenfeld umfasst. Der 8-Bit-Recordlaengenwert wird in zwei ASCII-Zeichen konvertiert, wobei die hoeherwertige Zahl zuerst kommt.

Ladeadressenfeld - Dieses Feld besteht aus acht ASCII-Zeichen, die sich aus der Konvertierung des binaeren Wertes der Adresse, ab der dieses Record geladen werden soll, ergeben. Die Anordnung ist folgendermassen:

Hoeherwertige Ziffer des High-Byte des High-Word der Adresse.
Niederwertige Ziffer des High-Byte des High-Word der Adresse.
Hoeherwertige Ziffer des Low-Byte des High-Word der Adresse.
Niederwertige Ziffer des Low-Byte des High-Word der Adresse.
Hoeherwertige Ziffer des High-Byte des Low-Word der Adresse.
Niederwertige Ziffer des High-Byte des Low-Word der Adresse.
Hoeherwertige Ziffer des Low-Byte des Low-Word der Adresse.
Niederwertige Ziffer des Low-Byte des Low-Word der Adresse.

In einem 'End-of-file'-Record besteht dieses Feld aus acht ASCII-Nullen.

Datenfeld - Dieses Feld besteht aus den eigentlichen Daten, konvertiert in zwei ASCII-Zeichen, die hoeherwertige Ziffer zuerst. In einem 'End-of-file'-Record gibt es keine Datenbytes.

Pruefsummenfeld - Das Pruefsummenfeld ist eine binaere 8-Bit-Summe, gebildet aus dem Record-Laengenfeld, dem Ladeadressenfeld und dem Datenfeld. Diese Summe wird dann negiert (1-er Komplement) und in zwei ASCII-Zeichen konvertiert, die hoeherwertige Ziffer zuerst.

Teil 5

W E G A

K1810WM86 Cross-Assembler

Nutzerhandbuch

Inhaltsverzeichnis Seite

1. Assembler

1.1. Einfuehrung

Dieser Abschnitt gibt einen Ueberblick ueber den K1810WM86 Cross-Assembler. Mit diesem Handbuch soll die Arbeitsweise des Cross-Assemblers beschrieben werden. Es wird vorausgesetzt, dass der Anwender mit der Arbeitsweise und dem Befehlsvorrat des K1810WM86 vertraut ist.

Der K1810WM86 Cross-Assembler wandelt Programme, die auf einem P8000/WEGA System geschrieben wurden, in Objektkode um. Dieser kann dann unter Verwendung des Linkers gebunden und auf die gewuenschte Abarbeitungsadresse gebracht werden.

Die Groesse der vom Assembler zu bearbeitenden Dateien wird nur durch die Speicherkapazitaet des Disk-Speichers bestimmt. Der Ueberlauf aller Puffer, einschliesslich des Puffers fuer die Symboltabelle, erfolgt auf den Disk-Speicher. Der Abschnitt ueber Makros behandelt Zeichenkettenvergleiche, Substitution von Formalparametern, Rekursion und Verschachtelung. Letztere werden nur durch die Groesse des verfuegbaren Disk-Speicherplatzes begrenzt.

Der Abschnitt ueber die bedingte Assemblierung befaehigt den Anwender, den Assembler so zu steuern, dass, in Abhaengigkeit vom Ergebnis waehrend der Abarbeitungszeit, unterschiedliche Abschitte der Quelldatei bearbeitet werden koennen.
Bedingungen koennen bis zu 248 Ebenen verschachtelt sein. Dabei unterstuetzt der Assembler den Programmierer beim Auffinden von Fehlern bei den bedingten Verschachtelungen, indem er nicht nur nachprueft, ob alle Verschachtelungsebenen abgeschlossen sind, sondern auch dadurch, dass er die momentane aktive Bedingungsebene im Objektkodefeld des Listings anzeigt.

Der Abschnitt ueber die waehrend der Assemblierung durchgefuehrte Berechnung von Ausdruecken zeigt, dass Berechnungen mit bis zu 16 Operanden moeglich sind. Dabei wird eine 32-Bit-Arithmetik verwendet. Die algebraische Hierarchie kann durch die Verwendung von Klammern veraendert werden.

Im Abschnitt ueber die Steuerung des Listendrucks werden Hinweise gegeben, wie das gesamte Programm oder nur Teile davon ausgegeben werden koennen. Das Programmlisting enthaelt brauchbare Hinweise zur Fehlerfindung. Es werden Kommandos beschrieben, mit denen waehrend der Assemblierung der Ausgabemodus veraendert werden kann.

Der Linker gestattet es, Dateien miteinander zu verbinden oder sie auch nur zur Herstellung von externen Adressverbindungen zu benutzen. Genau wie bei dem Assembler erfolgt der Ueberlauf fuer alle vom Linker verwendeten Puffer auf den Disk-Speicher. Das bedeutet, dass die Groessen von Globals, Externals und Objektkode nur durch

den verfuegbaren Disk-Speicherplatz begrenzt werden. In dem Bemuehen, unterschiedliche Ausgabeerfordernisse zu unterstuetzen, werden mehrere Versionen des Linkers zur Verfuegung gestellt. Eine vollstaendige Beschreibung wird in dem Abschnitt ueber den Linker gegeben.

1.2. K1810WM86 Cross-Assembler Bedienungsanweisungen

1.2.1. Prompt-Modus

Um den Assembler aufzurufen, wird eingegeben: asm86

Der Assembler antwortet:

```
LISTING DESTINATION ? (TI,DD,NL,DL,EO)
```

Dabei haben die Abkuerzungen folgende Bedeutung:

TI = Terminal Immediate (unmittelbar auf das Terminal)
DD = Disk Drive (auf den Disk-Speicher)
NL = No Listing (kein Listing)
DL = Direktive Listing (Listing gemaess Assemblersteueranweisungen)
EO = Error Only (nur Fehler)

Die Listingdatei erhaelt immer denselben Namen wie die eingegebene Datei, aber mit der Namenserweiterung "lst". Wird das Listing auf den Disk-Speicher gebracht, so wird zwischen dem Ende des Objektkodefeldes und dem Beginn des Markenfeldes ein zusaetzlicher Tabulator eingefuegt, so dass beim Ausdrucken der Datei ein ordentliches Druckbild entsteht.

Soll das Listing unter Steuerung des Assemblers entstehen (DL), wird jetzt die nachstehende Eingabeanforderung ausgegeben.

```
ASSEMBLER DIRECTIVE LISTING DESTINATION ? (TI,DD)
```

Die Listingausgabe unter Assemblersteuerung ermoeglicht es dem Nutzer, nur die Teile des Listings auszugeben, die durch die Steueranweisung LIST ON/OFF speziell gekennzeichnet wurden. Weitere Informationen zu LIST ON/OFF sind dem Abschnitt ueber die Steuerung der Listingausgabe zu entnehmen.

Wurde "Error Only" (EO) gewaehlt, fordert der Assembler den Nutzer folgendermassen zur Angabe des Bestimmungsortes auf.

```
ERROR ONLY LISTING DESTINATION ? (TI,DD)
```

Wurde "Disk Drive" (DD) gewaehlt, bzw. soll bei "Directive Listing" (DL) das Listing auf den Disk-Speicher gehen, fragt der Assembler, ob eine Cross-Referenz-Liste gewuenscht wird. Ist das nicht der Fall, wird der Assembler

nur eine alphabetische Symboltabelle anlegen.

```
GENERATE CROSS REFERENCE ? (Y/N, <CR> = NO)
```

Danach fordert der Assembler den Bediener zur Eingabe des Quellkodedateinamens auf.

```
INPUT FILENAME ? :
```

Schliesslich bittet der Assembler um den Namen der Ausgabedatei.

```
OUTPUT FILENAME ?:
```

Beantwortet der Bediener die Eingabeanforderung nur mit einem CR (Wagenruecklauf), so wird die Ausgabedatei denselben Dateinamen wie die Eingabedatei erhalten, aber mit der Namenserweiterung "obj".

1.2.2. Submit-Modus

Der Assembler kann auch unter Verwendung des "Submit-Modus" aufgerufen werden. In diesem Fall erstellt der Nutzer eine Datei, die die Antworten enthaelt, die er auf die Eingabeanforderungen im "Prompt-Modus" geben wuerde.

Ist zum Beispiel der Name der Eingabedatei "test.asm", der der Ausgabedatei "test.obj" und wird kein Listing gewuenscht, dann muss die zu erstellende Datei "file" folgendermassen aussehen:

```
nl
test.asm
test.obj
```

Der Aufruf des Assemblers erfolgt dann durch:

```
asm86 <file
```

Enthaelt die Datei bedingungsabhaengige Steueranweisungen zur Listingausgabe und sollen die gekennzeichneten Abschnitte auf das Terminal ausgegeben werden, dann muss die folgende Datei erstellt werden (vorausgesetzt dieselben Dateinamen wie oben sollen verwendet werden):

```
dl
ti
test.asm
test.obj
```

Wird jedoch gewuenscht, das Listing im obigen Beispiel auf den Disk-Speicher statt auf das Terminal auszugeben und wird keine Cross-Referenz-Liste gewuenscht, so muss man die folgende Datei anlegen:

```
dl
dd
n
test.asm
test.obj
```

Wird die ".ASK" Assemblersteueranweisung verwendet, so muss die entsprechende Eingabe in der richtigen Reihenfolge an das Ende der Datei gesetzt werden.

Die Kommandos zur Steuerung der Listingausgabe, die erst waehrend des Assemblierens eingegeben werden, lassen sich nur ueber die Konsole eingeben. Man kann sie nicht ueber eine Submitdatei bereitstellen. Jedoch ist ihre Eingabe auch bei der Bearbeitung einer Submitdatei moeglich.

1.3. Standardfestlegungen des Systems

Die nachfolgend angegebenen Dateinamenserweiterungen werden standardmaessig durch das System festgelegt, sofern der Nutzer keine anderen Festlegungen trifft.

asm - Eingabe fuer den Assembler
obj - Ausgabe vom Assembler
lst - Listingdatei

obj - Eingabe fuer den Linker

exe - Ausgabe von 'link1' - DCP Load File
cmd - Ausgabe von 'link2' - SCP86 Load File
com - Ausgabe von 'link3' - reine Abarbeitungsdatei
hex - Ausgabe von 'link5' - Ext. I-Hex-File

Es ist zu beachten, dass wegen der zusaetzlichen Informationen, die die Assembler-Ausgabedatei enthaelt, der Linker in jedem Fall benutzt werden muss. Das gilt auch, wenn das Programm schon auf der gewuenschten Startadresse assembliert wurde und keine externen Adressbezuege notwendig sind. Es ist einfach so, dass erst durch den Linker alle zusaetzlichen Informationen entfernt werden und eine Ladedatei erzeugt werden kann.

1.4. Assembler-Abarbeitungskommandos

Die folgenden Kommandos sind waehrend des Assemblierungsvorganges aktiv. Jedesmal bevor die naechste Programmzeile bearbeitet wird, wird ueberprueft, ob ueber die Konsole eine Eingabe erfolgte. Diese Kommandos sind sowohl waehrend des ersten als auch waehrend des zweiten Durchlaufs aktiv. Sie ueberschreiben den beim Aufruf des Assemblers festgelegten Ausgabemodus.

CTRL S = Unterbrechung des Assemblers
CTRL Q = Fortsetzung des Assemblers
DEL C = Beenden der Assemblierung
DEL T = Ausgabe des Listings auf dem Bildschirm
DEL D = Ausgabe des Listings auf den Disk-Speicher
DEL M = Ausgabe des Listings sowohl auf Bildschirm als auch auf Disk-Speicher
DEL N = Abbruch der Listingausgabe

Wird die DEL D-Funktion benutzt und es wurde noch keine Datei auf dem Disk-Speicher angelegt, so legt der Assembler zunaechst diese Datei an, bevor er mit der Assemblierung fortfaehrt.

1.5. Assembler-Fehlerbehandlung

Tritt ein Assemblierungsfehler auf, so sind die Massnahmen, die der Assembler trifft, abhaengig von dem Ausgabemodus, unter dem er gerade arbeitet.

Wurde die NL-Option (kein Listing) angegeben, so wird die Anweisung, die den Fehler und die Fehlernachricht verursacht hat, auf dem Bildschirm angezeigt. Die Bildschirmanzeige wird eingeschaltet und der Assembler haelt an, als haette der Nutzer ^S eingegeben. Der Grund dafuer ist, dass dem Nutzer die Moeglichkeit gegeben werden soll, genau zu sehen, wo sich der Fehler befindet. Das geschieht sowohl beim ersten als auch beim zweiten Durchlauf. Man beachte, dass manche Fehler, wie z.B. undefinierte Symbole beim ersten Durchlauf nicht gefunden werden koennen. Nachdem der Fehler angezeigt wurde, kann die Bildschirmanzeige mit DEL N wieder unterdrueckt werden.

Ist die Ausgabe des Listings auf dem Drucker vorgesehen, so werden im ersten Durchlauf festgestellte Fehler auf dem Bildschirm angezeigt aber nicht auf dem Drucker ausgegeben. Der Assembler haelt auch nicht an. Beim zweiten Durchlauf wird der Fehler sowohl auf dem Drucker als auch auf dem Bildschirm ausgegeben und der Assembler arbeitet weiter.

Ist die Ausgabe des Listings auf dem Drucker gemaess Steueranweisungen des Assemblers vorgesehen, so werden Fehler, die im ersten Durchlauf festgestellt wurden, auf dem Bildschirm ausgegeben, nicht jedoch auf dem Drucker. Der Assembler arbeitet weiter. Fehler, die im zweiten Durchlauf festgestellt werden, werden sowohl auf dem Drucker als auch auf dem Bildschirm ausgegeben, selbst wenn sie sich in einem Block befinden, der nicht fuer die Ausgabe vorgesehen war.

Eine Symboltabelle wird ausgegeben, wenn fuer die Ausgabe des Listings entweder der Drucker oder der Disk-Speicher festgelegt wurden. Die Symboltabelle wird nicht auf dem Bildschirm ausgegeben, es sei denn die Ausgabe sollte sowohl ueber den Bildschirm als auch ueber den Drucker bzw. sowohl ueber den Bildschirm als auch auf den Disk-Speicher erfolgen. Erfolgt die Ausgabe entsprechend den Assemblersteueranweisungen, so wird die Symboltabelle nur dann ausgegeben, wenn sich das Programmende in einem LIST ON-Block befindet. Andernfalls wird die Symboltabelle nicht ausgegeben.

1.6. Beschreibung der fuer die Assemblierung benoetigten Dateien

Waehrend des Assemblierungsvorganges arbeitet der Assembler mit den folgenden Dateien:

Quellendatei:
Die Datei, die den Kode des Quellenprogramms enthaelt.

Objektdatei:
Die Ausgabedatei des Assemblers, die den Objektkode, Informationen ueber Adressverschiebungen, externe und globale Symbole mit den dazugehoerigen Adressinformationen und die Information ueber den Programmeintrittspunkt enthaelt.

Datei fuer Adressverschiebungen:
Dies ist eine temporaere Datei, die dazu benutzt wird, Informationen ueber Adressverschiebungen waehrend des Assemblierungsprozesses abzulegen.

Symboltabellendatei:
Diese Datei findet Verwendung, wenn die Symboltabelle vollgeschrieben ist. Das ist bei etwa 1000 Symbolen der Fall.

Makrotabellendatei:
Diese Datei wird fuer die Speicherung der Makronamen benutzt, wenn die Makrotabelle vollgeschrieben ist. Das geschieht, nachdem 204 Makros definiert wurden.

Makropufferdatei:
Diese Datei wird zur Speicherung der Makrodefinitionen und der Formalparameterlisten verwendet. Die Parameterliste kann bis zu 80 Zeichen enthalten. Die Datei enthaelt weiterhin Informationen ueber Makroverschachtelungen, die nur durch die Groesse des Disk-Speichers begrenzt sind.

Datei fuer externe Symbole:
Diese Datei enthaelt Informationen ueber die Adressen der externen Symbole, die im Programm verwendet wurden.

Mit Ausnahme der Quellendatei werden alle Dateien auf demselben Laufwerk wie die Objektdatei gespeichert. Da der Ueberlauf eines jeden Puffers, den der Assembler benutzt, auf den Disk-Speicher erfolgt, liegt die einzige Begrenzung fuer die Groesse eines zu assemblierenden Programms in dem verfuegbaren Disk-Speicherplatz.

Am Ende des ersten Durchlaufs durchforscht der Assembler die Symboltabelle nach mehrfach definierten Symbolen. Bei Programmen mit weniger als etwa 600 Symbolen ist die dafuer benoetigte Zeit bedeutungslos. Bei Programmen von etwa dieser Groesse oder darueber hinaus kann der Nutzer fuer einige Sekunden den Eindruck gewinnen, als haette der Assembler aufgehoert zu arbeiten. Dieser Hinweis wird gege-

ben, um zu verhindern, dass sich der Nutzer in solch einem Fall Sorgen macht.

1.7. Syntaxregeln der K1810WM86 Assemblersprache

Dieser Abschnitt beschreibt die vom K1810WM86 Cross-Assembler verwendete Syntax. Im Abschnitt 1.8. werden Beispiele fuer die verschiedenen Adressierungsarten gegeben.

1.7.1. Bezeichnung der Zahlenbasis

Zahlenbasen werden folgendermassen bezeichnet:

Binary - B
Octal - Q
Decimal - D oder ohne Bezeichnung der Basis
HEX - H
ASCII - "X" oder 'X'

Die nachfolgend angegebenen 2-Zeichenfolgen, die in einfache oder doppelte Hochkommas eingeschlossen sind, sind folgendermassen definiert:

"CR" oder 'CR' - Carriage return (Wagenruecklauf)
"LF" oder 'LF' - Line feed (Zeilenschaltung)
"SP" oder 'SP' - Space (Leerzeichen)
"HT" oder 'HT' - Horizontal tab (Tabulator)
"NL" oder 'NL' - Null

1.7.2. Segment-Darstellung

Segmentnummern werden durch die Segmentnummer, der ein Doppelpunkt folgt, angegeben. Die Zahlenbasis der Segmentnummer soll dieselbe sein, wie die des Offsets (Versatz).

1.7.3. Marken

Marken koennen bis zu 10 Zeichen lang sein und in jeder Spalte beginnen, wenn der Name durch einen Doppelpunkt abgeschlossen wird. Wird kein Doppelpunkt verwendet, muss die Marke in Spalte 1 beginnen. Alle Marken muessen mit einem Buchstaben beginnen. Es wird zwischen Gross- und Kleinbuchstaben unterschieden.

1.7.4. Programmkommentare

Kommentarzeilen muessen mit einem Semikolon in Spalte 1 beginnen, falls nicht die .COMMENT-Steueranweisung verwendet wird. Kommentare nach einem Befehl benoetigen kein Semikolon, sofern sie wenigstens durch ein Leerzeichen oder einen Tabulator vom Befehl getrennt sind.

1.7.5. Befehlszaehler ($)

Das Sonderzeichen '$' kann in einem Ausdruck verwendet werden, um den Befehlszaehler zu bezeichnen. Der Wert, der dem $-Zeichen zugeordnet wird, ist der Wert des Befehlszaehlers bei Beginn des Befehls. Wenn der Befehl einen Wortzugriff ausfuehrt, wird auch der Stand des Befehlszaehlers entsprechend veraendert.

1.7.6. Gross-/Kleinbuchstaben

Fuer alle Mnemonik- und Registerbezeichnungen koennen Gross- oder Kleinbuchstaben verwendet werden. Bei Marken wird jedoch zwischen Gross- und Kleinbuchstaben unterschieden. Zu dem vorliegenden Programmpaket gehoert auch ein Programm, 'cscc' genannt, mit dem Quellenkode von Klein- in Grossbuchstaben konvertiert werden kann. Das Programm laesst ASCII-Zeichenketten, die in Apostrophe eingeschlossen sind, unveraendert. Aber es verwandelt alle Zeichen einer Marke in Grossbuchstaben. Weitere Hinweise sind der Beschreibung dieses Dienstprogramms (s.Abschn. 4.2.) zu entnehmen.

1.8. Adressierungsarten

Die verschiedenen Adressierungsarten, die der K1810WM86 gestattet, werden nachfolgend anhand von Beispielen gezeigt.

1.8.1. Unmittelbare (immediate) Adressierung

Der Datenwert ist im Befehl enthalten.

Beispiele:

Die Datengroesse ist aus dem Operandenfeld (Register als Ziel) ersichtlich und der Wert ist nicht verschiebbar. Da vorausgesetzt wird, dass es sich um einen unmittelbaren Datenwert handelt, steht hier nicht die Frage, ob die Adresse eines Speicherplatzes oder dessen Inhalt gewuenscht wird.

```
MOV     AL,10
MOV     AX,10
```

Die Datengroesse kann aus dem Operandenfeld nicht bestimmt werden. Deshalb ist ein Groessenbezeichner erforderlich. Der Datenwert ist nicht verschiebbar. Da vorausgesetzt wird, dass es sich um einen unmittelbaren Datenwert handelt, steht hier nicht die Frage, ob die Adresse eines Speicherplatzes oder dessen Inhalt gewuenscht wird.

```
MOV     BYTE PTR [BX],10
MOV     WORD PTR [BX+DI],10
```

Die Datengroesse kann aus dem Operandenfeld bestimmt werden und der Datenwert ist verschiebbar. Da der Datenwert verschiebbar ist, muss es sich um eine Adresse handeln. Deshalb kann der Assembler nicht erkennen, ob der Adresswert selbst oder der Inhalt dieser Adresse gewuenscht wird. Der OFFSET-Operator wird verwendet, um die Adresse eines Speicherplatzes statt seines Inhalts zu bezeichnen.

```
ADDRESS:  .WORD   10
          MOV     AX,OFFSET ADDRESS
```

1.8.2. Register Adressierung

Der Datenwert ist in einem CPU-Register enthalten und deshalb gibt es bezueglich der Operandengroesse keine Fragen.

Beispiele:

```
          MOV     AL,AH
          MOV     AX,BX
```

1.8.3. Indirekte Register Adressierung

Ein Register weist auf die Operandenadresse.

Beispiele:

Das Quellenregister liefert die Information, die benoetigt wird, um die Operandengroesse zu bestimmen.

```
          MOV     [BX],AL
          MOV     [BX],AX
```

Der Quellenoperand ist ein unmittelbarer Datenwert. Deshalb ist ein Bezeichner fuer die Operandengroesse erforderlich.

```
          MOV     BYTE PTR [BX],10
          MOV     WORD PTR [BX],10
```

1.8.4. Direkte Adressierung

Die Adresse des Operanden ist im Befehl enthalten. Gewuenscht wird der Inhalt des Speicherplatzes.

Beispiele:

Der Assembler beruecksichtigt die Groessen der vereinbarten Speicherplaetze. Deshalb ist kein Bezeichner fuer die Operandengroesse erforderlich.

```
DATABYTE: .BYTE
DATAWORD: .WORD
```

```
MOV     AL,DATABYTE
MOV     AX,DATAWORD
MOV     DATABYTE,AL
MOV     DATAWORD,AX
```

1.8.5. Index Adressierung

Die Operandenadresse ist die Summe aus zwei gueltigen Indexregistern oder aus zwei gueltigen Indexregistern plus einem Offset (Versatz). Verwendet man nur ein Indexregister, so wird es in Klammern eingeschlossen. Verwendet man zwei Indexregister, so kann man jedes von ihnen in Klammern einschliessen oder man klammert nur einmal und setzt ein '+'-Zeichen zwischen beide. In jedem Fall ist die Reihenfolge der Anordnung der Register und des Versatzes beliebig.

Beispiele:

Einer der Operanden ist ein CPU-Register. Deshalb ist kein Bezeichner fuer die Operandengroesse erforderlich.

```
MOV     AL,[BX][DI]
MOV     AX,[BX][DI]
MOV     AL,[BX+DI]
MOV     AX,[BX+DI]
MOV     AL,10[BX][DI]
MOV     AX,10[BX][DI]
MOV     [BX][DI],AL
MOV     [BX][DI],AX
MOV     [BX+DI],AL
MOV     [BX+DI],AX
MOV     10[BX][DI],AL
MOV     10[BX][DI],AX
```

Der Quellenoperand ist ein unmittelbarer Datenwert. Deshalb kann der Assembler nicht feststellen, welche Groesse der Operand hat. Es muss ein Groessenbezeichner verwendet werden.

```
MOV     BYTE PTR [BX][DI],10
MOV     WORD PTR [BX+DI],10
MOV     BYTE PTR 10[BX+DI],10
```

1.8.6. Relative Adressierung

Die Operandenadresse ist relativ zum aktuellen Befehl. Wird die Adresse als numerischer Wert gegeben, so wird am Anfang des naechsten Befehles begonnen zu zaehlen.

Beispiele:

```
JMP     4           ;Springe 4 Bytes vorwaerts
JMP     ADDRESS     ;Der Assembler berechnet die
                     entsprechende Verschiebung
```

1.9. Intra-Segment, Inter-Segment-Bezeichner

In diesem Abschnitt wird beschrieben, in welcher Weise Programmverzweigungen angegeben werden. Es gibt zwei Arten von CALL- und JMP-Befehlen. Die erste Art ist ein CALL oder JMP innerhalb desselben Segments (Intra-Segment). Fuer diese Art der Programmverzweigung ist kein Bezeichner erforderlich. Solange kein Bezeichner angegeben ist, setzt der Assembler immer voraus, dass es sich um eine Programmverzweigung innerhalb des Segments handelt. Diese Intra-Segment-Programmverzweigungen werden in der folgenden Weise notiert:

```
        JMP         ADDRESS     ;ADDRESS muss im selben
                                 Segment sein
        CALL        ADDRESS     ;ADDRESS muss im selben
                                 Segment sein
```

Fuer Programmverzweigungen zu einem anderen Segment (Inter-Segment) muss der FAR-Bezeichner verwendet werden. Inter-Segment-Programmverzweigungen werden folgendermassen notiert:

```
        JMP     FAR ADDRESS     ;ADDRESS in einem ande-
                                 ren Segment
        CALL    FAR ADDRESS     ;ADDRESS in einem ande-
                                 ren Segment
```

Ein Inter-Segment-CALL muss einen entsprechenden Inter-Segment-Return nach sich ziehen. Ebenso wie bei dem CALL-Befehl geschieht das unter Verwendung des FAR-Bezeichners.

```
        RET    FAR      ;Return fuer den oben gezeigten
                         Inter-Segment-CALL
```

Sowohl bei einem Intra- als auch bei einem Inter-Segment-Return kann eine 16-Bit-Konstante zum Stackpointer addiert werden. Das wird dann folgendermassen angegeben:

```
        RET     4           ;Inra-Segment-RET mit 4
                            Bytes, die zum SP addiert
                            werden
        RET     FAR     4   ;Inter-Segment-RET mit 4
                            Bytes, die zum SP addiert
                            werden
```

Intra-Segment-Spruenge koennen auch als relative Spruenge angegeben werden, sofern sich der Zielspeicherplatz innerhalb +128 oder -127 Bytes vom naechsten Befehl befindet.Dafuer muss der SHORT-Bezeichner verwendet werden.

```
        JMP     SHORT   ADDRESS ;ADDRESS muss innerhalb
                                 +128 oder -127 Bytes vom
                                 naechsten Befehl liegen
```

1.10. Assembler-Steueranweisungen

Dieser Abschnitt beschreibt die Assembler-Steueranweisungen. Es ist moeglich, den Steueranweisungen einen Dezimalpunkt voranzustellen. Das erleichtert es, sie von Programmbefehlen zu unterscheiden.

1.10.1. Steueranweisungen fuer den Speicher

```
        ORG  EXP
```

Legt die absolute Adresse fuer die Assemblierung fest. Die Anweisung kann verwendet werden, um entweder die Startadresse eines CODE- oder eines DATA-Abschnittes zu bestimmen, abhaengig davon, was zuvor festgelegt wurde. Wird diese Steueranweisung nicht gegeben, wird die Assemblierungsadresse auf 0000:0000 voreingestellt. Der festgelegte Segmentwert gestattet es auch festzustellen, ob es moeglich ist, mit den in den Segmentregistern angenommenen Werten auf den Operanden zuzugreifen. Eine weitere Moeglichkeit, den Assembler darueber zu informieren, was fuer Werte in den Segmentregistern stehen, wird bei der Beschreibung der ASSUME-Anweisung gegeben.

```
LABEL:      DB      EXP
            DEFB
            BYTE
```

Der Assembler legt den Wert des Ausdrucks in aufeinanderfolgenden Speicherplaetzen ab. Der BYTE-Ausdruck kann eine beliebige Mischung von Operandentypen sein, die voneinander durch ein Komma getrennt sind. ASCII-Zeichenketten muessen in Apostrophe eingeschlossen sein. Wenn die Zeichenkette ein Apostroph enthaelt, wird das durch zwei nebeneinander stehende Apostrophe angegeben. Wird kein Ausdruck angegeben , so wird ein Byte reserviert und zu Null gesetzt. Eine Marke ist optional, muss aber, falls sie verwendet wird, auf derselben Zeile stehen wie die BYTE-Steueranweisung. Nachfolgend werden ein paar Beispiele fuer die Verwendung der BYTE-Steueranweisung gegeben.

```
    BYTE                ;Reserviert 1 zu Null gesetztes
                         Byte
    BYTE    10          ;Reserviert 1 Byte=10 (dezimal)
    BYTE    1,2,3       ;Reserviert 3 Bytes = zu 1,2  &
                         3 gesetzt,  in dieser Reihen-
                         folge
    BYTE    SYMBOL-10   ;Sucht in der Symboltabelle
                         nach SYMBOL, subtrahiert 10
                         (dezim.) von seinem Wert und
                         speichert das Ergebnis
    BYTE    'HELLO'     ;Speichert die der Zeichenkette
                         Hello entsprechenden Ascii-
                         Zeichen in aufeinanderfolgen-
                         den Speicherplaetzen
    BYTE    'Hello', 0DH ;Wie im obigen Beispiel, mit
                         zusaetzlichem CR am Ende.
```

```
                                Leerzeichen vor den Operanden
                                bleiben unberuecksichtigt,
                                aber das Komma ist erforder-
                                lich.
     BYTE       'Hello''s'      ;Eingeschlossene Apostrophe.
```

```
LABEL:          DW          EXP
                DEFW
                WORD
```

Arbeitet genau wie BYTE, nur mit dem Unterschied, dass fuer jeden Operanden ein volles Wort verwendet wird. Von dem Wert wird das niederwertigste Byte zuerst gespeichert. Die Marke ist optional, wird sie jedoch verwendet, muss sie auf derselben Zeile wie die WORD-Steueranweisung stehen.

```
LABEL:          DD          EXP
                LONG
```

Speichert den Wert des Ausdrucks als eine vollstaendige segmentierte Adresse. Der Offset (Versatz) wird zuerst gespeichert, mit dem niederwertigsten Byte zuerst. Dann wird das Segment gespeichert, wiederum das niederwertigste Byte zuerst. Die Marke ist optional, wird sie jedoch verwendet, muss sie auf derselben Zeile wie die LONG-Steueranweisung stehen.

```
LABEL:          ASCII       STRING
```

Speichert die Zeichenkette STRING, die entweder bis zu einem CR (Wagenruecklauf) oder einem "|" (HEX 7C) reicht, es aber nicht enthaelt. Die Marke ist optional, wird sie jedoch verwendet, muss sie auf derselben Zeile wie die ASCII-Steueranweisung stehen. Nachfolgend werden ein paar Beispiele fuer die ASCII-Steueranweisung gegeben.

```
     ASCII      Hello       ;Speichert die ASCII-Darstellung
                             von Hello in aufeinanderfolgenden
                             Speicherplaetzen. Uebrigens wuer-
                             de auch dieser Kommentar mit
                             gespeichert werden.
     ASCII      Hello|      ;Nun wuerde der Kommentar nicht
                             mit gespeichert werden. Im naech-
                             sten Beispiel wurde die Zeichen-
                             kette nur mit einem CR abge-
                             schlossen.
     ASCII      Hello
```

```
LABEL:          BLKB        EXP
```

Reserviert die Anzahl von Bytes, die durch den Ausdruck 'EXP' festgelegt wird. Die reservierten Bytes werden zu Null gesetzt. Die Marke ist optional, wird sie jedoch verwendet, muss sie auf derselben Zeile wie die BLKB-Steueranweisung stehen.

```
LABEL:          BLKW      EXP
```

Reserviert die Anzahl von Worten, die durch den Ausdruck 'EXP' festgelegt werden. Die Marke ist optional, wird sie jedoch verwendet, muss sie auf derselben Zeile wie die BLKW-Steueranweisung stehen.

```
LABEL:          RESERVE   EXP
```

Diese Steueranweisung kann verwendet werden, um dem Lader fuer DCP oder SCP86 mitzuteilen, wieviel Bytes ueber das Ende des Programms hinaus reserviert werden sollen. Die Marke ist optional und falls sie verwendet wird, erhaelt sie keine BYTE-oder WORD-Kennzeichnung. Der Linker wird die Werte der in jedem Modul reservierten Groessen addieren und die Laenge auf FFFFH begrenzen, falls dieser Wert ueberschritten wird. DCP beginnt das Laden der Programme bei niedriger Speicheradresse, wogegen SCP86 das Laden von Programmen bei hoher Speicheradresse beginnt. Deshalb ist es noetig, Speicherplatz zu reservieren. Das geschieht, indem entweder eine der auf den vorhergehenden Seiten beschriebenen Daten- oder Block-Steueranweisungen fuer den Speicher oder die RESERVE-Steueranweisung verwendet wird. Der Unterschied ist, dass RESERVE nicht die Datei vergroessert und auch nicht den Speicherplatzzaehler des Assemblers erhoeht. Unten folgt ein Beispiel fuer RESERVE.

Dieses Beispiel zeigt zwei Moeglichkeiten fuer die Zuweisung von Pufferspeicherplatz. Im ersten Fall wird die Zuweisung tatsaechlicher Bestandteil der Objektkodedatei, im zweiten Fall ist das nicht so.

```
BUFFER1:  .BLKB     80          ;80 Bytes fuer Buffer1
BUFFER2:  .BLKB     80          ;80 Bytes fuer Buffer2

BUFFER1:  .EQUAL    $           ;Adresse von BUFFER1
BUFFER2:  .EQUAL    $+80        ;Adresse von BUFFER2
BUFF_END: .EQUAL    BUFFER2+80  ;Adresse des Endes von
                                 Buffer2

          .RESERVE  BUFF_END-BUFFER1+1
```

Man beachte, dass, da der Speicherplatz nicht tatsaechlich in der Ladedatei reserviert ist, eine Verschiebung nicht funktionieren wird, wenn RESERVE in der Mitte einer Datei verwendet wurde. Dasselbe gilt fuer Mehrfachdateien, wenn der Versuch gemacht wird, Speicherplatz zwischen zwei Daten erzeugenden Steueranweisungen zu reservieren. Die Ergebnisse werden nicht wie erwartet sein, da der Befehlszaehler des Assemblers nicht erhoeht wird.

```
          END
```

Definiert das Ende des Programms. Jedes Programm muss eine END-Anweisung haben. Ebenso muss jedes Programm, das durch eine INCLUDE-Anweisung in die Assemblierung einbezogen wird, eine END-Anweisung haben.

1.10.2. Steueranweisungen fuer Definitionen

```
LABEL:          EQU       EXP
                EQUAL
```

Setzt 'LABEL' dem Wert von 'EXP' gleich.

```
LABEL:          VAR       EXP
                DEFL
```

Setzt 'LABEL' auf den Wert von 'EXP'. Diese Zuordnung kann jedoch innerhalb des gesamten Programms immer wieder veraendert werden. Eine Marke, die als Variable definiert wurde, sollte nicht durch eine EQUAL-Steueranweisung neu definiert werden.

```
LABEL:          MACRO     ARGS
```

Kennzeichnet den Beginn einer Makrodefinition.

```
                ENDM
                MACEND
```

Kennzeichnet das Ende einer Makrodefinition.

```
                MACEXIT
```

Diese Steueranweisung veranlasst den unmittelbaren Austritt aus einem Makro. Der Unterschied zwischen MACEXIT und MACEND ist der, dass waehrend des Vorgangs der Makrodefinition das Makro durch MACEXIT nicht beendet wird und dass, wenn sich MACEXIT im Zweig der nicht erfuellten Bedingung eines bedingten Assemblerblocks befindet, es nicht ausgefuehrt wird.

```
                EXTERNAL  LABEL
```

Macht kenntlich, dass die Marke in einem anderen Programm definiert wurde. Es koennen mehrere Marken angegeben werden. Sie muessen aber durch Komma voneinander getrennt sein. Typbezeichner koennen verwendet werden. In manchen Faellen sind sie erforderlich, um dem Assembler mitzuteilen, welche Operandengroesse die Marke repraesentiert. Nachfolgend illustrieren ein paar Beispiele die Verwendung von Groessenbezeichnern.

```
    EXTERNAL  BYTE     DATABYTE   ;8 Bit Speicherplatz
    EXTERNAL  WORD     DATAWORD   ;16 Bit Speicherplatz
    EXTERNAL  LONG     DATALONG   ;32 Bit Speicherplatz
    EXTERNAL  SEGMENT  ADDRESS    ;Segmentadresse
    EXTERNAL  ADDRESS             ;Befehlsadresse
```

```
                GLOBAL    LABEL
                PUBLIC
```

Kennzeichnet die Marke als eine globale Marke, auf die von anderen Programmen zugegriffen werden kann. Es koennen mehrere Marken angegeben werden, die aber durch Komma getrennt sein muessen. Es werden einige Beispiele fuer den korrekten Gebrauch von GLOBAL angegeben.

```
GLOBAL    SYM1          ;Deklariert die Marke SYM1 fuer
                          andere Programme als verfueg-
                          bar. Der Linker wird die exter-
                          nen Adressbezuege aufloesen.
GLOBAL    SYM1, SYM2    ;Mehrfache Deklarationen auf ei-
                          ner Zeile sind zulaessig, wenn
                          sie durch Komma getrennt sind.
                          Leerzeichen werden ignoriert.
```

```
ASSUME    EXP1, EXP2 ...
```

Diese Steueranweisung kann dazu verwendet werden, den Assembler ueber den Inhalt der Segmentregister zu informieren. Das erlaubt dem Assembler, die Adressierbarkeit von Operanden zu ueberpruefen und sich davon zu ueberzeugen, dass der Operand sich wirklich in dem Segment befindet, in dem seine Adressierung erfolgte. Es folgen ein paar Beispiele fuer die Verwendung von ASSUME.

```
ASSUME    CS = 0, DS = 0, ES = 0, SS = 0
ASSUME    CS = 0000H
ASSUME    DS = 1000H
ASSUME    CS = 0000H, DS = 1000H, ES = 2000H, SS = 3000H
```

Das erste Beispiel ist identisch mit den vom Assembler voreingestellten Werten (Standardmodus). Diese Einstellung kann fuer Programme verwendet werden, die insgesamt, sowohl fuer Kode als auch fuer Daten, nur 64 K Speicher benoetigen. Dieser Modus ist auch nuetzlich fuer K580IK80- oder U880-Programme, die umgesetzt wurden, da der K580IK80 und der U880 keine Segmentregister haben. Das zweite und dritte Beispiel beschreiben eine Situation, wo der Kode- und der Datenbereich jeder seinen eigenen 64 K Byte Speicherbereich hat. Das vierte Beispiel zeigt den Fall, wo alle vier Segmente ihren eigenen, sich nicht ueberlappenden Speicherbereich haben. Man beachte, dass die ASSUME-Steueranweisung nur zur Ueberpruefung der Adressierbarkeit dient. Das Programm muss den entsprechenden Kode enthalten, um die Segmentregister mit den gewuenschten Werten zu laden. Darueber hinaus muessen dem Assembler die verschiedenen Abschnitte durch die Verwendung der CODE, DATA, EXTRA und STACK Steueranweisungen fuer Programmabschnitte mitgeteilt werden. Diese Steueranweisungen sagen dem Assembler, was fuer einen Befehlszaehler er verwenden soll. Wenn jeder Abschnitt bei 0000:0000 beginnt, kann der Linker dazu benutzt werden, jeden Abschnitt in das endgueltige Segment zu verschieben. Hierzu alternativ kann jeder Abschnitt eine ORG Steueranweisung mit einer Segmentangabe enthalten. Dann erhaelt man ein Listing mit korrekten Adressbezuegen.

LABEL: ASK PROMPT

Gibt PROMPT auf das Teminal aus und wartet auf eine 1 Zeichen Eingabe, von der 30H subtrahiert wird. Gewoehnlich ist das Ziel dieser Steueranweisung, ein 0/1 Flag in das Programm einzufuehren. LABEL wird dem Ergebnis gleichgesetzt. Ein CR beendet PROMPT. Beim zweiten Durchlauf wird die Zeile gemeinsam mit der Antwort ausgegeben.

Nachfolgend ein Beispiel fuer ASK:

```
DISK_SIZE: ASK ASSEMBLE FOR 8''(=1) or 5 1/4''(=0) DRIVES?:
```

1.10.3. Assemblierungsmodus

RADIX EXP

Setzt die Assembler Zahlenbasis wie folgt:

```
 2 oder B = Binaer
 8 oder Q = Oktal
10 oder D = Dezimal
16 oder H = Hexadezimal
```

Wird kein Ausdruck angegeben, so wird der Standardmodus, also Basis 10, voreingestellt. Es wird vorausgesetzt, dass jede andere Basis entsprechend mit B, Q, D oder H hinter der Konstanten gekennzeichnet wird.

COMMENT X

Weist den Assembler an, ausgehend von dieser Zeile alle nachfolgenden Zeilen bis zum naechsten 'X' als Kommentar anzusehen. 'X' kann jedes beliebige Zeichen sein.

LABEL: CODE

Bezeichnet den Beginn eines Abschnittes mit Befehlskode. Diese Steueranweisung kann in Verbindung mit den DATA, EXTRA und STACK Steueranweisungen dazu benutzt werden, separate Programmabschnitte zu erzeugen. Bei Beginn ist der Assembler auf einen CODE-Abschnitt voreingestellt. Wenn sich alle Segmente gegenseitig ueberschneiden, ist es nicht noetig, diese Steueranweisungen zu verwenden. ORG-Steueranweisungen, die in einem CODE-Abschnitt auftreten, gelten nur innerhalb des CODE-Abschnitts. Sie haben keinen Einfluss auf den Befehlszaehler von DATA, EXTRA oder STACK-Abschnitten.

LABEL: DATA

Bezeichnet den Beginn eines Abschnittes mit Programmdaten. Die Marke ist optional, aber wenn sie als Operand verwendet wird, wird sie durch die Segmentnummer, in der sich die Marke befindet, ersetzt. Auf diese Art werden Segmentregister normalerweise geladen. Zum Beispiel koennte der folgende Kode zum Laden des DS-Registers verwendet

werden.

```
DSEG:     .DATA
          BYTE                ;Definiert ein paar Datenwerte

          .CODE
          MOVE      AX,DSEG   ;Laedt die Segmentnummer des
                                DATA-Abschnittes
          MOVE      DS,AX     ;Jetzt kann auf die Daten
                                zugegriffen werden
```

ORG-Steueranweisungen, die in einem DATA-Abschnitt auftreten, wirken sich nur im DATA-Abschnitt aus. Sie haben keinen Einfluss auf irgendeinen CODE, EXTRA oder STACK-Abschnitt. Der Assembler ist auf CODE-Abschnitt voreingestellt. Weitere Hinweise sind der Beschreibung der CODE-Steueranweisung zu entnehmen.

LABEL: EXTRA

Bezeichnet den Beginn eines EXTRA-Abschnittes. Die Marke ist optional, aber wenn sie in einem Befehl verwendet wird, liefert sie die Segmentnummer des Abschnitts. ORG-Steueranweisungen, die in einem EXTRA-Abschnitt auftreten, gelten nur innerhalb des EXTRA-Abschnitts. Sie haben keinen Einfluss auf CODE, DATA oder STACK-Abschnitte. Bei Beginn ist der Assembler auf einen CODE-Abschnitt voreingestellt. Weitere Informationen dazu sind der Beschreibung der CODE-Steueranweisung zu entnehmen.

LABEL: STACK

Bezeichnet den Beginn eines STACK-Abschnittes. Die Marke ist optional, aber wenn sie in einem Befehl verwendet wird, liefert sie die Segmentnummer des Abschnitts. ORG-Anweisungen, die in einem STACK-Abschnitt auftreten, gelten nur innerhalb des STACK-Abschnittes. Sie haben keinen Einfluss auf CODE,DATA oder EXTRA-Abschnitte. Bei Beginn ist der Assembler auf einen CODE-Abschnitt voreingestellt. Weitere Informationen dazu sind der Beschreibung der CODE-Steueranweisung zu entnehmen.

INCLUDE filename

Weist den Assembler an, die angegebene Datei in die Assemblierung einzubeziehen. Dateinamenserweiterungen und direkte Pfadnamen muessen vollstaendig angegeben werden. Jede einbezogene Datei muss eine END-Anweisung haben. Include-Anweisungen sollen nicht verschachtelt sein.

1.10.4. Bedingte Assemblierung

IFZ EXP

Der Assembler wird die auf diese Steueranweisung folgenden Anweisungen bis zu einer ELSE- oder ENDIF-Steueranweisung uebersetzen, falls der Wert von EXP gleich Null ist. Bedingte Anweisungen koennen bis zu 248 Ebenen verschachtelt sein.

IFNZ EXP

Die auf diese Steueranweisung folgenden Anweisungen bis zu einer ELSE- oder ENDIF-Steueranweisung werden uebersetzt, wenn der Wert von EXP ungleich Null ist. Bedingte Anweisungen koennen bis zu 248 Ebenen verschachtelt sein.

IFTRUE EXP

Diese Steueranweisung ist eigentlich die gleiche wie IFNZ, aber sie ist logischer, wenn man Vergleichsoperatoren verwendet. Wenn die angegebene Bedingung "wahr" ist, werden die nachfolgenden Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Ist die Bedingung "nicht wahr", werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung nicht assembliert.

IFFALSE EXP

Diese Steueranweisung ist die gleiche wie IFZ und sie ist das Komplement zu IFTRUE. Wenn die angegebene Bedingung "falsch" ist, werden die nachfolgenden Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Ist die Bedingung "wahr", werden die Anweisungen bis zu einer ELSE oder ENDIF-Anweisung nicht assembliert.

IFDEF LABEL

Auf diese Steueranweisung hin wird die Symboltabelle durchsucht. Wird LABEL gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Wird LABEL nicht gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE oder ENIF-Steueranweisung nicht assembliert.

IFNDEF LABEL

Diese Steueranweisung ist das Komplement zu IFDEF. Die Symboltabelle wird durchsucht und wird LABEL nicht gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE oder ENIF-Anweisung assembliert. Wird LABEL gefunden, dann werden die dieser Anweisung nachfolgenden Anweisungen bis zu einer ELSE- oder ENIF-Steueranweisung nicht assembliert.

IFSAME STRING1,STRING2

Diese Steueranweisung vergleicht STRING1 mit STRING2 und assembliert bedingt, d.h. abhaengig vom Ergebnis des Vergleichs, die dieser Anweisung nachfolgenden Anweisungen. Sind die beiden Zeichenketten identisch, dann werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Sind die Zeichenketten nicht identisch, so werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung nicht assembliert. Es sind zwei Typen von Zeichenketten moeglich, entweder enthalten sie Leerzeichen oder sie enthalten keine. Jedoch muessen die miteinander zu vergleichenden Zeichenketten vom gleichen Typ sein. Wenn die Zeichenketten Leerzeichen enthalten, dann muss der Anfang und das Ende jeder Zeichenkette durch ein Apostroph gekennzeichnet werden, wobei in ihr enthaltene Apostrophe durch die Verwendung von zwei Apostrophen dargestellt werden muessen. Enthalten die Zeichenketten keine Leerzeichen, dann sind die Apostrophe nicht erforderlich. Diese Steueranweisung ist sehr nuetzlich fuer den Vergleich von Makro-Parameterargumenten. In beiden Faellen muessen die Zeichenketten voneinander durch Komma getrennt sein. Es folgen ein paar Beispiele fuer die Verwendung von IFSAME.

```
IFSAME    'test string','test string'
IFSAME    'LUDWIG''S','LUDWIG''S'
IFSAME    HL,DE
```

Im ersten Beispiel oben enthalten die Zeichenketten Leerzeichen. Deshalb muessen sie in Apostrophe eingeschlossen sein. Beim zweiten Beispiel enthalten die Zeichenketten im Innern ein Apostroph, das durch zwei Apostrophe dargestellt wird. Beim dritten Beispiel koennte es sich um das Testen von Registern in einem Makro handeln, und da die Zeichenketten keine Leerzeichen enthalten, muessen sie auch nicht in Apostrophe eingeschlossen sein.

IFDIFF STRING1,STRING2

Diese Steueranweisung ist das Komplement zu IFSAME. Wenn die beiden Zeichenketten nicht identisch sind, werden die Anweisungen nach dieser Anweisung bis zu einer ELSE oder ENDIF-Steueranweisung assembliert. Sind die beiden Zeichenketten identisch, werden die Anweisungen bis zu einer ELSE oder ENDIF-Steueranweisung nicht assembliert. Die Syntaxregeln fuer die Bildung dieser Zeichenketten sind dieselben wie fuer IFSAME. Die Beispiele bei IFSAME zeigen die Anwendung dieser Steueranweisung.

IFEXT LABEL

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wurde diese Marke als external deklariert, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

IFNEXT LABEL

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wurde diese Marke nicht als external deklariert, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

IFABS LABEL

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wenn diese Marke absolut ist (d.h. nicht verschiebbar), werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Externe Marken werden als verschiebbar betrachtet. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

IFREL LABEL

Diese Steueranweisung veranlasst den Assembler, in der Symboltabelle nach der Marke LABEL zu suchen. Wenn diese Marke verschieblich ist , werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Externe Marken werden als verschieblich betrachtet. Es wird eine Fehlermeldung ausgegeben, wenn die Marke nicht gefunden wird.

IFMA EXP

Diese Steueranweisung ist fuer die Verwendung im Innern eines Makros bestimmt. Sie ueberprueft, ob die dem Wert EXP entsprechende Anzahl von Argumenten in der Makroaufrufzeile enthalten ist. Wenn die entsprechende Anzahl von Argumenten vorhanden ist, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Stimmt die Argumentanzahl nicht ueberein, werden die nachfolgenden Anweisungen bis zu einem ELSE oder ENDIF nicht assembliert. Soll das Nichtvorhandensein von Argumenten geprueft werden, wird EXP = 0 verwendet. In diesem Fall werden, sofern in dem Makroaufruf keine Argumente vorhanden sind, die nachfolgenden Anweisungen assembliert. Sind in der Makroaufrufzeile jedoch Argumente angegeben, werden die nachfolgenden Anweisungen nicht assembliert. Beispiele fuer die Verwendung dieser Steueranweisung sind im Abschnitt ueber Makros in diesem Handbuch zu finden.

IFNMA EXP

Diese Steueranweisung ist das Komplement zu IFMA. Sie ueberprueft, ob die dem Wert EXP entsprechende Anzahl von Argumenten in der Makroaufrufzeile enthalten ist. Wenn die entsprechende Anzahl von Argumenten nicht vorhanden ist, werden die dieser Anweisung folgenden Anweisungen bis zu einem ELSE oder ENDIF assembliert. Stimmt die Argumentanzahl ueberein, werden die nachfolgenden Anweisungen bis zu einem ELSE oder ENDIF nicht assembliert. Soll lediglich das

Vorhandensein von Argumenten geprueft werden, wird EXP = 0 verwendet. In diesem Fall werden, sofern in dem Makroaufruf wenigstens ein Argument vorhanden ist, die nachfolgenden Anweisungen assembliert. Sind in der Makroaufrufzeile jedoch keine Argumente angegeben, werden die nachfolgenden Anweisungen nicht assembliert. Beispiele fuer die Verwendung dieser Steueranweisung sind im Abschnitt ueber Makros in diesem Handbuch zu finden.

ELSE

Beginn der Anweisungen, die assembliert werden sollen, falls fuer irgendeine der obigen Steueranweisungen des Typs IF die Bedingung nicht erfuellt ist.

ENDIF

Kennzeichnet das Ende eines bedingt zu assemblierenden Blockes. Stellt der Assembler fest, dass nicht fuer jedes IF ein ENDIF vorhanden ist, gibt er eine Fehlermeldung aus. Da rekursive Makros fast immer durch Steueranweisungen vom Typ IF gesteuert werden, ist es moeglich, dass man die IFCLEAR-Steueranweisung verwenden muss. Der Unterschied zwischen den beiden besteht darin, dass ENDIF immer ausgefuehrt wird, waehrend IFCLEAR nicht ausgefuehrt wird, wenn es sich im Innern eines bedingt zu assemblierenden Blockes befindet, fuer den die Bedingung nicht erfuellt ist.

IFCLEAR

Diese Steueranweisung erfuellt genau dieselbe Funktion wie ENDIF, nur wird sie nicht ausgefuehrt, wenn sie sich innerhalb eines bedingt zu assemblierenden Blockes befindet, fuer den die Bedingung nicht erfuellt ist. Diese Steueranweisung kann in rekursiven Makros verwendet werden, um Uebereinstimmung bei den IF-ENDIF Paaren zu erzielen. Auf diese Weise hat man die Moeglichkeit, das Makro schliesslich zu beenden und trotzdem den Vorteil wahrzunehmen, dass der Assembler das paarweise Vorhandensein von IF-ENDIF ueberprueft. Diese Steueranweisung kann zum gleichen Zweck verwendet werden, wenn ein Makro MACEXIT fuer einen fruehzeitigen Makroaustritt enthaelt, denn fast immer wird das durch irgendeine IF-Steueranweisung eingeleitet.

1.10.5. Steuerung der Listenausgabe

LIST ON

Schaltet die Listenausgabe ein, falls beim Aufruf des Assemblers die Steueranweisung "Direktive Listing" (Listenausgabe gemaess Assemblersteueranweisung) eingegeben wurde. Diese Steueranweisung muss immer verwendet werden, bevor man LIST OFF benutzt. Mit anderen Worten, beim Start des Programms wird immer LIST OFF vorausgesetzt.

LIST OFF

Schaltet die Listenausgabe aus, falls die Steueranweisung "Direktive Listing" (Listenausgabe gemaess Assemblersteueranweisung) eingegeben und LIST ON ausgefuehrt wurde. Das ist der voreingestellte Standardmodus und deshalb sollte diese Steueranweisung nur verwendet werden, wenn vorher schon ein LIST ON gegeben wurde.

MACLIST ON

Schaltet die Listenausgabe fuer Makroeinfuegungen ein. Das ist der voreingestellte Standardmodus.

MACLIST OFF

Schaltet die Listenausgabe fuer Makroeinfuegungen aus. Durch Voreinstellung ist sie eingeschaltet.

CONDLIST ON

Schaltet die Listenausgabe fuer bedingt zu assemblierende Bloecke, deren Bedingung nicht erfuellt ist, ein. Das ist der voreingestellte Standardmodus.

CONDLIST OFF

Schaltet die Listenausgabe fuer bedingt zu assemblierende Bloecke, deren Bedingung nicht erfuellt ist, aus. Durch Voreinstellung ist sie eingeschaltet.

PASS1 ON

Schaltet die Listenausgabe fuer den 1.Durchlauf ein. Man kann diese Moeglichkeit nutzen, um Fehler zu finden, die dadurch entstanden sind, dass der Assembler beim ersten Durchlauf einen anderen Weg genommen hat, als beim zweiten. Gewoehnlich wird dieser Umstand zu dem Fehler 'Symbolwert zwischen den Durchlaeufen gewechselt' fuehren. Die Steueranweisung ist aber auch von Nutzen, wenn Fehler bei verschachtelter bedingter Assemblierung gefunden werden sollen.

PASS1 OF

Schaltet die Listenausgabe fuer den 1.Durchlauf aus, vorausgesetzt, es wurde vorher PASS1 ON ausgefuehrt.

PAGE

Gibt einen Seitenvorschub an das Listenausgabegeraet aus.

TITLE STRING

Veranlasst, dass 'STRING' am Anfang jeder Seite als Titelzeile gedruckt wird. Wird 'STRING' nicht angegeben, so wird die 'TITLE'-Steueranweisung ausgeschaltet. Der Titel kann so oft wie gewuenscht geaendert und zu jedem Zeitpunkt ausgeschaltet werden. Die maximale Titellaenge betraegt 80

Zeichen. Die ersten beiden Tabulatoren zwischen der 'TITLE'-Steueranweisung und der STRING-Zeichenkette werden, sofern sie vorhanden sind, ignoriert. Alle nachfolgenden Leerzeichen und Tabulatoren bleiben in der Titelzeile enthalten.

SUBTITLE STRING

Veranlasst, dass 'STRING' am Anfang jeder Seite gedruckt wird. Wenn 'TITLE' ausgefuehrt wurde, wird der Untertitel darunter erscheinen. Wenn 'TITLE' nicht ausgefuehrt wurde oder ausgeschaltet wurde, wird der Untertitel dennoch ausgegeben. Wird 'STRING' nicht angegeben, so wird die Steueranweisung ausgeschaltet. Der Untertitel kann so oft wie gewuenscht geaendert und zu jedem Zeitpunkt ausgeschaltet werden. Die maximale Laenge des Untertitels betraegt 80 Zeichen. Wie bei der 'TITLE'-Steueranweisung werden die ersten beiden Tabulatoren zwischen der SUBTITLE-Steueranweisung und dem Anfang der 'STRING'-Zeichenkette, sofern sie vorhanden sind, ignoriert. Alle danach erscheinenden Leerzeichen oder Tabulatoren bleiben in der Untertitelzeile enthalten.

PW EXP

Legt die Breite der Druckerseite fest. Die voreingestellte Breite einer Seite betraegt 132 Spalten.

PL EXP

Legt die Laenge der Druckerseite fest. Die voreingestellte Laenge einer Seite betraegt 61 Zeilen. Der Assembler loest einen Seitenvorschub aus, wenn dieser Grenzwert erreicht oder ueberschritten wird. Wenn ein Fehler auftritt, gibt der Assembler den Seitenvorschub nach der Fehlermeldung aus.

TOP EXP

Diese Steueranweisung bestimmt die Zeilenanzahl zwischen dem Seitenbeginn und der Seitenzahl. Der voreingestellte Wert betraegt Null.

1.11. Arithmetische und logische Operatoren

Die nachfolgende Liste enthaelt die zulaessigen arithmetischen und logischen Operatoren fuer die Berechnung von Ausdruecken. Dazu wird auch ihre Prioritaetsebene angegeben. Operationen der Prioritaetsebene 7 werden als erste ausgefuehrt. Durch die Verwendung von Klammern kann die Reihenfolge bei der Ausfuehrung von Berechnungen veraendert werden. Alle Berechnungen werden unter Verwendung einer 32-Bit-Integer-Arithmetik ausgefuehrt, mit Ausnahme der Potenzierung, die nur einen 8-Bit-Exponenten verwendet. In den meisten Faellen werden keine Kontrollen auf Ueberlauf vorgenommen. Die maximale Anzahl fuer noch andauernde Operationen ist 16. Leerzeichen zwischen den Argumenten

sind nicht erlaubt.

OPERATION	PRIORITAET	BESCHREIBUNG
Unaeres +	7	Optionale Kennzeichnung eines positiven Operanden
Unaeres -	7	Macht den folgenden Ausdruck negativ
\ oder .NOT.	7	Bildet das Komplement des folgenden Ausdrucks
**	6	Vorzeichenlose Potenzierung
*	5	Vorzeichenlose Multiplikation
/	5	Vorzeichenlose Division
.MOD.	5	Rest
.SHR.	5	Verschiebe den vorhergehenden Ausdruck so oft nach rechts (mit 0 aufgefuellt), wie in dem nachfolgenden Ausdruck angegeben ist.
.SHL.	5	Verschiebe den vorhergehenden Ausdruck so oft nach links (mit 0 aufgefuellt), wie in dem nachfolgenden Ausdruck angegeben ist.
+	4	Addition
-	4	Subtraktion
& oder .AND.	3	Logisches UND
^ oder .OR.	2	Logisches ODER
.XOR.	2	Logisches exklusives ODER

1.12. Vergleichsoperatoren

Die folgende Liste enthaelt die zulaessigen Vergleichsoperatoren. Wird die Vergleichsbedingung erfuellt, so ist das Ergebnis des Vergleichs 1. Ist der Vergleich falsch, d.h. die Bedingung wird nicht erfuellt, so ist das Vergleichsergebnis 0 :

```
=  oder .EQ.  -  gleich
>  oder .GT.  -  groesser als
<  oder .LT.  -  kleiner als
   .UGT.      -  vorzeichenlos groesser als
   .ULT.      -  vorzeichenlos kleiner als
```

1.13. Makros

1.13.1. Definition

Ein Makro ist eine Folge von Quellprogrammzeilen, die fuer eine einzelne Quellprogrammzeile eingesetzt werden. Ein Makro muss definiert werden, bevor es verwendet werden kann. Der Assembler speichert die Makrodefinition und setzt dann, beim Auftauchen des Makronamens, die vorher definierten Quellprogrammzeilen ein. Argumente koennen in der Makrodefinition enthalten sein.

In Makrodefinitionen sollen die Formalargumente keine Leerzeichen enthalten. In den eigentlichen Makroaufrufen

koennen die Argumente von jedem Typ sein: direkt, indirekt, Zeichenketten oder Register. Leerzeichen sind in Argumenten nicht erlaubt, es sei denn, es ist eine Ascii-Zeichenkette. In diesem Fall muss die Zeichenkette in Apostrophe eingeschlossen sein. Enthaelt die Zeichenkette selbst ein Apostroph, so wird dies durch zwei nebeneinander stehende Apostrophe dargestellt. Argumente werden an beliebig verschachtelte Makros uebergeben, wenn die Namen der Formalargumente identisch sind. Die Verschachtelung von Makros wird nur durch den verfuegbaren Disk-Speicherplatz begrenzt.

Um ein Makro zu definieren, wird die ".MACRO"-Steueranweisung verwendet. Ein Makro muss, anschliessend an die Makrodefinition, die Steueranweisung ".MACEND" oder ".ENDM" enthalten. Der Name des Makros steht im Markenfeld.

1.13.2. Trennzeichen fuer Argumente

In der Makroaufrufzeile muessen die Argumente voneinander durch Komma getrennt werden. Fuehrende Leerzeichen und Tabulatoren werden ignoriert. Fuer ein nicht vorhandenes Argument kann ein einzelnes Komma als Platzhalter dienen. In einem Makrokoerper sind die folgenden Trennzeichen fuer Argumente zulaessig:

, + - * / ** \ & ^ = < > ( ) [ ] |

.NOT. .AND. .OR. .XOR. .EQ. .GT. .LT. .UGT.

.ULT. .SHR. .SHL.

1.13.3. Verkettung

Der Senkrechtstrich (| = hex 7C) wird als Verkettungsoperator fuer Zeichenketten verwendet.

1.13.4. Marken in Makros

Marken sind in Makrodefinitionen zugelassen. Marken koennen auf zwei Arten definiert sein: explizit oder implizit. Explizite Marken werden in der Makrodefinition durch den Assembler nicht veraendert. Implizite Marken haben am Ende ein "#" stehen. Der Assembler ersetzt "#" durch eine dreistellige Zahl, die Makroerweiterungsnummer. In diesem Fall darf die Marke mit der Makroerweiterungsnummer nicht laenger als 10 Zeichen sein. Endet eine Marke, die sich ausserhalb eines Makros befindet, mit dem Symbol "#", so erhaelt sie die Makroerweiterungsnummer des letzten Makros. Damit ist es moeglich, Marken umzubenennen, ohne die tatsaechliche Makroerweiterungsnummer zu kennen. Zur Bezeichnung einer Marke darf kein Argument benutzt werden. Falls das erforderlich ist, setzt man im Innern des Makros das Argument einer Marke gleich. Wenn das Argument ein Parameter ist, der im Innern des Makros veraendert wird, kann man

es einer Variablen gleichsetzen, indem man die 'VAR'-Steueranweisung verwendet. Siehe dazu das Beispiel in Abschnitt 1.13.6.

1.13.5. Umdefinieren von Mnemoniks

Die Assemblertabellen werden in der folgenden Reihenfolge durchsucht:

1. Tabelle der Assemblersteueranweisungen
2. Tabelle der Makrodefinitionen
3. Tabelle der K1810WM86-Mnemoniks

Dadurch ist es moeglich, K1810WM86-Mnemoniks umzudefinieren, indem man Makros verwendet. Assemblersteueranweisungen koennen hinzugefuegt, aber nicht umdefiniert werden, solange ihnen kein Dezimalpunkt vorangestellt wird. Der Grund dafuer ist die Tatsache, dass jedes Mnemonikfeld, das mit einem Dezimalpunkt beginnt, automatisch fuer eine eingebaute Assemblersteueranweisung gehalten wird.

1.13.6. Rekursive Abarbeitung

Wenn Makros verschachtelt werden, so werden alle Informationen, die fuer die Rueckkehr in die vorhergehende Quellenkodeumgebung benoetigt werden, auf dem Disk-Speicher gerettet. Diese Verfahrensweise macht es dem Assembler moeglich, Makros auf dem Disk-Speicher zu speichern und Verschachtelungen bis zu jeder gewuenschten Tiefe zu gestatten. Rekursionen (ein Makro ruft sich selbst auf) sind erlaubt. Sie werden in der Weise realisiert, dass festgestellt wird, dass das Makro bereits aktiv ist und daraufhin das Abspeichern der vorhergehenden Quellenkodeumgebung verhindert wird. Jedoch ist es nicht gestattet, dass ein Makro ein anderes Makro aufruft, welches wiederum das erste Makro aufruft. Wenn das geschieht, wird die urspruengliche Rueckkehrinformation zerstoert und der Assembler ist nicht mehr in der Lage, in die urspruengliche Umgebung zurueckzukehren. Darueber hinaus ist es dem Assembler auch nicht moeglich, diese Bedingung festzustellen, weil er, wenn das zweite Makro das erste, das ja schon aktiv ist, aktiviert, einen rekursiven Makrozustand vorraussetzt und die Informationen, die zur Rueckkehr zum zweiten Makro noetig waeren, nicht aufzeichnet.

Unten folgt ein Beispiel fuer ein rekursives Makro, das die Anzahl von Datenbytes, die durch das Formalargument ARG1 bestimmt werden, reserviert und mit dem Wert fuellt, der durch ARG2 bestimmt wird.

```
FILL:     .MACRO    ARG1,ARG2
COUNT:    .VAR      ARG1          ;Speichere Byteanzahl
          .IFZ      COUNT         ;Kontrolle der  Ausfueh-
                                   rung
          .IFCLEAR                ;Fuer   paarweises   IF-
                                   ENDIF sorgen
```

```
          .MACEXIT                ;Austritt
          .ENDIF
COUNT:    .VAR      COUNT-1       ;Else      dekrementiere
                                    Bytezaehler
          .BYTE     ARG2          ;Reserviert das Byte
          FILL      COUNT,ARG2    ;Erneuter Durchlauf
          .MACEND
```

Dieses Makro wuerde man mit einer Anweisung wie der folgenden aufrufen:

```
          FILL      10,55H        ;Reserviere 10 Bytes und
                                    speichere in jedem  55
                                    hex
```

Fuer ein rekursives Makro, wie beispielsweise das oben beschriebene, ist es durchaus zulaessig, ein weiteres rekursives Makro aufzurufen usw., bis zu jeder gewuenschten Tiefe. Man beachte auch die Verwendung der IFCLEAR-Steueranweisung, die die Paarigkeit der bedingten IF- ENDIF-Steueranweisungen aufrechterhaelt. Das ist erforderlich fuer den Fall, dass die MACEXIT-Steueranweisung ausgefuehrt wird, da dann die ENDIF-Steueranweisung nicht ausgefuehrt wird.

1.14. Assembler-Fehlernachrichten

1.14.1. Programierungsfehler

In diesem Abschnitt wird eine Liste von Fehlernachrichten, die der Assembler ausgibt, zum Nachschlagen bereitgestellt. Es wird zusaetzlich eine moegliche Ursache fuer die Fehlernachricht angegeben, sowie ein Beispiel fuer eine Kodierung, die diesen Fehler verursachen koennte. Der Grund dafuer, dass keine genaueren Angaben gemacht werden, liegt darin, dass bei manchen Fehlern Abhaengigkeiten zu vorhergehenden Ereignissen bestehen.

```
Fehler      -  ILLEGAL REGISTER SIZE
               (Unzulaessige Registergroesse)
Bedeutung   -  Registergroesse unvereinbar mit der Verwendung
Beispiel    -  MOV   AL,[BL]        ;Sollte BX sein


Fehler      -  ILLEGAL REGISTER #
               (Unzulaessige Register #)
Bedeutung   -  Register ist nicht erlaubt.
Beispiel    -  ADD   AL,[CX]        ;CX kann nicht als Quelle verwendet werden


Fehler      -  SYNTAX ERROR
               (Syntaxfehler)
Bedeutung   -  Gewoehnlich ein fehlendes Komma oder Klammer.
Beispiel    -  MOV   AL,DATA]       ;Linke Klammer fehlt
```

Fehler - CAN'T RESOLVE OPERAND
(Operand kann nicht aufgeloest werden)
Bedeutung - Es ist nicht erkennbar, was der Programmierer beabsichtigte.
Hinweis - Dieser Fehler wird angezeigt, wenn der Assembler das Operandenfeld aufloest und kein vordefiniertes Muster findet.

Fehler - ILLEGAL ADDRESSING MODE
(Unzulaessige Adressierungsart)
Bedeutung - In dieser Form kann der Operand nicht adressiert werden.
Beispiel - MOV AL,[BX+CX] ;CX ist als Indexregister unzulaessig

Fehler - MULTIPLY DEFINED SYMBOL
(Mehrfach definiertes Symbol)
Bedeutung - Das Symbol wurde vorher definiert ('.VAR' ist davon ausgenommen).
Beispiel - LABEL: .EQUAL $;1. Definition
LABEL: .EQUAL $;2. Definition

Fehler - ILLEGAL MNEMONIC
(Unzulaessige Mnemonik)
Bedeutung - Mnemonik existiert nicht und wurde nicht als Makro definiert.
Beispiel - LD AL,2 ;Es gibt kein LD

Fehler - ILLEGAL BINARY #
(Unzulaessige Binaerzahl)
Bedeutung - Die Zahl ist nicht 0 oder 1.
Beispiel - MOV AL,02B ;2 ist keine Binaerzahl
Hinweis - Dieser Fehler ist auch moeglich, wenn ein nicht existierendes Symbol mit einem 'B' endet.

Fehler - ILLEGAL OCTAL NUMBER
(Unzulaessige Oktalzahl)
Bedeutung - Die Zahl liegt nicht zwischen 0 und 7.
Beispiel - MOV AL,1FQ ;F ist keine Oktalzahl
Hinweis - Dieser Fehler ist auch moeglich, wenn ein nicht existierendes Symbol mit einem 'Q' endet.

Fehler - ILLEGAL HEX #
(Unzulaessige Hexadezimalzahl)
Bedeutung - Die Zahl liegt nicht zwischen 0 und 15.
Beispiel - MOV AL,0GH ;G ist keine Hexadezimalzahl
Hinweis - Dieser Fehler ist auch moeglich, wenn ein nicht existierendes Symbol mit einem 'H' endet.

Fehler - # TOO LARGE
(Zahl ist zu gross)
Bedeutung - Der Bestimmungsort ist zu klein fuer den Operanden.
Beispiel - MOV AL,123H ;123H laesst sich mit 8 Bit nicht darstellen
Hinweis - Das kann der Fall sein, wenn die Berechnung zu einem negativen Ergebnis fuehrt, z.B. bei der Subtraktion einer groesseren von einer kleineren Adresse. Der Assembler zeigt eine Ueberschreitung nicht in jedem Fall an, da er nicht weiss, ob die Zahl ein Vorzeichen traegt oder nicht. Mit der AND-Funktion kann das Ergebnis auf die passende Groesse reduziert werden.

Fehler - HEX # AND SYMBOL ARE IDENTICAL
(Hexadezimalzahl und Symbol sind identisch)
Bedeutung - Es existiert eine Marke, die voellig identisch mit einer als Operand verwendeten Hexadezimalzahl ist. Der Fehler tritt nur auf, wenn sich auch das zur Kennzeichnung der Hexadezimalzahl verwendete 'H' an der gleichen Stelle befindet.
Beispiel - ABCH: NOP ;Definiere Marke
MOV BX,ABCH ;Der Assembler kann nicht zwischen der Marke und der gueltigen Hexadezimalzahl unterscheiden.

Fehler - UNDEFINED SYMBOL
(Undefiniertes Symbol)
Bedeutung - Das Symbol wurde waehrend des 1.Durchlaufs nicht definiert.
Beispiel - LABEL: .BYTE ;Die Marke LABEL wird definiert
MOV BX,LABLE ;Durch den Schreibfehler wird die Marke nicht erkannt

Fehler - ILLEGAL ASCII DESIGNATOR
(Ungueltiger Ascii-Bezeichner)
Bedeutung - Schlechte Zeichensetzung bei Ascii-Zeichen.
Beispiel - MOV AL,"T' ;Die Einschliessungszeichen muessen vom gleichen Typ sein, entweder " oder '.

Fehler - END OF OPERAND EXPECTED BUT NOT FOUND
(Ende des Operanden erwartet, aber nicht gefunden)
Bedeutung - Gewoehnlich ein Syntax- oder Formatfehler.
Hinweis - Dieser Fehler wird bei der letzten Kontrolle, die der Assembler an jedem Befehl vornimmt,

bevor er zur naechsten Zeile geht, festgestellt. Er zeigt an, dass nach dem gueltigen Ende des Operanden zusaetzliche Zeichen vorhanden sind.

Fehler - ILLEGAL ASSEMBLER DIRECTIVE
(Ungueltige Assembler-Steueranweisung)
Bedeutung - Die Assembler-Steueranweisung existiert nicht.
Beispiel - .BLK 100 ;Fehlendes B oder W

Fehler - CAN'T RECOGNIZE # BASE
(Zahlenbasis nicht erkennbar)
Bedeutung - Es wurde eine andere Zahlenbasis als B, Q, D oder H angegeben.
Beispiel - LD R0,101O ;Oktal mit Q kennzeichnen

Fehler - ATTEMPTED DIVISION BY ZERO
(Versuchte Division durch Null)
Bedeutung - Der Operand fuer den Divisor hat den Wert 0.
Beispiel - DATA: .EQUAL 10/(8-(2**3))

Fehler - SYMBOL EXCEEDS 10 CHARACTERS
(Symbol besteht aus mehr als 10 Zeichen)
Bedeutung - Die maximale Symbollaenge betraegt 10 Zeichen.
Beispiel - DAS_IST_ZU_LANG:

Fehler - OPERAND NOT IN CURRENT SEGMENT
(Operand befindet sich nicht im aktuellen Segment)
Bedeutung - Die Segmentnummer des Operanden, auf den Bezug genommen wird, entspricht nicht den Segmentnummern, die durch die letzten ORG- oder ASSUME-Steueranweisungen angegeben wurden, und sie entspricht auch nicht der Segmentnummer 0000, falls keine ORG- oder ASSUME-Steueranweisungen ausgefuehrt wurden. Im Abschnitt ueber die Assemblersteueranweisungen kann ueber die Verwendung dieser beiden Steuranweisungen nachgelesen werden.

Fehler - OPERAND SIZE MISMATCH
(Unangepasste Operandengroesse)
Bedeutung - Der Quellen- und der Zieloperand haben unterschiedliche Groesse.
Beispiel - DATABYTE: .BYTE ;Definiert 8-Bit-Speicher
MOV DATABYTE,AX
;AX wuerde nicht in 8 Bit passen

Fehler - 'BYTE PTR' OR 'WORD PTR' NEEDED
('BYTE PTR' oder 'WORD PTR' notwendig)
Bedeutung - Die Operandengroesse kann durch das Operandenfeld des Befehls nicht bestimmt werden.
Beispiel - MOV [BX],10 ;Nicht eindeutig, ob ein Byte oder ein Wort gespeichert werden soll

Fehler - SYMBOL NEEDS TO BE DEFINED BEFORE BEING USED
(Symbol muss vor Verwendung definiert werden)
Bedeutung - Beim ersten Durchlauf war das Symbol noch nicht definiert. Deshalb machte der Assembler eine bestimmte Annahme, die die Anzahl der erzeugten Bytes beeinflusste. Beim zweiten Durchlauf wurde das Symbol gefunden und der Assembler stellte fest, dass die im ersten Durchlauf gemachte Annahme nicht nur falsch war, sondern dass es auch keine Moeglichkeit gab, die gleiche Anzahl Bytes wie im ersten Durchlauf zu erzeugen.
Beispiel - ADD AX,DATA8 ;Der Assembler haelt DATA8 fuer eine Adresse, fuer die 4 Bytes ausgegeben werden muessen.
DATA8: .EQUAL 10 ;DATA8 ist als unmittelbarer Datenwert definiert, ausgedrueckt durch einen 3-Byte-Befehl

Fehler - TITLE/SUBTITLE EXCEEDS 80 CHARACTERS
(Titel/Untertitel laenger als 80 Zeichen)
Bedeutung - Die Puffergroesse ist fuer 80-Byte-Titel und Untertitel angelegt.
Beispiel - .TITLE ' ... (81 Bytes dahinter) ...'

Fehler - SYMBOL VALUE CHANGED BETWEEN PASSES
(Symbolwert zwischen den Durchlaeufen veraendert)
Bedeutung - Der im ersten Durchlauf ermittelte Symbolwert entspricht nicht dem im zweiten Durchlauf ermittelten.
Beispiel - LABEL: .EQUAL LABEL2
;LABEL2 noch nicht definiert (betrifft nicht '.VAR')

Fehler - PARAMETERS EXCEED 80 CHARACTERS
(Parameter laenger als 80 Zeichen)
Bedeutung - Die maximale Parameterlaenge betraegt 80 Zeichen.
Beispiel - TEST: .MACRO ARG1,ARG2 ...(80 Zeichen dahinter)... ARG80

Fehler - NESTED CONDITIONAL ASSEMBLY UNBALANCE DETECTED
(Unpaarigkeit in verschachtelter bedingter Assemblierung festgestellt)
Bedeutung - Beliebiger '.IF'-Typ-Befehl ohne das dazugehoerige '.ENDIF'.
Beispiel -
```
.IFNZ 1 ;erstes IF
.IFNZ 1 ;zweites IF
NOP ;unmittelbarer Kode
.ENDIF ;nur ein ENDIF
```
Bemerkung - Man beachte die Hinweise zur Verwendung der '.IFCLEAR'-Steueranweisung , die im Abschnitt ueber die Assembler-Steueranweisungen gegeben wurden.

Fehler - NESTED MACRO UNBALANCE DETECTED
(Unpaarigkeit bei verschachteltem Makro festgestellt)
Bedeutung - Zusaetzliches .ENDM bei einer Makroerweiterung gefunden.
Beispiel -
```
MACRO1 ;Makroaufruf
.ENDM ;In der Makrodefinition ist schon ein .ENDM enthalten
```

1.14.2. Systemfehler

Die folgenden Fehler sind keine eigentlichen Programmierungsfehler, aber sie werden vom Assembler gefunden. In manchen Faellen koennen diese Fehlernachrichten helfen, unerwartete Ergebnisse zu erklaeren, besonders, wenn man sie in Verbindung mit den Steueranweisungen fuer den ersten Durchlauf benutzt.

Fehler - MISSING END STATEMENT
(Fehlende END-Anweisung)
Bedeutung - Am Ende einer Datei oder eines eingeschlossenen Moduls wurde kein .END gefunden.

Fehler - CAN'T FIND INCLUDE FILE
(Einzubeziehende Datei kann nicht gefunden werden)
Bedeutung - Ein Datei- oder Directory-Pfadname ist inkorrekt.

Fehler - ILLEGAL NESTED INCLUDE
(Unzulaessige Verschachtelung bei einer INCLUDE-Anweisung)
Bedeutung - Eine einbezogene Datei, die eine INCLUDE-Steueranweisung enthaelt. Dieser Fehler kann aber auch darauf hinweisen, dass eine einbezogene Datei keine END-Anweisung enthaelt.

Fehler - NOT ENOUGH PARAMETERS
(Nicht genuegend Parameter)
Bedeutung - Der Assembler wurde mit einer Kommandozeile aufgerufen, die nicht genuegend Informationen enthielt. Man kontrolliere besonders die Zielangaben fuer die Listenausgabe und die Antworten auf die .ASK-Steueranweisung.

Fehler - OPEN FILE ERROR
(Fehler beim Eroeffnen einer Datei)
Bedeutung - Der Assembler kann die Datei nicht eroeffnen. Weist gewoehnlich auf eine nicht existierende Datei hin.

Fehler - CREATE NEW FILE ERROR
(Fehler beim Anlegen einer neuen Datei)
Bedeutung - Der Assembler ist nicht in der Lage, die Ausgabedatei anzulegen. Das geschieht gewoehnlich infolge einer vollen Diskette. Man beachte, dass der Assembler 6 Dateien anlegt, und dass dieser Fehler selbst bei kleinen Programmen auftreten kann. Stellt der Assembler das fest, so loescht er alle Dateien, die er bis dahin angelegt hat, so dass der Eindruck entstehen kann, auf dem Disk-Speicher waere noch Platz. Es muss mindestens das Sechsfache der kleinsten zugewiesenen Blockgroesse verfuegbar sein.

Fehler - WRITE ERROR
(Fehler beim Schreiben)
Bedeutung - Die wahrscheinlichste Ursache ist ein voller Disk-Speicher. Siehe auch CREATE NEW FILE ERROR.

Fehler - CLOSE FILE ERROR
(Fehler beim Schliessen der Datei)
Bedeutung - Dieser Fehler weist darauf hin, dass der Assembler eine der sechs von ihm erzeugten Dateien nicht schliessen kann, gewoehnlich wegen unzureichenden Speicherplatzes auf dem Disk-Speicher oder in der Directory. Siehe auch CREATE NEW FILE ERROR.

Fehler - RANDOM RECORD READ ERROR
(Fehler beim zufaelligen Lesen eines Records)
Bedeutung - Dieser Fehlertyp wird gefunden, wenn unzulaessige Operationen mit Daten durchgefuehrt werden, die auf dem Disk-Speicher gespeichert sind, z.B. wenn sich zwei Makros gegenseitig aufrufen.

2. Linker

2.1. Beschreibung des Linkers

Der Linker gestattet dem Anwender, Programme in der K1810WM86-Assemblersprache zu schreiben, die aus mehreren Modulen bestehen. Der Linker loest externe Adressbezuege auf und fuehrt Adressverschiebungen durch. Dabei wird die Anzahl der globalen Symbole, der externen Symbole und die Groesse des Objektkodes nur durch den verfuegbaren Disk-Speicherplatz begrenzt. Der Linker arbeitet, wie unten beschrieben, entweder im "Promt-Modus" oder im "Submit-Modus".

Um den unterschiedlichen Ausgabeanforderungen gerecht zu werden, werden mehrere Linkerversionen zur Verfuegung gestellt. Diese verschiedenen Versionen, die durch unterschiedliche Namen gekennzeichnet sind, werden nachfolgend beschrieben.

link1 - DCP Load File

Der Linker mit dem Namen 'link1' wird verwendet, wenn das Programm unter DCP laufen soll und die Datei getrennte CODE-, DATA-, STACK- und EXTRA-Abschnitte enthaelt. Der Linker erzeugt eine Datei mit einer Kopfinformation (header), die alles enthaelt, was DCP benoetigt, um das Programm in jedem verfuegbaren Speicherblock ausreichender Groesse laufen zu lassen. Die Standardnamenserweiterung von 'link1' fuer die Ausgabedatei ist 'exe'. Die maximale Anzahl von Eingabedateien fuer 'link1' betraegt 128.

link2 - SCP86 Load File

Der Linker mit dem Namen 'link2' wird verwendet, wenn das Programm unter SCP86 laufen soll und die Datei getrennte CODE-, DATA-, STACK- und EXTRA-Abschnitte enthaelt. Der Linker erzeugt eine Datei mit einer Kopfinformation (header), die alles enthaelt, was SCP86 benoetigt, um das Programm in jedem verfuegbaren Speicherblock ausreichender Groesse laufen zu lassen. Die Standardnamenserweiterung von 'link2' fuer die Ausgabedatei ist 'cmd'. Die maximale Anzahl von Eingabedateien fuer 'link2' betraegt 256.

link3 - ausfuehrbare Ausgabedatei

Der Linker mit dem Namen 'link3' wird verwendet, wenn die gewuenschte Ausgabe eine ausfuehrbare Objektkodedatei sein soll, bei der die CODE-, DATA-, STACK- und EXTRA-Abschnitte alle in demselben 64 K Block des Speichers liegen. Das Programm wird gebunden und auf die gewuenschte Abarbeitungsadresse verschoben. Luecken, die von ORG-Steueranweisungen herruehren, werden mit Bytes aufgefuellt, die den Wert Null enthalten. Nur ORG-Steueranweisungen mit aufsteigenden Adresswerten werden korrekt behandelt. Das Auffuellen der durch die ORG-Steueranweisungen bedingten Luecken ist erforderlich, um eine ausfuehrbare Datei zu erhalten. ORG-Steueranweisungen in abfallender Reihenfolge

werden wahrscheinlich eine Ausgabedatei erzeugen, die infolge der riesigen aufzufuellenden Luecken viel groesser als erwartet sein wird. Wenn die ORG-Anweisungen in abfallender Reihenfolge sind und/oder wenn die durch ORG-Anweisungen bedingten Luecken nicht aufgefuellt werden sollen, darf 'link3' nicht verwendet werden. Die Standardnamenserweiterung fuer die Ausgabedatei von 'link3' ist 'tsk'. Die maximale Anzahl von Eingabedateien fuer 'link3' betraegt 256.

link5 - 'Extended I-Hex'-Ausgabedatei

Der Linker mit dem Namen 'link5' wird verwendet, wenn die gewuenschte Ausgabedatei das Format 'Extended I-Hex' haben soll. Der Linker wird alle Eingabemodule verbinden und verschieben, aber von ORG-Steueranweisungen herruehrende Luecken werden dabei nicht aufgefuellt. Stattdessen wird ein neuer 'I-Hex'-Datensatz (record) begonnen, und sollte die neue Segmentnummer unterschiedlich zu der vorhergehenden sein, so wird ein neuer Segmentnummer-Datensatz erzeugt. Deshalb koennen ORG-Anweisungen auch in abfallender Reihenfolge auftreten. Die Ausgabedatei ist nicht in aufsteigender Reihenfolge sortiert, sondern erscheint genauso, wie es die Linker-Ladetabelle zeigt. Die Standardnamenserweiterung fuer die Ausgabedatei von 'link5' ist 'hex'. Die maximale Anzahl von Eingabedateien fuer 'link5' betraegt 256.

2.2. Linker Bedienungsanweisungen

In der nachfolgenden Beschreibung wird vorausgesetzt, dass der Nutzer die Bezeichnung fuer den entsprechenden Linker jeweils in 'link' umbenannt hat.

2.2.1. Prompt-Modus

Um den Linker aufzurufen, wird eingegeben: link

Das Programm antwortet mit: INPUT FILENAME :

Der Nutzer gibt einen Dateinamen einschliesslich der Namenserweiterung ein. Wird nur der Dateiname eingegeben, so nimmt der Linker an, dass die Namenserweiterung '.obj' ist, weil das die Standardnamenserweiterung des Assemblers fuer die Ausgabedatei ist.

Nachdem der Nutzer den Dateinamen eingegeben hat, fragt das Programm nach der Ladeadresse. Alle Adressen innerhalb der Datei werden entsprechend dieser Adresse veraendert, d.h., es erfolgt eine Adressenverschiebung. Es wird erwartet, dass die Ladeadresse fuer die erste Datei angegeben wird, aber danach wuenscht der Nutzer normalerweise, dass jede weitere Datei jeweils dort beginnt, wo die vorhergehende aufhoerte. Mit anderen Worten, sie werden 'dicht gepackt'. Um Dateien dicht zu packen, wird auf die Eingabeanforderung fuer die Ladeadresse nur ein 'CR' (Wagenruecklauf) eingegeben.

In den Faellen, in denen eine Datei globale Symbolwerte enthaelt, die von einem anderen Programm benoetigt werden, und ihr eigener Objektkode aber nicht gebraucht wird (z.B. bei Ueberlagerungsprogrammen), wird vor die einzugebende Ladeadresse ein Minuszeichen (-) gesetzt. Der Linker wird dann die globalen Symbolwerte verwenden, aber der Objektkode dieser Datei wird nicht in die Ausgabedatei des Linkers aufgenommen. Dieser Typ von Datei wird als 'Nur-Verweisdatei' ('Reference Only') bezeichnet und erscheint auch als solche in der Ladetabelle. Die 'Reference Only'-Datei wird vollstaendig gebunden und die Adressenverschiebung wird vorgenommen, d.h., der Linker kann auch 'Reference Only'-Dateien dicht packen. Das wird erreicht, indem das '-'-Zeichen eingegeben wird, um 'Reference Only' zu kennzeichnen, und durch nachfolgende Eingabe von CR, um dem Linker mitzuteilen, dass die Datei unmittelbar an die vorhergehende anzuschliessen ist. Im Abschn. 2.7. werden hierfuer Beispiele angegeben.

Der Linker wird solange zur Eingabe weiterer Dateinamen und Ladeadressen auffordern, bis der Nutzer statt eines Dateinamens CR eingibt. Die Dateinamen sollten in der Reihenfolge eingegeben werden, in der sie fuer die Abarbeitung in den Speicher geladen werden sollen.

Nachdem alle Eingabedateinamen eingegeben wurden, fordert der Linker zur Eingabe des Namens fuer die Ausgabedatei auf. Wird dafuer nur ein CR eingegeben, gibt der Linker der Ausgabedatei den Namen der ersten Eingabedatei, jedoch verlaengert um die jeweilige Namenserweiterung des verwendeten Linkers. Dem Kapitel "Beschreibung des Linkers" (Abschn. 2.1.) koennen die Standardnamenserweiterungen fuer die Ausgabedateien der verschiedenen Linker entnommen werden.

2.2.2. Submit-Modus

Wenn eine grosse Anzahl von Dateien gebunden werden sollen, koennen im Promt Modus leicht Fehler gemacht werden. In solch einem Fall wird der Linker besser unter Verwendung einer Submitdatei abgearbeitet. Das Dateiformat fuer die Arbeit unter dem Submit Modus ist unten angegeben. Im Grunde genommen wird eine Datei erzeugt, die genau die Eingaben enthaelt, die man als Erwiderung auf die Eingabeanforderungen des Linkers im Promt Modus machen wuerde. In den Faellen, in denen man nur ein CR eingeben wuerde, wird das Unterstreichungszeichen verwendet, um in der Datei eine CR-Eingabe zu bewirken.

```
Eingabedateiname
Ladeadresse
Eingabedateiname
Ladeadresse  (oder wenigstens 1 Leerzeichen oder  '_'  fuer
              dicht gepackt)
 ...
 ...
Letzter Dateiname
Letzte  Adresse   (oder  Leerzeichen oder  '_'  fuer  dicht
                   gepackt)
```

Wenigstens 1 Leerzeichen oder '_' fuer "keine weitere Datei"
Ausgabedateinamen (oder '_' fuer Standardnamen)

Hat die Submitdatei den Namen "file", so erfolgt der Aufruf der Linker folgendermassen (*=1, 2, 3 oder 5):

```
link* <file
```

2.3. Linker Arbeitsweise

Dieser Abschnitt beschreibt einige der vom Linker ausgefuehrten Operationen. Eine Beschreibung der verschiedenen Betriebsarten des Linkers kann dem Abschnitt "Linker Beispiele" (s. Abschn. 2.4.) entnommen werden.

2.3.1. Handhabung der urspruenglichen Assembleradressen

Normalerweise enthaelt ein Programm, das verschiebbar sein soll, keine ORG-Steueranweisung und wird deshalb ab der Startadresse 0 assembliert. Der Linker addiert dann die eingegebene Ladeadresse zu der vom Assembler berechneten Adresse, um die der Verschiebung entsprechende Adresse zu erhalten. Jedoch muss die urspruengliche Assembleradresse nicht unbedingt 0 sein. Wurde das Programm ab einer anderen Adresse als 0 assembliert, hat man als Ergebnis die Assembleradresse plus dem Versatz (offset) der Ladeadresse zu erwarten. Die Ladetabelle, die der Linker erzeugt, gibt immer den Wert an, auf den das Programm gebunden und verschoben wurde, um dort abgearbeitet zu werden. Sorgfalt ist geboten, wenn Module mit anderen Modulen, die ORG-Steueranweisungen in abfallender Reihenfolge enthalten, dicht gepackt werden sollen. In solch einem Fall muss der Nutzer dafuer sorgen, dass der Linker eine feste Adresse hat, ab der die Module dicht gepackt werden sollen. Das kann erreicht werden, indem man fuer die Module, die ORG-Anweisungen in abfallender Reihenfolge enthalten, absolute Ladeadressen verwendet und nur jene Module dicht packt, die aufsteigende Adressen enthalten. 'link1' kann niemals fuer Dateien benutzt werden, die abfallende ORG-Anweisungen enthalten.

2.3.2. Getrennte Kode- und Datenadressenangaben

Bei der Angabe von Kode- und Datenadressen wird zuerst die Kode- und dann die Datenadresse angegeben, wobei beide durch ein Komma getrennt sein muessen. Wird keine Datenadresse angegeben, so werden alle Datenabschnitte verschoben und, an die Kodeabschnitte anschliessend, dicht gepackt. Die Kodeadresse kann uebersprungen werden, indem man einfach ein Komma eingibt. Wenn keine Kodeadresse, sondern nur eine Datenadresse angegeben wurde, werden alle Kodeabschnitte verschoben und, an die Datenabschnitte anschliessend, dicht gepackt. Kode und Daten koennen unabhaengig voneinander bearbeitet werden, indem man fuer jeden Abschnitt eine eigene Ladeadresse eingibt.

2.3.3. Globale und externe Symbole

Der Linker verbindet die externen Symbole, sofern welche vorhanden sind, mit den globalen (oder allgemein gueltigen) Symbolen. Werden globale Symbole mehrfach angetroffen, so wird eine Fehlermeldung ausgegeben. Der Wert der Ladeadresse (Versatz) wird zum Assemblerwert des globalen Symbols addiert, bevor es als Operand benutzt wird. Wenn das externe Symbol ein 4- oder 8-Bit-Wert ist und das globale Symbol dafuer zu gross ist, wird eine Fehlermeldung mit dem Symbolnamen, mit dem Namen der Datei, in der das Symbol als GLOBAL vereinbart wurde und mit dem Namen der Datei, in der das Symbol als EXTERNAL vereinbart wurde, ausgegeben. Findet sich fuer ein externes Symbol kein entsprechendes globales Symbol, wird eine Fehlermeldung mit dem Namen der Datei, in der das Symbol als EXTERNAL vereinbart wurde, ausgegeben.

2.3.4. Adressenverschiebung

Adressen werden verschoben, indem die Ladeadresse zu dem durch die Assemblierung zugewiesenen Adresswert addiert wird. Normalerweise beginnt die Assemblierung eines Programms bei der Startadresse 0. Aber das muss nicht immer der Fall sein. Deshalb wird die Ladeadresse einfach zu jeder Adresse addiert, die der Assembler erzeugt hat.

Der Assembler erzeugt eine Tabelle, in der die Eigenschaften eines jeden im Programm verwendeten Symbols eingetragen sind. Handelt es sich nur um eine Marke, die vor einem Befehl steht, so ist sie als verschiebbar gekennzeichnet. Ist die Marke in einer ".EQUAL"-Steueranweisung definiert, so haengt ihre Verschiebbarkeit vom Typ des Operandenfeldes ab. Enthaelt der Operand keine verschiebbaren Bestandteile, dann ist der Ausdruck nicht verschiebbar. Enthaelt der Operand nur einen verschiebbaren Bestandteil, dann ist der Ausdruck verschiebbar. Enthaelt der Operand zwei oder mehr verschiebbare Bestandteile, so ist der Ausdruck nicht verschiebbar.

Bytewerte sind nur verschiebbar, wenn das unaere groesser als '>'-Zeichen fuer das hoeherwertige Byte und/oder das unaere kleiner als '<'-Zeichen fuer das niederwertige Byte verwendet wird. Diese Operanden folgen denselben Verschiebungsregeln, wie vollstaendige 16-Bit-Adresswerte.

Die nachfolgenden Beispiele sollen die Erklaerungen deutlicher machen.

```
LABEL1:   NOP       ;Die Marke ist so definiert, dass sie
                     gleich der Adresse des Befehls ist.
                     Deshalb ist sie verschiebbar.

LABEL2:   .EQUAL    LABEL1
                    ;Gemaess Definition soll die Marke
                     gleich einem Wert sein, der als ver-
```

```
                    schiebbar gekennzeichnet wurde. Des-
                    halb ist LABEL2 ebenfalls verschieb-
                    bar.

LABEL3:   .EQUAL    10
                    ;Gemaess Definition ist die Marke
                     gleich einer Konstanten. Deshalb ist
                     LABEL3 nicht verschiebbar.

LABEL4:   .EQUAL    $+10
                    ;Gemaess Definition ist die Marke
                     gleich einem verschiebbaren Wert plus
                     einem nicht verschiebbaren Wert. Da
                     nur ein Wert verschiebbar ist, ist
                     auch das Symbol LABEL4 verschiebbar.

LABEL5:   .EQUAL    10+$
                    ;Gemaess Definition ist die Marke
                     gleich einem nicht verschiebbaren Wert
                     plus einem verschiebbaren Wert. Da nur
                     ein Wert verschiebbar ist, ist auch
                     LABEL5 verschiebbar.

LABEL6:   .EQUAL    LABEL5-LABEL2
                    ;Gemaess Definition ist die Marke
                     gleich einem verschiebbaren Wert minus
                     einem anderen verschiebbaren Wert. Das
                     fuehrt zu einem nicht verschiebbaren
                     Ergebnis.
```

An das letzte Beispiel sollte man sich erinnern, wenn man mit Hilfe des Assemblers die Groesse von Datenbereichen berechnen moechte. Hierzu das nachfolgende Beispiel. Es enthaelt eine Tabelle von Datenwerten. Die Anzahl der Bytes wird waehrend der Assemblierung automatisch durch den Assembler berechnet. Das gestattet es dem Programmierer, die Tabelle zu erweitern oder sie zu verkuerzen, ohne sich um die Groesse des Datenblocks kuemmern zu muessen.

```
DATA:           .BYTE     0
                .WORD     10
                .BYTE     20
                .BLKB     5

DATA_SIZE:      .EQUAL    $-DATA
```

Der Assembler wird die Groesse des Datenblocks berechnen, und da das Ergebnis nicht verschiebbar ist, wird der Linker die Datenblockgroesse nicht veraendern.

2.4. Linker Beispiele

Dieser Abschnitt besteht aus Beispielen, an denen die Verwendung des Linkers demonstriert werden soll. Die Eingabeanforderungen des Linkers werden in Grossbuchstaben wiedergegeben, die Eingaben des Nutzers in Kleinbuchstaben. Das Symbol <cr> kennzeichnet einen Wagenruecklauf und wird nur angegeben, wenn keine andere Antwort auf eine Eingabeanforderung gewuenscht wird. Andernfalls wird angenommen, dass jede Eingabe mit einem Wagenruecklauf abgeschlossen wird.

2.4.1. Einzelne Datei, ab der gewuenschten Startadresse assembliert

Im ersten Beispiel wird nur eine einzige Datei behandelt, die unter Verwendung einer ORG-Steueranweisung fehlerfrei ab der gewuenschten Adresse assembliert wurde. Da nur die Adresse fuer den Kodebereich eingegeben wird, werden die Datenabschnitte, sofern die Datei welche enthaelt, unmittelbar an die Kodeabschnitte anschliessend, dicht gepackt.

```
INPUT   FILENAME:  filename    LOAD ADDRESS (OFFSET): 0
INPUT   FILENAME:  <cr>

OUTPUT  FILENAME:  <cr>
```

Die obigen Eingaben werden den Linker veranlassen, eine Datei mit dem Namen 'filename.obj' zu lesen, zu allen verschiebbaren Adressen des Kodebereiches eine 0 zu addieren, zu allen verschiebbaren Adressen aller Datenabschnitte die Gesamtlaenge aller Kodeabschnitte zu addieren und eine Datei auszugeben, die denselben Namen wie die Eingabedatei traegt, jedoch verlaengert um eine Standardnamenserweiterung, wie sie im Abschnitt 'Beschreibung des Linkers' (s. Abschn. 2.1.) angegeben wurde.

Gibt man nur die Ladeadresse fuer den Datenbereich an und laesst die Ladeadresse fuer den Kodebereich weg, so werden alle Kodeabschnitte dicht an das Ende der Datenabschnitte gelegt. Das wuerde man folgendermassen angeben.

```
INPUT   FILENAME:  filename    LOAD ADDRESS (OFFSET):  ,0
INPUT   FILENAME:  <cr>

OUTPUT  FILENAME:  <cr>
```

Wenn sowohl der Kode- als auch der Datenbereich durch Verwendung der ORG-Anweisung ab der gewuenschten Adresse assembliert wurden, dann muss fuer beide Bereiche ein Versatz von Null angegeben werden, wie es nachfolgend gezeigt wird.

```
INPUT   FILENAME: filename    LOAD ADDRESS (OFFSET): 0,0
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

2.4.2. Einzelne Datei, ab der Adresse 0000 assembliert

Das nachfolgende Beispiel unterscheidet sich von dem vorhergehenden nur darin, dass das Programm entweder keine ORG-Steueranweisung enthielt oder dass der Wert des Operanden der ORG-Anweisung Null betrug.

```
INPUT   FILENAME:  filename    LOAD ADDRESS (OFFSET): 100
INPUT   FILENAME:  <cr>

OUTPUT  FILENAME:  <cr>
```

Das veranlasst den Linker, eine Datei einzulesen, die den Namen 'filename.obj' traegt. Der Linker verschiebt alle verschiebbaren Werte des Kodebereiches um 100H, indem er zu jedem vom Assembler erzeugten Adressenversatz 100H addiert. Dann verschiebt er alle verschiebbaren Werte des Datenbereiches, indem er zu diesen Werten 100H plus die Gesamtlaenge des Kodebereiches addiert. Das Ergebnis ist, dass die Datenabschnitte dicht an das Ende der Kodeabschnitte gelegt werden. Es wird eine Datei ausgegeben, die den Namen der Eingabedatei und eine vom verwendeten Linker abhaengige Namenserweiterung erhaelt. Die in Frage kommenden Namenserweiterungen sind im Abschnitt 2.1. beschrieben.

Das folgende Beispiel unterscheidet sich von dem obigen nur darin, dass jetzt die Datenabschnitte zuerst verschoben werden und die Kodeabschnitte dicht an das Ende der Datenabschnitte angeschlossen werden.

```
INPUT   FILENAME: filename  LOAD ADDRESS (OFFSET):  ,100
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

In dem nachfolgenden Beispiel werden sowohl fuer den Kode- als auch fuer den Datenbereich Adressen angegeben, zu denen sie verschoben werden sollen.

```
INPUT   FILENAME: filename  LOAD ADDRESS (OFFSET): 100,200
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

Man beachte, dass, falls der Kodebereich ueber die Startadresse des Datenbereiches hinausreicht, 'link3' diese Situation nicht meistern kann. Es ist unmoeglich, eine ausfuehrbare Objektdatei zu erzeugen, da sich sowohl Kode als auch Daten im gleichen Adressbereich befinden. Die daraus entstehende Datei wird mit hoechster Wahrscheinlichkeit groesser als 64 K sein, da 'link3' versucht, die Luecken aufzufuellen, wobei eine Fehlermeldung ausgegeben wird, die den Nutzer auf diese Situation hinweist.

2.4.3. Zwei Dateien, ab der Startadresse assembliert, dicht gepackt

Dieses Beispiel besteht aus zwei Dateien, die fehlerfrei ab der gewuenschten Startadresse assembliert wurden. Jede der Dateien enthaelt zur anderen Adressbezuege. Die erste Datei enthaelt den Eintrittspunkt fuer das Programm. Man moechte die beiden Dateien so miteinander verbinden, als waeren sie als eine Datei assembliert worden. Zuerst sollen die Kodeabschnitte von beiden verschoben und dicht aneinander gelegt werden, dann sollen die Datenabschnitte verschoben und, unmittelbar an alle Kodeabschnitte anschliessend, ebenfalls dicht gepackt werden.

```
INPUT   FILENAME: filename1  LOAD ADDRESS (OFFSET): 0
INPUT   FILENAME: filename2  LOAD ADDRESS (OFFSET): <cr>
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

In diesem Fall liest der Linker eine Datei mit Namen 'filename1.obj', berechnet die Groesse des Objektkodes und addiert den Wert 0 zum vorgegebenen ORG-Wert. Dann liest der Linker die Datei mit Namen 'filename2.obj' und addiert den Wert 0 + den vorgegebenen ORG-Wert aus 'filename1.obj' + die Groesse des Objektkodes von 'filename1.obj'. Man beachte, dass das nur dann dicht gepackten Kode erzeugt, wenn 'filename2' nicht selbst eine ORG-Steueranweisung enthaelt, oder nur eine mit dem Wert 0. Die Ausgabe des Linkers ist eine Datei mit demselben Namen wie die erste Eingabedatei, aber mit der Namenserweiterung, die dem verwendeten Linker entspricht. Im Abschnitt 2.1. werden die Namenserweiterungen der verschiedenen Linker beschrieben.

Das folgende Beispiel ist dasselbe wie das obige, nur dass jetzt die Datenabschnitte zuerst dicht gepackt werden sollen. Dann sollen alle Kodeabschnitte , unmittelbar anschliessend an saemtliche Datenabschnitte, dicht gepackt werden.

```
INPUT   FILENAME: filename1  LOAD ADDRESS (OFFSET): ,0
INPUT   FILENAME: filename2  LOAD ADDRESS (OFFSET): <cr>
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

Im folgenden Beispiel werden sowohl die Kode- als auch die Datenstartadresse angegeben. Kode wird dicht an Kode gepackt, und Daten werden dicht an Daten gepackt.

```
INPUT   FILENAME: filename1  LOAD ADDRESS (OFFSET):100,500
INPUT   FILENAME: filename2  LOAD ADDRESS (OFFSET):<cr>
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

2.4.4. Zwei Dateien, ab der Adresse 0000 assembliert, dicht gepackt

Dieses Beispiel besteht aus zwei Dateien, die zu einer Datei verbunden werden sollen. Es kann gegenseitige Adressbezuege geben, es muss aber nicht der Fall sein. Beide Dateien wurden ab der Startadresse 0 assembliert und muessen deshalb zur gewuenschten Startadresse verschoben werden.

```
INPUT   FILENAME: filename1  LOAD ADDRESS (OFFSET): 100
INPUT   FILENAME: filename2  LOAD ADDRESS (OFFSET): <cr>
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

Zuerst liest und verschiebt der Linker den Objektkode von der Datei 'filename1.obj', dann liest er den Objektkode von der zweiten Datei 'filename2.obj' und verschiebt ihn so, dass er nach dem letzten Byte des Objektkodes von 'filename1.obj' beginnt. Irgendwelche Adressbezuege zur anderen Datei werden waehrend dieses Prozesses aufgeloest. Das Ergebnis ist dann eine Datei, bei der alle Adressen so verschoben wurden, dass sie ab 100 Hex abgearbeitet werden kann. Dabei sind als erstes alle Kodeabschnitte abgelegt, denen dann Datenabschnitte folgen koennen, sofern welche vorhanden sind.

2.4.5. Einzelne Datei mit mehrfachen ORG-Steueranweisungen

In diesem Beispiel wird der Fall betrachtet, wo eine Datei mehr als eine ORG-Steueranweisung enthaelt. Die Ladeadresse wird jeweils addiert, unabhaengig von dem in der Quelldatei festgelegten Wert.

```
INPUT   FILENAME: filename   LOAD ADDRESS (OFFSET): 100
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

Der Linker liest eine Datei mit Namen 'filename.obj' und beginnt, zu den verschieblichen Adressen 100 Hex zu addieren. Stoesst er dabei auf eine ORG-Steueranweisung, haengt der nachfolgende Ablauf vom verwendeten Linker ab. Wurde 'link3' verwendet, prueft er, ob durch die ORG-Anweisung eine Luecke entstanden ist. Ist das der Fall, wird er sie mit zu Null gesetzten Bytes auffuellen, da das notwendig ist, um eine ausfuehrbare Objektdatei zu erhalten. Deshalb kann, falls die ORG-Anweisungen in abfallender Reihenfolge auftreten, 'link3' keine richtig ausfuehrbare Datei erzeugen, weil er ueber FFFF Hex hinaus bis zu 0000 Hex hin die Luecken fuellen wird. Das wuerde hoechstwahrscheinlich eine Datei erzeugen, die groesser als 64 K waere und der Linker wuerde eine Fehlermeldung ausgeben, die den Nutzer auf diesen Umstand hinweist. Linkerversionen, die die durch ORG-Anweisungen bedingten Luecken nicht auffuellen, wie z.B. 'link5', der eine Datei im 'I-Hex' Format

ausgibt, koennen fuer ORG-Anweisungen in abfallender Reihenfolge verwendet werden.

2.4.6. Zwei Dateien, eine nur wegen der globalen Werte benutzt

Dieses Beispiel illustriert das Konzept der Reference-Only-Datei ('Nur-Verweisdatei'). Die erste Datei soll Objektkode enthalten, der ab 0 assembliert wurde, und ausserdem externe Adressbezuege, beispielsweise zu einer Sprungtabelle fuer Systemroutinen. Die Sprungtabelle existiert schon und soll nicht zum Bestandteil des Programms gemacht werden. Es werden einfach nur die Sprungadressen benoetigt. Es wird angenommen, dass sich die Sprungtabelle auf der Adresse A000 Hex befindet.

```
INPUT   FILENAME: filename1  LOAD ADDRESS (OFFSET): 100
INPUT   FILENAME: filename2  LOAD ADDRESS (OFFSET): -A000
INPUT   FILENAME: <cr>

OUTPUT  FILENAME: <cr>
```

Der Linker liest die Datei mit dem Namen 'filename1.obj' und realisiert alle externen Adressbezuege zur Datei 'filename2.obj'. Bevor die Adressen von 'filename2.obj' verwendet werden, werden zu ihnen A000 Hex addiert, sofern diese Datei ab 0 assembliert wurde. Wurde sie stattdessen ab A000 Hex assembliert, so erhaelt man das gewuenschte Ergebniss, wenn man fuer den OFFSET von 'filename2.obj' -0 eingibt. Die Ausgabedatei besteht aus dem Objektkode von 'filename1.obj', der alle aufgeloesten Adressbezuege zu 'filename2.obj' enthaelt, jedoch nicht den Objektkode von 'filename2.obj' selbst. Man beachte, dass die erste Datei den Programmeintrittspunkt (es wird vorausgesetzt, dass er gleich dem Operanden der ersten ORG-Anweisung ist, falls kein Eintrittspunkt hinter der END-Steueranweisung angegeben wurde, oder dass er sich auf der verschieblichen 0 befindet, falls keine ORG-Steueranweisung vorhanden ist) enthalten muss. Deshalb wurde die Reference-Only-Datei nach der Datei mit dem Hauptobjektkode angegeben.

Im obigen Beispiel gelten fuer die Datenabschnitte dieselben Attribute wie fuer die Kodeabschnitte, d.h., wenn der Kodebereich 'Reference-Only' ist, so ist es auch der Datenbereich. Sowohl Kode- als auch Datenabschnitte koennen aber auch unabhaengig voneinander angegeben werden.So kann der Kode 'Reference-Only' sein, ohne dass es auch die Daten sind. Oder die Daten koennen 'Reference-Only' sein, ohne dass es auch der Kode ist.

2.4.7. Drei Dateien, zwei davon dicht gepackt und nur wegen der globalen Werte benutzt

Dieses Beispiel ist aehnlich dem obigen, nur wird jetzt eine weitere Reference-Only-Datei verwendet, die dicht an das Ende der ersten Reference-Only-Datei gelegt wird.

```
 INPUT   FILENAME: filename1  LOAD ADDRESS (OFFSET): 100
 INPUT   FILENAME: filename2  LOAD ADDRESS (OFFSET): -A000
 INPUT   FILENAME: filename3  LOAD ADDRESS (OFFSET): -<cr>
 INPUT   FILENAME: <cr>

 OUTPUT  FILENAME: <cr>
```

Dieses Beispiel illustriert die Faehigkeit des Linkers, Reference-Only-Dateien dicht zu packen. Der Linker liest die Datei 'filename1.obj', verschiebt sie, indem er zu allen verschiebbaren Werten 100 Hex addiert, und loest dann alle externen Adressverbindungen zu 'filename2.obj' und 'filename3.obj' auf. Bevor irgendwelche externen Adressen verwendet werden, wird die Objektkodegroesse von 'filename2.obj' berechnet. Dann werden alle verschiebbaren Adressen aus 'filename3.obj' so verschoben, dass 'filename3.obj' unmittelbar an 'filename2.obj' anschliesst. Das Ergebnis ist eine Datei, die den Namen der ersten Eingabedatei traegt, mit der durch den verwendeten Linker festgelegten Namenserweiterung und in der alle externen Adressverbindungen zu 'filename2.obj' und 'filename3.obj' hergestellt wurden. Die Ausgabedatei wird jedoch nur aus der Datei 'filename1.obj' bestehen. Es koennen beliebig viele Reference-Only-Dateien in den Linkprozess eingeschlossen werden (solange die fuer den Linker zulaessige Anzahl von Dateien nicht ueberschritten wird). Dabei koennen sie dicht gepackt oder auch mit separaten Ladeadressen versehen werden. Dasselbe gilt fuer die Dateien, die in der Ausgabedatei enthalten sein sollen. Auch sie koennen dicht gepackt oder mit separaten Ladeadressen versehen werden. In beiden Faellen kann die Angabe auch gemischt erfolgen.

Im obigen Beispiel galten fuer die Datenabschnitte dieselben Attribute wie fuer die Kodeabschnitte, d.h., wenn der Kodebereich 'Reference-Only' war, so war es auch der Datenbereich. Sowohl Kode- als auch Datenbereich koennen unabhaengig voneinander angegeben werden, so dass der Kode 'Reference Only' sein kann, ohne dass es auch die Daten sein muessen. Oder die Daten koennen 'Reference Only' sein, ohne dass es auch der Kode sein muss.

3. Quellenkode-Translator

3.1. Beschreibung

Der Quellenkode-Translator konvertiert Assemblerprogramme, die entweder in K580IK80- oder U880-Syntax geschrieben sind, in K1810WM86-Quellenkode. Das auf diese Weise erzeugte Quellenkodeprogramm kann mit dem K1810WM86 Cross Assembler assembliert werden.

3.2. Registersubstitution

Der Ersatz der Register wird folgendermassen vorgenommen:

K580IK80/U880-Register	-	K1810WM86-Register
A	-	AL
B	-	CH
C	-	CL
D	-	DH
E	-	DL
H	-	BH
L	-	BL
BC	-	CX
DE	-	DX
HL	-	BX

Zusaetzlich zu den oben angegebenen Registern werden DI und SI von einigen Befehlen als temporaere Register verwendet. Die in den temporaeren Registern gespeicherten Werte muessen nur fuer die Dauer der Uebersetzung des aktuellen Befehls gueltig sein.

Die U880-Hilfsregister (Strichregister) und das IX- und IY-Register werden in Speicherbereiche eingetragen, die der Nutzer definiert.

3.3. Translater Bedienungsanweisungen

Um den Translater aufzurufen, wird eingegeben: trans

Zuerst fragt der Translater nach dem Typ des Quellenkodeprogramms:

```
SOURCE CODE MNEMONIC TYPE (I=IK80, Z=U880) ? :
```

Danach fragt der Translater, wohin das "Arbeitspapier" ("Worksheet") ausgegeben werden soll. Das "Arbeitspapier" ist eine zeilenweise Uebersetzungsliste. Die oberste Zeile enthaelt den Originalbefehl. Ihm folgt der entsprechende K1810WM86-Befehl, der laenger als eine Zeile sein kann. Die Aufforderung zur Eingabe sieht folgendermassen aus:

```
WORKSHEET DESTINATION (T=TERMINAL, P=PRINTER, D=DISK) ? :
```

Wird das "Arbeitspapier" auf den Disk-Speicher ausgegeben, so wird es auf demselben Laufwerk abgelegt, auf dem sich die Eingabedatei befindet. Es erhaelt denselben Namen wie die Ausgabedatei, aber mit der Namenserweiterung '.WS'. Man beachte, dass das "Arbeitspapier" wenigstens zweimal so lang wie die Eingabedatei sein kann.

Werden U880-Mnemoniks bearbeitet, gibt der Translator die folgenden vier Eingabeanforderungen aus:

```
NAME OF STORAGE SPACE FOR " EX  AF,AF' " INSTRUCTION ? :

NAME OF STORAGE SPACE FOR " EXX " INSTRUCTION ? :

NAME OF STORAGE SPACE FOR " IX " REGISTER ? :

NAME OF STORAGE SPACE FOR " IY " REGISTER ? :
```

Diese Namen repraesentieren Speicherplaetze, die in das Programm, nachdem es konvertiert wurde, eingefuegt werden muessen. Jeder Name kann bis zu 10 Zeichen lang sein und wird im Programm folgendermassen definiert:

```
AFSTORE:        .WORD           ;Speicher fuer AF'
EXXSTORE:       .BLKW    3      ;Speicher fuer BC', DE' & HL'
IXSTORE:        .WORD           ;Speicher fuer IX Register
IYSTORE:        .WORD           ;Speicher fuer IY Register
```

Als naechstes fragt der Translator nach dem Dateinamen des K580IK80/U880-Quellenkodes.

Abschliessend fragt der Translator nach dem Dateinamen fuer den K1810WM86-Quellenkode. Dieser Name muss nicht derselbe sein, wie der der Eingabedatei.

3.4. Vorbereitungen fuer die Konvertierung

3.4.1. Symbole

Ein Teil des Konvertierungsvorganges beinhaltet den Aufbau einer Symboltabelle. Symbole, die in anderen Programmen verwendet werden, koennen kenntlich gemacht werden, indem man sie als ".EXTERNAL" deklariert, bevor das Programm konvertiert wird. Dafuer gilt folgendes Format:

```
.EXTERNAL       SYMBOL1,SYMBOL2,SYMBOL3, ...
```

Die Symboltabelle wird auf dem Disk-Speicher unter dem hoffentlich einmaligen Namen TRANS.TBL abgelegt. Nach der Konvertierung wird diese Datei geloescht. Sollte aus irgendeinem Grund der Konvertierungsprozess abgebrochen werden, muss diese Datei vom Bediener geloescht werden, bevor ein neuer Konvertierungsversuch unternommen wird. Sind mehrere Laufwerke vorhanden, wird die Symboltabellendatei auf demselben Laufwerk gespeichert, wie die Quellenkodedatei.

3.4.2. Kommentare

Der Translator erkennt zwei Typen von Kommentarzeilen. Der erste ist eine einzelne Zeile mit einem Semikolon in Spalte 1. Fuer lange Kommentare haben manche Assembler eine spezielle Assemblersteueranweisung. Der Translator erkennt einen Kommentarblock, wenn er folgendermassen gekennzeichnet ist:

```
.COMMENT X
```

wobei 'X' jedes beliebige Zeichen sein kann. Der Translator wird alles, was sich zwischen dem ersten und dem zweiten 'X' befindet, als Kommentarblock behandeln.

3.5. Handhabung der U880-Hilfsregister

Wie schon in der Bedienungsanweisung (s. Abschn. 3.3.) erwaehnt wurde, fragt der Translator nach den Namen der Speicherbereiche fuer die U880-Befehle "EXX" und "EX AF,AF'". In diesem Abschnitt wird das noch genauer erklaert.

Der U880 hat sechs 16-Bit-Register, nicht eingeschlossen den Hilfsregistersatz (Strichregister). Aus diesem Grund, und unter Beruecksichtigung der Tatsache, dass der Translator zwei temporaere Register fuer die Befehle, die keine direkte Entsprechung haben, benoetigt, sind nicht genuegend CPU-Register fuer den U880-Hilfsregistersatz vorhanden. Um dieses Problem zu loesen, erzeugt der Translator die notwendigen Befehle, um einen Register-Speicheraustausch durchzufuehren. Von einer Standardfestlegung der Marken fuer diese Speicherplaetze wird abgesehen, um nicht eine Mehrfachdefinition von Marken zu riskieren (, die Aenderungen in vielen Befehlen erforderlich machen koennte). Deshalb muss der Nutzer beliebige Markennamen eingeben, die bis zu 10 Zeichen lang sein koennen.

Unten wird ein Beispiel fuer den Quellenkode, den der Translator fuer die Austauschbefehle erzeugt, angegeben. In diesem Fall ist der Name des Speicherbereiches fuer das AF Registerpaar 'AFSTORE', und der Name des Speicherbereiches fuer BC, DE, und HL ist 'EXXSTORE'.

```
EX      AF,AF'          ;U880-Befehl

LAHF                    ;Rette Flags
XCHG    AX,AFSTORE      ;Tausche Registerpaar
SAHF                    ;Speichere die neuen Flags

EXX                     ;U880-Befehl

XCHG    CX,EXXSTORE     ;Tausche Registersatz
XCHG    DX,EXXSTORE+2
XCHG    BX,EXXSTORE+4
```

Der sich ergebende Quellenkode wird nicht korrekt assembliert werden, wenn diese Speicherplaetze nicht folgendermassen definiert sind:

```
AFSTORE        .BLKW     1    ;1 Word reservieren fuer AF'
EXXSTORE       .BLKW     3    ;3 Word reservieren fuer EXX
```

Fuer Adressbezuege werden die Namen dieser Speicherplaetze auf der letzten Seite des "Arbeitspapiers" ausgegeben.

3.6. Warnmeldungen des Translators

Nachfolgend wird eine Liste der Warnmeldungen, die der Translator ausgibt, angegeben. In Beispielen wird sowohl der Quellenkode, der die Warnmeldung erzeugt, als auch eine moegliche Loesung des Problems aufgezeigt.

Warnmeldung:

```
WARNING:  DESTINATION ADDRESS MAY HAVE CHANGED
          (Zieladresse kann sich geaendert haben)
```

Bedeutung:

Diese Warnung wird immer dann ausgegeben, wenn der Translator die Zieladresse einer Programmverzweigung nicht in der Symboltabelle finden kann. Das geschieht einfach, um den Nutzer zu warnen, dass, falls ein Sprung oder ein Unterprogrammaufruf zu einer absoluten Adresse ausgefuehrt wird, die Moeglichkeit besteht, dass die angegebene Adresse nicht mehr gueltig ist, weil sich die Laenge jedes Befehls wie auch die Hardwareumgebung veraendert haben.

Beispiel:

Nachfolgend wird Quellenkode angegeben, der diese Warn-meldung zur Folge haette.

```
          AND       A         ;Setze Flags
          JP        Z,1000H
```

Wird konvertiert in:

```
          AND       AL,AL     ;Setze Flags
          JZ        1000H
***  WARNING: DESTINATION ADDRESS MAY HAVE CHANGED  ***
```

Abhilfe:

Das einzige, was man tun kann, ist, dass man kontrolliert, ob der Speicherplatz 1000 Hex noch die gewuenschte Adresse ist. Man beachte, dass bei Verzweigungen zu einer Speicherstelle in einem anderen Programm die entsprechende Marke vom Translator erkannt wird, wenn sie mit der Translator-Steueranweisung ".EXTERNAL" gekennzeichnet wurde.

Warnmeldung:

WARNING: THIS INSTRUCTION HAS NO EXACT EQUIVALENT
(Fuer diese Anweisung gibt es keine genaue Entsprechung)

Bedeutung:

Diese Warnung wird ausgegeben, wenn der Translator auf einen Befehl stoesst, der fuer den K1810WM86 nicht eindeutig ist. Dazu gehoeren die Befehle zum Freigeben und Sperren des Interrupts, der Restartbefehl sowie die Befehle zum Festlegen des Interruptmodus.

Beispiel:

Nachfolgend ein paar Beispiele fuer Quellenkode, der diese Warnmeldung erzeugt und einige Vorschlaege fuer die Loesung des Problems.

```
        IM   0          ;Setze Interruptmodus 0
        EI              ;Freigabe der Interrupts
        RST  1          ;Ausfuehren der Restartroutine
```

Wird konvertiert in:

```
        IM   0          ;Setze Interruptmodus 0
*** WARNING: THIS INSTRUCTION HAS NO EXACT EQUIVALENT ***
        EI              ;Freigabe der Interrupts
*** WARNING  THIS INSTRUCTION HAS NO EXACT EQUIVALENT ***
        RST  1          ;Ausfuehren der Restartroutine
*** WARNING: THIS INSTRUCTION HAS NO EXACT EQUIVALENT ***
```

Abhilfe:

Der Anwender muss mit der Interrupt- und der Systemrufstruktur vertraut sein, den richtigen Interrupttyp freigeben oder sperren und den richtigen Systemruf anstelle des Restartbefehls einsetzen.

Warnmeldung:

WARNING: THIS INSTRUCTION IS ASSUMED TO BE A MACRO
(Dieser Befehl wird fuer ein Makro gehalten)

Bedeutung:

Diese Warnmeldung erscheint, wenn der Translator den Befehl oder die Steueranweisung in seinen Tabellen nicht finden kann. Das wird nicht geschehen, wenn der Eingabequellenkode nur unter Verwendung von K580IK80- oder U880-Mnemoniks geschrieben wurde. Es existieren jedoch auch Assembler fuer den K580IK80, die so modifiziert wurden, dass sie die zusaetzlichen U880-Befehle enthalten, und die sich nicht an diese Konvention halten. In dieser Situation kann ein Editor mit einem Such- und Austauschkommando sehr nuetzlich sein.

3.7. Hinweise fuer die Nutzung des Translators

In diesem Abschnitt werden nur einige Vorschlaege unterbreitet, wie man den Translator am besten nutzen kann, um die erwarteten Ergebnisse zu erzielen.

Wenn man die Konvertierung eines Programms vorbereitet, ist es oft sinnvoll, mehrere Durchlaeufe vorzusehen. Die erste Konvertierung kann Informationen ueber Steueranweisungen liefern, die entweder der Translator oder der K1810WM86- Assembler nicht erkennt. Auch kann man sich dabei vergewissern, dass alle externen Symbole mit der Translator-Steueranweisung ".EXTERNAL" deklariert wurden. Die Translator-Steueranweisung ".COMMENT" kann dazu verwendet werden, Teile des Kodes, die nicht konvertiert werden sollen, von der Konvertierung auszuschliessen. Das ermoeglicht es, Veraenderungen des K580IK80- oder U880-Quellenkodes vorzunehmen, ohne dass man sie bei jedem neuen Durchlauf durch den Translator wiederholen muss, d. h., man kann K1810WM86-Kode in ein K580IK80- oder U880-Programm einfuegen, und den Translator anweisen, ihn zu ignorieren.

Es ist auch lohnenswert, sich damit zu befassen, in welcher Weise der Translator die Symboltabelle verwendet, um festzustellen, welchen Typ von Ladebefehl er erzeugen muss. Bei 8-Bit-Ladebefehlen erhaelt er die erforderliche Information aus dem Adressierungsmodus und aus den Operanden des Befehls und kann eine genaue Konvertierung vornehmen.

Bei 16-Bit-Ladebefehlen ist die Situation komplizierter. Der Translator muss versuchen herauszufinden, ob der Operand eine Adresse oder nur ein unmittelbarer Datenwert ist. Ist das Ziel ein Register, durchsucht der Translator die Symboltabelle nach einer Eintragung, die mit dem Quellenoperanden uebereinstimmt. Wird keine mit dem Quellenoperanden uebereinstimmende Eintragung gefunden, wird angenommen, dass es sich um einen unmittelbaren Datenwert handelt und der Operand wird als 16-Bit-Zahl geladen. Wird eine Uebereinstimmung festgestellt, dann wird der Datentyp geprueft, ob er eine 'Zahl' oder eine 'Adresse' darstellt. Das wird mit dem nachfolgenden Schema festgestellt. Wird das Symbol im Markenfeld eines Befehls vorgefunden, dann wird es als eine Adresse betrachtet. Ist das Symbol unter Verwendung einer 'EQU'-Assemblersteueranweisung definiert, dann sucht der Translator im Operandenfeld nach weiteren Informationen. Ist das erste Zeichen des Operanden eine Zahl, dann wird das Symbol als 'Zahl' gekennzeichnet, andernfalls als 'Adresse'. Bei Einhaltung nachfolgender Regeln erhaelt man korrekte Ergebnisse:

- Hexadezimale Zahlen muessen mit einer Zahl zwischen 0 und 9 beginnen.

- Symbole muessen mit einem Buchstaben beginnen.

- Symbole, die einer Zahl gleichgesetzt wurden, duerfen nicht einem anderen Symbol gleichgesetzt werden.

3.8. Erlaeuterungen zum "Arbeitspapier"("Worksheet")

Der Translator gibt die K580IK80/U880-Quellenkodezeile aus, gefolgt von der (den) K1810WM86-Quellenkodezeile(n). Moechte man die Groesse der Ausgabedatei reduzieren, so kann man das "Arbeitspapier" dazu benutzen, genau festzustellen, was der Translator gemacht hat.

Am ehesten koennen Zeilen gestrichen werden, die hinzugefuegt wurden, um dieselbe Verwendung der Flags zu ermoeglichen. In vielen Faellen ist das Setzen der Flags nicht von Interesse.

Bei einigen Befehlen ist es erforderlich, zu ueberpruefen, ob sie auch das leisten, wofuer sie vorgesehen waren. Ein gutes Beispiel hierfuer sind die 16-Bit-INCREMENT- und -DECREMENT-Befehle. Beim K580IK80 und beim U880 setzen diese Befehle keinerlei Flags. Beim K1810WM86 setzen sie Flags. Jedoch werden auf diese Weise sehr oft Register inkrementiert oder dekrementiert, die nur als Zeiger verwendet werden, so dass die Veraenderung der Flags ohne Bedeutung ist. Deshalb fuegt der Translator auch keine Befehle ein, die diese Flagveraenderung beruecksichtigen. Das wird keine Probleme verursachen, solange die Flags nicht abgetestet werden, um das Ergebnis eines vorhergehenden Befehls zu erfahren. Um etwaige Probleme zu verhindern, sollte man den Kode in der nachfolgend beschriebenen Weise ueberpruefen:

Treten zwischen dem 16-Bit-'INC'- oder -'DEC'-Befehl und der ersten bedingten Verzweigung keine weiteren Befehle auf, die die Flags veraendern, so sollte der Zustand der Flags wiederhergestellt werden. Befindet sich jedoch zwischen einem 16-Bit-'INC'- oder -'DEC'-Befehl und der naechsten bedingten Verzweigung noch ein Befehl, der die Flags veraendert, so wird das Programm in der gewuenschten Weise funktionieren. Bei unbedingten Verzweigungen oder bei Returns sollte man entweder ueberpruefen, ob die obige Situation vorliegt, oder man sollte die Flags vor einem 'INC' oder 'DEC' retten und danach wiederherstellen. Es ist aber auch moeglich, den Befehl, der die auszuwertenden Flags setzt, zu verschieben.

Es ist auch ratsam, bei komplizierten Ausdruecken, die waehrend der Assemblierung aufgeloest werden sollen, zu ueberpruefen, ob die richtige Reihenfolge eingehalten wurde. Das gilt besonders, wenn die Indexregister einbezogen sind. In diesem Fall kann die Konvertierung ziemlich kompliziert sein.

Alle Fehler- und Warnmeldungen sind in der K1810WM86-Quellenkodedatei enthalten und muessen, nachdem das Problem korrigiert wurde, daraus entfernt werden, bevor das Programm richtig assembliert werden kann.

Abschliessend sei darauf hingewiesen, dass das Programm keine fatalen Fehler enthalten darf. Der Translator setzt voraus, dass er mit einem Quellenkode arbeitet,

der sich fehlerfrei assemblieren laesst. Entdeckt er Kode, den er nicht kennt, wie z.B. ein unzulaessiges Adressenformat, so erzeugt er einen 'fatal error'. Das geschieht mit grosser Sicherheit, wenn der Nutzer versucht, K580IK80-Quellenkode im U880-Modus zu konvertieren, oder umgekehrt, U880-Quellenkode im K580IK80-Modus. Auch macht es sich der Translator zunutze, dass Kode fehlerfrei uebersetzt werden kann, wenn jede Zeile zerlegt wird. Das kann zu unvorhersehbaren Ergebnissen fuehren, wenn Fehler vorhanden sind, besonders in jenen Faellen, wo ein Mnemonik sowohl in der K580IK80- als auch in der U880-Syntax zulaessig ist, aber die Operanden ein unterschiedliches Format haben.

4. Dienstprogramme

4.1. "I-Hex"-Format-Konvertierungsprogramm

Mit dem "I-Hex"-Format-Konvertierungsprogramm 'hex' kann eine ausfuehrbare Objektkodedatei in das I-Hex-Format konvertiert werden. Es ist zur Verwendung fuer Dateien, die mit 'link3' (fuer die Beschreibung von 'link3' siehe Abschn. 2.1.) erzeugt wurden, vorgesehen. Die durch 'hex' erzeugte Ausgabedatei kann viel groesser als das eigentliche Programm sein. Der Grund dafuer ist folgender. Die kleinste Groesse fuer die Eingabedatei ist ein Block von 128 Bytes (ein Record). Durch die Konvertierung wird die Datei doppelt so gross, also 256 Bytes. Da zusaetzliche Informationen, wie Startzeichen, Ladeadressen und Pruefsummen hinzugefuegt werden, ist die kleinste Ausgabedatei des Konvertierungsprogramms drei Records gross. Werden genaue Groessen gefordert, sollte 'link5' verwendet werden.

Um das Konvertierungsprogramm aufzurufen, wird eingegeben: hex

Das Programm antwortet mit der folgenden Eingabeanforderung:

```
ENTER INPUT FILENAME:
```

Die Eingabedatei muss eine Dateinamenserweiterung (z.B. 'tsk') besitzen.
Die Ausgabedatei hat denselben Namen wie die Eingabedatei, nur mit der Namenserweiterung 'hex'.

Nach der Eingabe des Eingabedateinamens fragt das Konvertierungsprogramm nach der Startadresse. Das ist die Adresse, ab der das Programm tatsaechlich laufen soll und die kodiert in die Ausgabedatei aufgenommen wird.

Nachfolgend wird das "I-Hex"-Format beschrieben:

Record-Markenfeld - Dieses Feld kennzeichnet den Anfang eines Records und besteht aus einem ASCII-Doppelpunkt (:).

Record-Laengenfeld - Dieses Feld besteht aus zwei ASCII-Zeichen, die die Anzahl der Bytes in diesem Record angeben. Die Zeichen sind das Ergebnis der Konvertierung der binaeren Anzahl der Datenbytes in zwei ASCII-Zeichen, wobei die hoeherwertige Zahl zuerst kommt. Das 'End-of-file'-Record enthaelt zwei ASCII-Nullen in diesem Feld. Die maximale Anzahl von Datenbytes in einem Record betraegt 255. Das Konvertierungsprogramm verwendet 32 Datenbytes pro Record.

Ladeadressenfeld - Dieses Feld besteht aus vier ASCII-Zeichen, die sich aus der Konvertierung des binaeren Wertes der Adresse, ab der dieses Record geladen werden soll, ergeben. Die Anordnung ist folgendermassen:

Hoeherwertige Ziffer des hoeherwertigen Bytes der Adresse.
Niederwertige Ziffer des hoeherwertigen Bytes der Adresse.
Hoeherwertige Ziffer des niederwertigen Bytes der Adresse.
Niederwertige Ziffer des niederwertigen Bytes der Adresse.

In einem 'End-of-file'-Record besteht dieses Feld entweder aus vier ASCII-Nullen oder der Eintrittsadresse fuer das Programm. Da das Konvertierungsprogrammm die Eintrittsadresse des Programms nicht kennt, wird dieses Feld in einem 'End-of-file'-Record immer mit Nullen gefuellt sein.

Record-Typfeld - Dieses Feld kennzeichnet den Recordtyp. Er ist entweder 00 fuer Datenrecords oder 01 fuer ein 'End-of-file'-Record. Er besteht aus zwei ASCII-Zeichen, mit der hoeherwertigen Ziffer des Recordtyps zuerst, gefolgt von der niederwertigen Ziffer des Recordtyps.

Datenfeld - Dieses Feld besteht aus den aktuellen Daten, konvertiert in zwei ASCII-Zeichen, die hoeherwertige Ziffer zuerst. In dem 'End-of-file'-Record gibt es keine Datenbytes.

Pruefsummenfeld - Das Pruefsummenfeld ist eine binaere 8-Bit-Summe, gebildet aus dem Record-Laengenfeld, dem Ladeadres-

senfeld, dem Record-Typfeld und dem Datenfeld. Diese Summe wird dann negiert (2-er Koplement) und in zwei ASCII-Zeichen konvertiert, die hoeherwertige Ziffer zuerst.

4.2. Klein- in Grossbuchstaben-Konvertierungsprogramm

Das Klein- in Grossbuchstaben-Konvertierungsprogramm 'cscc' konvertiert alle Kleinbuchstaben einer Quellendatei in Grossbuchstaben. Davon ausgenommen sind in Apostrophe eingeschlossene Zeichenketten. Enthaelt die Zeichenkette ein Apostroph, so wird es durch zwei nebeneinander stehende Apostrophe angegeben. Diese Regelung entspricht den Anforderungen der meisten Assembler.

Um das Konvertierungsprogramm aufzurufen, wird eingegeben: cscc
Das Programm antwortet mit der folgenden Eingabeanforderung:

```
ENTER SOURCE CODE INPUT FILENAME :
```

Nachdem der Nutzer den Namen der Eingabedatei eingegeben hat, fordert das Programm zur Eingabe des Namens der Ausgabedatei auf:

```
ENTER SOURCE CODE OUTPUT FILENAME:
```

Danach zeigt das Konvertierungsprogramm jede konvertierte Zeile auf dem Bildschirm an.

5. Beispiele

5.1. Schreiben von Programmen fuer DCP

Dieser Abschnitt ist dazu bestimmt, dem Nutzer zu helfen, sich mit dem Assembler und dem Linker vertraut zu machen. Die nachfolgenden Beispiele zeigen lediglich verschiedene Wege, um auf dem Bildschirm die Ausschrift "HELLO, WORLD" zu erzeugen.

5.1.1. Kode und Daten in demselben Segment

Dieses Beispiel betrachtet den Fall, wo sich Kode und Daten in demselben Segment befinden. Diese Anordnung ist sinnvoll bei kleinen Programmen sowie bei Programmen, die durch Konvertierung von K580IK80- oder U880-Kode entstanden sind.

```
Programm:
     MOV  BX,DS              ;Segmentnummer der Return-
;                             adresse retten
     MOV  AX,CS              ;Aktuelles Kode-Segment
;                             nehmen
```

```
     MOV  DS,AX               ;Alle zum gleichen Segment
;                              machen
     MOV  SS,AX
     MOV  ES,AX
     MOV  SP,OFFSET STACK     ;Stackpointer initialisieren

     PUSH BX                  ;Speichern der Segmentnummer
;                              der Returnadr.
     MOV  AX,00               ;Speichern des DCP-Offset
     PUSH AX                  ;im Stack
     MOV  DX,OFFSET MSG       ;Adresse der Nachricht laden
     MOV  AH,09H              ;'Print String'-Kommando la-
;                              den
     INT  21H                 ;Ausgabe der Zeichenkette

     RET  FAR                 ;Return zu DCP

MSG: .BYTE     "CR","LF",07H,'HELLO, WORLD',"CR","LF",'$'

STACK:
     .EQUAL    $+80H          ;80H fuer Stack vorsehen
     .END
```

Nachdem man dieses Programm assembliert hat, wird der Linker fuer DCP-Ausgabedateien ('link1') aufgerufen. Seine Eingabeanforderungen werden folgendermassen beantwortet:

```
INPUT  FILENAME ? hello
       LOAD ADDRESS (OFFSET) ?  0
INPUT  FILENAME ? <cr>

OUTPUT FILENAME ? <cr>
```

Es wird eine Datei erzeugt, die den Namen 'hello.exe' traegt und die unter DCP abgearbeitet werden kann.

5.1.2. Kode, Daten und Stack in verschiedenen Segmenten

In diesem Beispiel wird der Fall betrachtet, wo sich der Kode im Kode-Segment, die Daten im Daten-Segment und der Stack im Stacksegment befinden.

```
Programm:
     MOV  BX,DS               ;Segmentnummer der Return;
;                              adresse retten

     MOV  AX,OFFSET DATASEG   ;Adresse des Datensegments
;                              laden
     MOV  DS,AX               ;In das Segmentregister brin-
;                              gen
     MOV  AX,OFFSET STKSEG    ;Adresse des Stacksegments
;                              laden
     MOV  SS,AX               ;In das Segmentregister brin-
;                              gen
;                              Der Wert in SP wird vom DCP-
;                              Lader gleich dem Ende des
;                              Stackbereiches gesetzt
```

```
     PUSH BX                   ;Segmentnummer   der  Return-
;                               adresse speichern
     MOV  AX,00                ;Offset der DCP-Return
     PUSH AX                   ;adresse im Stack speichern
     MOV  DX,OFFSET MSG        ;Adresse der Nachricht laden
     MOV  AH,09H               ;'Print String'-Kommando  la-
;                               den
     INT  21H                  ;Ausgabe der Zeichenkette

     RET  FAR                  ;Return zu DCP

DATASEG:    .DATA              ;Beginn des Datensegments
MSG: .BYTE  "CR","LF",07H,'HELLO, WORLD',"CR","LF",'$'
STKSEG:   .STACK               ;Beginn des Stacksegments
          .BLKB     80H        ;80H fuer Stack vorsehen
          .END
```

Nachdem man dieses Programm assembliert hat, wird der Linker fuer DCP-Ausgabedateien ('link1') aufgerufen. Seine Eingabeanforderungen werden folgendermassen beantwortet:

```
INPUT  FILENAME ? hello
       LOAD ADDRESS (OFFSET) ?  0
INPUT  FILENAME ? <cr>

OUTPUT FILENAME ? <cr>
```

Es wird eine Datei erzeugt, die den Namen 'hello.exe' traegt und die unter DCP abgearbeitet werden kann.

5.1.3. Kode und Daten in verschiedenen Modulen

In diesem Beispiel wird der Fall betrachtet, wo sich der Kode und die Daten in verschiedenen Modulen befinden.

Programm in der Datei 'hello1.asm':

```
     .EXTERNAL  SEGMENT   DATASEG
     .EXTERNAL  MSG
     .EXTERNAL  STACK

     MOV  BX,DS                ;Segmentnummer  der   Return-
;                               adresse retten

     MOV  AX,OFFSET DATASEG    ;Adresse   des  Datensegments
;                               laden
     MOV  DS,AX                ;In das Segmentregister laden
     MOV  SS,AX                ;Das  Datensegment wird  auch
;                               fuer den Stack genutzt
     MOV  SP,OFFSET STACK      ;Laden der Stackadresse

     PUSH BX                   ;Segmentnummer  der   Return-
;                               adresse speichern
     MOV  AX,00                ;Offset der DCP-Return
     PUSH AX                   ;adresse im Stack speichern
     MOV  DX,OFFSET MSG        ;Adresse der Nachricht laden
```

```
     MOV  AH,09H              ;'Print  String'-Kommando la-
 ;                             den
     INT  21H                 ;Ausgabe der Zeichenkette

     RET  FAR                 ;Return zu DCP

     .END
```

Programm in der Datei 'hello2.asm':

```
     .GLOBAL   DATASEG
     .GLOBAL   MSG
     .GLOBAL   STACK

DATASEG:  .DATA               ;Beginn des Datensegments

MSG:      .BYTE  "CR","LF",07H,'HELLO, WORLD',"CR","LF",'$'

          .BLKB  80H          ;80H Bytes fuer Stack vorsehen
STACK:    .WORD

          .END
```

Nachdem man dieses Programm assembliert hat, wird der Linker fuer DCP-Ausgabedateien ('link1') aufgerufen. Seine Eingabeanforderungen werden folgendermassen beantwortet:

```
INPUT  FILENAME ? hello1
       LOAD ADDRESS (OFFSET) ?  0
INPUT  FILENAME ? hello2
       LOAD ADDRESS (OFFSET) ?  <cr>
INPUT  FILENAME ? <cr>

OUTPUT FILENAME ? <cr>
```

Es wird eine Datei erzeugt, die den Namen 'hello1.exe' traegt und die unter DCP abgearbeitet werden kann.

5.2. Schreiben von Programmen fuer SCP86

Dieser Abschnitt soll dem Nutzer helfen, sich mit dem Assembler und dem Linker vertraut zu machen. Die nachfolgenden Beispiele zeigen lediglich verschiedenen Wege, um auf dem Bildschirm die Ausschrift "HELLO, WORLD" zu erzeugen.

5.2.1. Kode und Daten in demselben Segment

In diesem Beispiel wird der Fall betrachtet, wo sich Kode und Daten in demselben Segment befinden. Diese Anordnung ist sinnvoll bei kleinen Programmen, sowie bei Programmen, die durch Konvertierung von K580IK80- oder U880-Kode entstanden sind.

```
Programm:
     .BLKB     100H            ;Bereich  auf Seite 0  reser-
                               ;vieren
     MOV       DX,OFFSET MSG   ;Adresse der Nachricht laden
     MOV       CL,09H          ;'Print  String'-Kommando la-
                               ;den
     INT       224             ;Ausgabe der Zeichenkette
     MOV       CL,00           ;Return zum SCP86-Kommando #
     MOV       DL,00           ;Verbleibt nicht im Speicher
     INT       224
MSG: .BYTE     "CR","LF",07H,'HELLO, WORLD',"CR","LF",'$'

STACK:
     .EQUAL    $+80H           ;80H Bytes fuer Stack  vorse-
                               ;hen
     .END
```

Nachdem man dieses Programm assembliert hat, wird der Linker fuer SCP86-Ausgabedateien ('link2') aufgerufen. Seine Eingabeanforderungen werden folgendermassen beantwortet:

```
INPUT  FILENAME ? hello
       LOAD ADDRESS (OFFSET) ?  0
INPUT  FILENAME ? <cr>

OUTPUT FILENAME ? <cr>
```

Es wird eine Datei erzeugt, die den Namen 'hello.cmd' traegt und die unter SCP86 abgearbeitet werden kann.

5.2.2. Kode, Daten und Stack in verschiedenen Segmenten

In diesem Beispiel wird der Fall betrachtet, wo sich der Kode im Kodesegment, die Daten im Datensegment und der Stack im Stacksegment befinden.

```
Programm:
     MOV  SP,OFFSET STACK      ;Neuen Stackpointer setzen
     MOV  DX,OFFSET MSG        ;Adresse der Nachricht laden
     MOV  CL,09H               ;'Print      String'-Kommando
;                                laden
     INT  224                  ;Ausgabe der Zeichenkette
     MOV  CL,00                ;Return zum SCP86-Kommando
     MOV  DL,00                ;Verbleibt nicht im Speicher
     INT  224

     .DATA                     ;Beginn des Datensegments
     .BLKB     100H            ;Bereich  auf Seite 0  reser-
;                               vie ren
MSG: .BYTE     "CR","LF",07H,'HELLO, WORLD',"CR","LF",'$'
     .STACK                    ;Beginn des Stacksegments
     .BLKB     80H             ;80H  Bytes fuer Stack vorse-
;                               hen
STACK:
     .WORD
     .END
```

Nachdem man dieses Programm assembliert hat, wird der Linker fuer SCP86-Ausgabedateien ('link2') aufgerufen. Seine Eingabeanforderungen werden folgendermassen beantwortet:

```
INPUT  FILENAME ? hello
       LOAD ADDRESS (OFFSET) ?  0
INPUT  FILENAME ? <cr>

OUTPUT FILENAME ? <cr>
```

Es wird eine Datei erzeugt, die den Namen 'hello.cmd' traegt und die unter SCP86 abgearbeitet werden kann.

5.2.3. Kode und Daten in verschiedenen Modulen

In diesem Beispiel wird der Fall betrachtet, wo sich der Kode und die Daten in verschiedenen Modulen befinden.

Programm in der Datei 'hello1.asm':

```
     .EXTERNAL      MSG
     .EXTERNAL      STACK

     MOV  SP,OFFSET STACK     ;Laden der Stackadresse
     MOV  DX,OFFSET  MSG      ;Laden der Adresse der  Nach-
;                              richt
     MOV  CL,09H              ;'Print  String'-Kommando la-
;                              den
     INT  224                 ;Ausgabe der Zeichenkette
     MOV  CL,00               ;Return zum SCP86-Kommando
     MOV  DL,00               ;Verbleibt nicht im Speicher
     INT  224

     .END
```

Programm in der Datei 'hello2.asm':

```
     .GLOBAL   MSG
     .GLOBAL   STACK

     .DATA                    ;Beginn des Datensegments
     .BLKB     100H           ;Bereich  auf Seite 0  reser-
                               vieren
MSG: .BYTE     "CR","LF",07H,'HELLO, WORLD","CR","LF",'$'
     .BLKB     80H            ;80H Bytes fuer Stack  vorse-
                               hen
STACK:
     .WORD
     .END
```

Nachdem man dieses Programm assembliert hat, wird der Linker fuer SCP86-Ausgabedateien ('link2') aufgerufen. Seine Eingabeanforderungen werden folgendermassen beantwortet:

```
INPUT  FILENAME ? hello1
       LOAD ADDRESS (OFFSET) ?  0
INPUT  FILENAME ? hello2
       LOAD ADDRESS (OFFSET) ?  <cr>
INPUT  FILENAME ? <cr>

OUTPUT FILENAME ? <cr>
```

Es wird eine Datei erzeugt, die den Namen 'hello1.cmd' traegt und die unter SCP86 abgearbeitet werden kann.

KOMBINAT VEB
ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW
»FRIEDRICH EBERT«

HEIM-ELECTRIC
EXPORT-IMPORT
Volkseigener Außenhandelsbetrieb
der Deutschen Demokratischen Republik

EAW-Automatisierungstechnik Export-Import

Storkower Straße 97
Berlin, DDR - 1055
Telefon 432010 · Telex 114158 heel dd

VEB ELEKTRO-APPARATE-WERKE BERLIN-TREPTOW
»FRIEDRICH EBERT«
Stammbetrieb des Kombinats EAW
DDR - 1193 Berlin, Hoffmannstraße 15-26
Fernruf: 2760
Fernschreiber: 0112263 eapparate bln
Drahtwort: eapparate bln