SIEMENS

Assembler-Handbuch
Personal Computer PC100

Vorläufige Ausgabe 1979/80

SIEMENS

Assembler-Handbuch
Personal Computer PC100

Vorläufige Ausgabe 1979/80

Vorwort

Um unsere Kunden unverzüglich mit detaillierten Informationen zu versorgen, haben wir uns entschlossen, eine "Vorläufige Ausgabe" zu produzieren, die wir Ihnen hiermit vorstellen. Die endgültige, redaktionell und technisch überarbeitete Ausgabe dieser Druckschrift wird derzeit erarbeitet. Sie steht auf Wunsch voraussichtlich ab 1.10.1979 zur Verfügung. Bitte benutzen Sie bei Bedarf die beiliegende Bestellkarte.

"Herausgegeben von Siemens AG, Bereich Bauelemente, Balanstraße 73, 8000 München 80.

Mit den Angaben im Datenbuch werden die Bauelemente spezifiziert, nicht Eigenschaften zugesichert. Liefermöglichkeiten und technische Änderungen vorbehalten.

Für die angegebenen Schaltungen, Beschreibungen und Tabellen wird keine Gewähr bezüglich der Freiheit von Rechten Dritter übernommen.

Fragen über Technik, Preise und Liefermöglichkeiten richten Sie bitte an unsere Zweigniederlassungen im Inland, Abteilung VB oder an unsere Landesgesellschaften im Ausland (siehe Geschäftsstellenverzeichnis)."

Inhaltsverzeichnis

1. Allgemeines

Der Vorgang des Übersetzens einzelner Computerprogrammbefehle, die in mnemonischer oder symbolischer Form geschrieben worden sind (das Quellprogramm), in tatsächliche Maschinenbefehle in Maschinencode Form (Maschinenprogramm), wird als Assemblierung bezeichnet. Das Computerprogramm, das diese Übersetzung durchführt, ist das ASSEMBLER-Programm oder ASSEMBLER. Die mnemonischen Symbole und die Symbole, die beim Schreiben des Quellprogrammes verwendet werden, üblicherweise Quellcode genannt, werden als ASSEMBLER-Sprache bezeichnet. Ein Befehl in ASSEMBLER-Sprache läßt sich in einen Befehl in der Maschinensprache übersetzen. Das Maschinenprogramm wird im allgemeinen als Maschinencode bezeichnet.

Der SIEMENS PC 100 ASSEMBLER ist ein ROM-residentes ASSEMBLER-Programm mit zwei Durchläufen. Es wird geliefert als ein 4K 2332 ROM, das in den Sockel Z24 auf der PC 100 Hauptplatine eingesteckt wird. Der ASSEMBLER erlaubt, daß sowohl Befehls- als auch Datenadressen symbolisch definiert werden. Als Gegensatz dazu stünde die Forderung nach absoluten Adressen. Während des ersten Durchganges bestimmt der ASSEMBLER die Werte der Symbole. Die Symbole und ihre dazugehörigen Werte werden in einer Symboltabelle abgelegt, für Verwendung während des zweiten Durchganges. Der ASSEMBLER erzeugt den tatsächlichen Maschinencode während des zweiten Durchganges, indem er die Symbolwerte aus der Symboltabelle verwendet, absolute und relative Adressen generiert um Datenwerte zu erzeugen. Umfassende Fehlerüberprüfung wird während des zweiten Durchlaufes durchgeführt, um festzustellen, ob die Befehle korrekt verschlüsselt wurden. Aufgefundene Fehler werden angezeigt/ausgedruckt. Die ASSEMBLER-Programmauflistung wird auch während des zweiten Durchganges erzeugt.

Um das ASSEMBLER-Programm zu verwenden, wird zuerst der Quellcode mittels Textaufbereitungsprogramm vorbereitet, dann wird dieser Quellcode auf Kassette aufgezeichnet, oder er wird direkt vom Textpuffer i RAM an das ASSEMBLER-Programm weitergegeben. Wird ein TTY verwendet, kann der Quellcode auf Lochstreifen gespeichert sein und kann als Eingabe an das ASSEMBLER-Programm zurückgelesen werden.

Die Operation des ASSEMBLER-Programmes ist innerhalb jeden Durchlaufes absolut automatisch, sobald Sie Informationen spezifiziert haben, wie z.B.: Herkunft des eingegebenen Quellcode, Ziel des auszugebenden Maschinencode, volle ASSEMBLER-Programmauflistung bzw. nur eine Fehlerauflistung.

Die ASSEMBLER-Befehle, die in dem Quellcode enthalten sind, liefern zusätzliche und Korrekturbefehle für die Erzeugung der Listen. Wird die Kassette als Eingabe verwendet, erlaubt eine Datei-Verbindungfähigkeit die Verwendung von Mehrfachdateien.

1.1 Einbau des Assembler-ROM

Bevor Sie das ROM aus seiner Verpackung nehmen, schalten Sie die Stromversorgung des PC 100 aus. Nehmen Sie das ROM aus seiner Verpackung. Überprüfen Sie die Stifte, um sicherzugehen, daß sie gerade sind und frei von Verpackungs- und Schutzschaumstoffmaterial. Stecken Sie das ROM in den Sockel Z24, achten Sie dabei auf die richtige Lage des Bausteines auf der Platine. Stützen Sie die Hauptplatine unter dem Sockel ab, drücken Sie fest und gleichmäßig auf das ROM, bis es ganz in den Sockel eingesteckt ist.

2. Symboltabelle

Für die Speicherung der Symboltabelle während des ersten Durchlaufes müssen im RAM Speicherstellen reserviert werden. Acht Bytes im RAM müssen für jedes einmalig definierte Symbol im Quellcode bereitgestellt werden; sechs Bytes für das Symbol selbst und zwei Bytes für den Symbolwert. Da jedes Symbol acht Bytes Speicherstellen erfordert, wird das Ende der Symboltabelle ein mehrfaches von acht Bytes ab der Startadresse sein. Sollte der zugeordnete Symboltabellenbereich nicht groß genug sein, wird der ASSEMBLER den ersten Durchlauf beenden, nachdem er folgende Fehlermeldung anzeigt:

```
SYM  TBL  OVERFLOW
```

Die Start- und Endadressengrenzen der Symboltabelle werden während des ersten ASSEMBLER-Durchlaufes eingegeben. Die tatsächliche Startadresse wird mit der eingegebenen Startadressengrenze übereinstimmen. Die tatsächliche Endadresse wird niedriger sein als die Endadressengrenze.

Die Start- und Oberadressengrenze der Symboltabelle und die Anzahl der Symbole in der Tabelle können festgestellt werden, indem man die Speicherstellen untersucht:

ADRESSE	PARAMETER	BEISPIEL
$003A	Symboltabelle Startadresse tief	$00
$003B	Symboltabelle Startadresse hoch	$03
$003E	Symboltabelle Adressenobergrenze tief	$FF
$003F	Symboltabelle Adressenobergrenze hoch	$03
$000B	Anzahl der Symbole (HEX) tief	$00
$000C	Anzahl der Symbole (HEX) hoch	$12

Die Adresse des letzten Symboles kann errechnet werden, indem man die Anzahl der Symbole in der Symboltabelle mit acht multipliziert und das Ergebnis zur Startadresse addiert.

Zum Beispiel:

$0300 + ($0012 X 8) = $0300 + $0090 = $0390
(Adresse des letzten Symboles)

ANMERKUNG

Wenn der Quellcode, der Maschinencode und die Symboltabelle alle während des Assemblierens im RAM Platz finden sollen, achten Sie darauf, daß sie ein Überschreiben des Quellcodes entweder mit der Symboltabelle oder dem Maschinencode oder ein Überschreiben der Symboltabelle mit dem Maschinencode verhindern. Achten Sie sehr darauf, nicht den Quellcode zu überschreiben, er muß sonst nochmal in den Textpuffer eingelesen werden. Es ist eine gute Angewohnheit, von Zeit zu Zeit den Quellcode auf Dauerspeichermedien (z.B. Kassette) zwischenzuspeichern, um unbeabsichtigten Verlust durch Überschreiben, Text - Editor - Initialisierung oder durch Ausschalten zu verhindern.

3. Assemblerprogramm-Erwägung

3.1 Von Speicher zu Speicher Assemblierung

Die tatsächlichen Maschinencodeadressen können überprüft werden, indem man die Assemblerprogrammauflistung während des zweiten Durchlaufes ausdruckt, ohne den Maschinencode in den Speicher zu geben. Der Maschinencode kann entweder zur Kassette oder zu dem Blindbaustein (d.h. keine Ausgabe) gerichtet werden. Wird die Ausgabe an die Kassette adressiert, kann diese dann mittels Monitorprogramm <L>Befehl in den Speicher geladen werden, ohne Rücksicht auf die Lage der vorher verwendeten Symboltabelle. Sollten die Maschinencodeadressen mit dem Quellcode kollidieren, sollte der Quellcode auf Kassette zwischengespeichert werden, bevor der Maschinencode geladen wird.

Wurde keine Maschinencodeausgabe generiert und zeigt die Überprüfung der Maschinencodeadressen auf der ASSEMBLER-Programmauflistung keine Quellcode oder Symboladresskollidierungen, kann der zweite Durchlauf wiederholt werden, um den Maschinencode an den Speicher zu adressieren und um keine ASSEMBLER-Programmauflistung zu generieren (da ja während des ersten Durchlaufes schon eine generiert wurde).

3.2 Band-zu Band- Assemblierung

Ein Programm mit vielen Symbolen könnte den größeren Teil oder den ganzen RAM für die Symboltabelle erfordern. In diesem Fall sollte der Quellcode vor der Assemblierung auf der Kassette zwischengespeichert werden. Durchlauf 1 und 2 sollten beide mit der Eingabe von der Kassette durchgeführt werden. Der ausgegebene Maschinencode sollte also an eine andere Kassette adressiert werden. (s. Audio-Kassettenoperation). Die Größe des Programmes wird jetzt nur beschränkt durch die RAM Speicher, die vorhanden sind, um die Anzahl der Symbole im Quellcode aufzunehmen.

3.3 Dauerhaftes Speichern von Anwenderprogrammen

Der Assembler verwendet die Adressen 0004 bis 00DE (ZERO PAGE) und 0170 bis 0183 (Seite Eins). Aus diesem Grund sollten Sie während der Assemblierung in den Speicher (OBJ?N) keine Befehle oder Konstanten in diese Adressen assemblieren. Variable des Anwenderprogrammes können jedoch diesen Adressen zugewiesen werden. Nachdem die Assemblierung durchgeführt ist, können Befehle/Konstanten in diesen Adressen geladen werden.

4. Anwendung des Assemblerprogramms

Verwenden Sie den PC 100 Assembler wie folgt:

1. Zum Aufruf des Assemblerprogrammes betätigen Sie N nach der Anzeige des Monitorprogramm-Zeichens <. PC 100 antwortet mit:

   ⟨N⟩
   ASSEMBLER
   FROM =^

2. Geben Sie die Symboltabellen-Startadresse hexadezimal ein. Beenden Sie die Adresse durch RETURN. Betätigen der RETURN Taste ohne die Eingabe eines Wertes bedingt die Verwendung des vorher eingegebenen Wertes als Startadresse. Entweder der neu eingegebene oder der vorherige Wert wird angezeigt.

   Zum Beispiel, wurde 0300 eingegeben, antwortet PC 100 mit:

   FROM=0300 TO=^

VORSICHT

Da die gesamte Seite 0 für Assemblerprogrammvariable und Seite 1 für Anwender- und PC 100 Monitorprogrammstapel und für Variable des PC 100 reserviert ist, muß die Startadresse der Symboltabelle gleich oder größer als 0200 sein.

3. Geben Sie die Symboltabellen-Endadresse hexadezimal ein. Beenden Sie die Adresseneingabe mit RETURN. Das Betätigen der RETURN Taste ohne die Eingabe eines Wertes bewirkt, daß als Endadresse der vorher eingetragene Wert eingegeben wird. Entweder der neu oder vorher eingegebene Wert werden angezeigt. Zum Beispiel, wurde 0400 eingegeben, wird PC 100 wie folgt antworten:

   FROM=0300 TO=0400

   IN=^

4. Geben Sie den Code des Eingabebausteines ein, der den Quellcode enthält. Gültige Möglichkeiten sind:

   M = Textpuffer im Speicher (RAM)
   T = Kassettenrecorder
   L = TTY-Lochstreifen
   U = vom Anwender definierte periphere Geräte

VORSICHT

Soll der erste Durchlauf vom Speicher durchgeführt werden (IN = M), stellen Sie sicher, daß die Symboltabelle nicht im Widerspruch zu den Adressen des Quellcode im Textpuffer steht. Der ganze Quellcode, oder Teile davon, werden in diesem Fall mit der Symboltabelle überschrieben.

BeachtenSie, daß der Quellcode in dem Augenblick angezeigt wird, in dem er von dem Eingabebaustein eingelesen wird.

A. Wird M eingegeben, wird PC 100 M anzeigen. Gehen Sie zu Schritt 5.

B. Wird T eingegeben, wird PC 100 nach dem Dateinamen fragen:

IN=T F=^

ANMERKUNGEN

- Zur Verwendung der Kassette als Eingabe muß der Kassetten-Recorder eine Fernbedienung besitzen, mit Anschlüssen wie in der Bedienungsanleitung beschrieben. Der Assembler verarbeitet den Quellcode, indem er einen Block von 80 Bytes auf einmal verarbeitet. Nachdem ein Block Quellcode eingelesen wurde, wird während dieser Zeit der Kassetten-Recorder mittels Fernbedienung angehalten, sodaß das Band zwischen zwei Blöcken von Eingabedaten zum Stehen kommt. Sobald der Assembler den Block von 80 Bytes verarbeitet hat, wird der Recorder neu gestartet, um einen neuen Datenblock einzulesen.

- Der Quellcode muß mit dem Wert $80 (oder größer) in $A4Ø9 aufgenommen worden sein.

Geben Sie den Dateinamen, unter dem der Quellcode gespeichert wurde, ein. Hat der Dateiname eine Länge von weniger als fünf Ziffern, beenden Sie die Eingabe mit einem RETURN oder Leertaste. PC 100 fragt

dann nach der Kassettenrecordernummer. Zum Beispiel:

```
IN=T    F=SRCE1    T=^
```

Geben Sie die Nummer des Kassetttenrecorders (1 oder 2) ein, von dem der Quellcode geladen werden soll. Beenden Sie die Eingabe mit einem RETURN oder der Leertaste. Wurde 1 eingegeben, wird PC 100 wie folgt antworten:

```
IN=T    F=SRCE1    T=1
```

PC 100 sucht nun nach dem definierten Dateinamen. Findet er eine lesbare Banddatei, wird der Dateiname auf dem Band mit dem eingegebenen Dateinamen verglichen. Sollten die Dateinamen nicht identisch sein, wird PC 100 die Suchmeldung und die Blockanzahl der aufgezeichneten Datei, wie sie abgefragt werden, anzeigen. Wird der Dateiname PROG1 gelesen, antwortet PC 100 wie folgt:

```
SRCH    F=PROG1    BLK=XX    Wobei XX= der Blockzählstand.
```

Sobald der eingegebene Dateiname auf dem Band gefunden wurde, geht das Assemblerprogramm weiter mit Schritt 5.

C. Wurde L eingegeben, sollte das Einlesen des Quellcode auf Lochstreifen vom TTY, wie in der Bedienungsanleitung beschrieben, eingeleitet werden.

D. Wurde U eingegeben, wird der erste Durchlauf unter Verwendung der anwenderdefinierte Eingabe eingeleitet (siehe Bedienungsanleitung).

5. PC 100 wird nachfragen, ob die gesamte Assemblerprogramm-Auflistung oder nur eine Fehlerliste angezeigt/ausgedruckt werden soll:

LIST?^

Sollen nur die Fehler aufgelistet werden, betätigen Sie N. Eine Nur-Fehlerauflistung wird während des zweiten Durchlaufes generiert.

Soll die volle Assemblerprogramm-Auflistung produziert werden, betätigen Sie Y. Die gesamte Assemblerprogramm-Auflistung beinhaltet das gesamte Quellprogramm, neu formatiert, um die richtigen Ausgabeabstände zu bekommen, die Adresse mit jeder Identifiziermarke (Label) de generierte Maschinencode und alle gefundenen Fehler.

6. PC 100 wird nachfragen, wohin die volle Assemblerprogramm- oder Fehlerauflistung gelenkt werden soll.

LIST-OUT=^

Schreiben Sie eine der folgenden gültigen Möglichkeiten:

RETURN oderLeert = Anzeige/Drucker
P = Drucker
T = Kassette (PC 100 Quellcode Format)
U = vom Anwender definiert
L = TTY

7. PC 100 wird nun fragen, wohin der Maschinencode geleitet werden soll:

   OBJ?↵

   Soll der Maschinencode direkt in den Speicher geleitet werden, betätigen Sie N. Gehen Sie weiter zu Schritt 9.

   VORSICHT

   Soll die Ausgabe in den Speicher geleitet werden, stellen Sie sicher, daß die Adressen des Maschinencode bei Eingabe vom Speicher nicht im Widerspruch stehen zum Quellcode im Textpuffer oder zu den Symboltabellenadressen.

   Soll der Maschinencode an einen Ausgabebaustein geleitet werden und nicht an den Speicher, betätigen Sie Y. PC 100 wird mit der Frage nach dem Ausgabebaustein antworten:

   OBJ-OUT=↵

8. Betätigen Sie eine der folgenden gültigen Codemöglichkeiten:

   RETURN oder SPACE = Anzeige/Drucker
   P = Nur Drucker
   T = Kassette (PC 100 Format)
   L = TTY
   X = Blindbaustein (keine Ausgabe)
   U = vom Anwender definiert

ANMERKUNG

Die ausgewählte OBJ-OUT= Option darf nicht mit der vorher gewählten LIST-OUT Option kollidieren, sonst werden beides, Auslistung und Maschinencodeausgabe, an den gleichen Ausgabebaustein in einer gemischten Form geleitet.

Wenn die folgende LIST-Option gewählt wird	sind folgende mit * gekennzeichneten OBJ-Ausgabe-Optionen erlaubt					
	OBJ?N	OBJ?Y OBJ - OUT =				
	(M)	X	T	K	U	RETURN oder SPACE
RETURN oder SPACE	*	*	*	*	*	
P	*	*	*	*	*	
T	*	*			*	*
U	●	●	●	●		●

● hängt ab von der gewählten Eingabe.

9. PC 100 wird den ersten Durchlauf einleiten und anzeigen:

PASS 1

Während des ersten Durchganges generiert das Assemblerprogramm die Symboltabelle. Ist der zugewiesene Symboltabellenspeicherbereich zu klein, um alle Symbole zu speichern, wird PC 100 SYM TBL OVERFLOW anzeigen und an den Assemblerprogrammanfang zurückkehren.

10. Ist der erste Durchlauf erfolgreich beendet, wird PC 100 automatisch den zweiten Durchlauf einleiten, wenn die Eingabe (IN=) vom Speicher (M) oder

Anwender-definiert (U) ist. Kommt die Eingabe von Kassette (T) oder Lochstreifen (L), wird der Assembler anhalten und anzeigen:

PASS 2

Spulen Sie das Band zurück und betätigen Sie SPACE, um den zweiten Durchlauf zu starten.

11. Der zweite Durchlauf wird durchgeführt. Die gewählte Auflistung der Fehler bzw. volle Assemblerprogramm-Auflistung wird an den LIST-OUT-Baustein geleitet. Der assemblierte Maschinencode wird an den OBJ?/OBJ-LIST-Baustein geleitet. Nach Beendigung des zweiten Durchlaufes geht die Kontrolle zurück zum Monitorprogramm.

Alle Fehler, die während des zweiten Durchlaufes festgestellt werden, werden durch eine Zahl identifiziert, die mit dem Fehlercode übereinstimmt (siehe Tabelle):

**ERROR XX wobei XX = 01 bis 21

Nach Beendigung der Assemblerprogramm-Auflistung wird die Anzahl der festgestellten Fehler gemeldet:

ERROR= XXXX wobei XXXX = der Fehlerzählstand

ANMERKUNG

Etwaige .OPT LIS, NOL, ERR oder NOE Befehle im Anwenderprogramm haben Vorrang vor der Anwenderantwort auf das LIST?-Stichwort.

4.1 Assembler-Fehlermeldungen

	SYM TBL OVERFLOW Während des ersten Durchlaufes wurden mehr einmalige Symbole festgestellt, als in der Symboltabelle erlaubt sind. Die zugewiesene Symboltabellenlänge kann vergrößert werden, indem man entweder die Symboltabelle-Start- und/oder Endadresse durch erneute Eingabe des ersten Durchlaufes wieder verändert.
01	** UNDEFINED SYMBOL Der Assembler hat ein Symbol in einem Operandenausdruck gefunden, das nirgendwo im Quellcode definiert ist (als Kennsatz oder als das Empfangsfeld eines vergleichbaren Befehles). Dieser Fehler wird auch dann auftreten, wenn ein reservierter Name (A, X, Y, S oder P) als Symbol in einem Ausdruck verwendet wird.
02	** LABEL PREVIOUSLY DEFINED Das erste Feld, interpretiert als ein Symbol, wurde schon vorher mit einem Wert in der Symboltabelle definiert. Eine Bezugnahme auf ein auf Seite 0 definiertes Symbol, hat eine Verschiebung aller Adresswerte, zwischen Durchlauf 1 und Durchlauf 2, verursacht.
03	** ILLEGAL OR MISSING OPCODE Das Assemblerprogramm hat eine Zeile gefunden, die einen Label beinhaltet, gefolgt von einem Ausdruck, den es als Befehl zu interpretieren versuchte.
04	** ADDRESS NOT VALID Eine Adresse, zu der in einem Befehl oder in einem der Assemblerbefehle verwiesen wurde (.BYTE, .WORD oder .DBYTE) ist ungültig.

4.1 Assembler-Fehlermeldungen (Fortsetzung)

05	** ACCUMULATOR MODE NOT ALLOWED Nach einer gültigen Befehlsmnemonic und einer oder mehreren Leerstellen folgt der Buchstabe A, gefolgt von einem oder mehreren Leerstellen (die Adressierbetriebsart des Akkumulators kennzeichnend Der Assembler versuchte den Akkumulator als Operanden zu verwenden. Die Ausgabe im Befehl erlaubt allerdings den Bezug zum Akkumulator nicht.
06	** FORWARD REFERENCE TO PAGE ZERO Es wurde ein Operandenausdruck gefunden, der einen Vorwärtsverweis enthält.
07	** RAN OFF END OF LINE Diese Fehlermeldung erscheint, wenn der Assembler ein benötigtes Feld sucht und über das Ende der Zeilenabbildung hinausläuft, bevor das Feld gefunden wird.
08	** LABEL DOESN'T BEGIN WITH ALPHABETIC CHARACTER Das erste nicht leere Feld, das weder ein Kommentar noch ein gültiger Befehl ist, wird als Label angenommen. Allerdings beginnt das erste Zeichen des Feldes mit einem Zahlenzeichen (0-9), welches die Regeln des Symbolaufbaues bricht.
09	** LABEL GREATER THAN SIX CHARACTERS Das erste Feld auf der Zeile ist eine Zeichenfolge, die mehr als sechs Zeichen enthält. Da das vorangestellte Semikolon, das einen Kommentar kennzeichnet, fehlt, wird angenommen, daß es sich um ein Symbol handelt, dessen zugelassene Länge überschritten wurde.

4.1 Assembler-Fehlermeldungen (Fortsetzung)

10	** LABEL OR OPCODE CONTAINS NON-ALPHANUMERIC Das Kennsatz- oder Maschinencodefeld auf einer Zeile enthält (ungültigerweise) ein Zeichen, das nicht alphanumerisch ist.
11	** FORWARD REFERENCE IN EQUATE OR ORG Der Ausdruck auf der rechten Seite eines Gleichheitszeichens enthält ein Symbol, das vorher nicht definiert wurde.
12	** INVALID INDEX - MUST BE X OR Y Einem Operationscode folgt ein gültiger Operandausdruck. Diesem Ausdruck folgt ein Komma (das eine indizierte Adressierung kennzeichnet), und eine ungültige Zeichenfolge, wobei entweder X oder Y erwartet wurde. Diese Fehlermeldung wird gegeben, ob eine indizierte Adressierbetriebsart für die entsprechende Befehlsmnemonic gültig ist oder nicht.
13	** INVALID EXPRESSION Bei der Auswertung eines Ausdruckes fand das Assemblerprogramm ein Zeichen, das es nicht interpretieren konnte.
14	** UNDEFINED ASSEMBLER DIRECTIVE Ist ein Punkt das erste Zeichen in einem nicht leeren Feld, interpretiert das Assemblerprogramm die folgenden drei Zeichen als eine Assemblerprogrammanweisung. Es ist entweder eine ungültige Anweisung gefunden worden oder die ersten drei Zeichen einer der Möglichkeiten in dem .OPT Befehl sind nicht interpretierbar.
15	** INVALID PAGE ZERO COMMAND Ein ungültig Null-Seiten-indizierter Operand wurde gefunden, zum Beispiel STA 20,X. Ein ungültig nicht Null-Seiten-indizierter Operand, zum Beispiel STA 20, oder ein ungültiger Null-Seiten Operand ohne Indizierung ergibt Fehler 18.

4.1 Assembler-Fehlermeldungen (Fortsetzung)

17	** RELATIVE BRANCH OUT OF RANGE Ein Sprungbefehl kann nur 127 Bytes vorwärts oder 128 Bytes rückwärts verzweigen. Diese Fehlermeldung zeigt eine Verzweigung außerhalb der Reichweite an.
18	** ILLEGAL OPERAND TYPE FOR THIS INSTRUCTION Nachdem es eine Befehlsmnemonic gefunden hat, die implizierte Adressierung nicht erlaubt, geht das Assemblerprogramm in das Operandfeld und stellt den Operandentyp fest (indiziert, absolut, usw.). Diese Fehlermeldung wird ausgedruckt, wenn der gefundene Operandentyp für den Befehl ungültig ist.
19	** OUT OF BOUNDS ON INDIRECT ADDRESSING Eine indirekte Adresse wird als solche durch die Klammern erkannt, mit denen sie in einem Operandenfeld einer Befehlsmnemonic umgeben ist. Da indirekte Adressierung zwei Bytes der Speicherseite Null erfordert, muß die Adresse, die auf diesen Bereich hinweist, kleiner oder gleich 254 sein.
20	** A, X, Y, S, AND P ARE RESERVED LABELS Einer der fünf reservierten Namen (A, X, Y, S und P) ist als Symbol verwendet worden.
21	** PROGRAMM COUNTER NEGATIVE - RESET TO 0 Ein Versuch, auf eine negative Speicherstelle Bezug zu nehmen, verursacht diese Fehlermeldung und verursacht, daß der Programmzähler auf Null rückgestellt wird.

5. Assemblerprogramm-Ausdrücke

Assemblerprogramm-Ausdrücke sind brauchbare Werkzeuge zur Vereinfachung des Programmierens und zur Erzeugung von sowohl lesbarem und leicht veränderbarem Code.

Es gibt zwei Komponenten der Assemblerausdrücke: Elemente und Operatoren.

5.1 Elemente

Elemente können in drei unterschiedliche Arten klassifiziert werden:

- Konstanten
- Symbole
- Adreßzähler

5.1.1 Konstanten

Numerische Konstanten können zu verschiedenen Basen geschrieben werden. Die Basis wird bestimmt durch die Art des vorangestellten Zeichens, wie in der folgenden Tabelle definiert.

VORANGESTELLTES ZEICHEN	BASIS
(keins)	10 (Dezimal)
$	16 (Hexadezimal)
@	8 (Oktal)
%	2 (Binär)

Beispiele:

```
==0000
        *=$200
==0200
10      .BYT $10,10,@
10,%10
0A
08
02
AD10F7 LDA $F710
A51D   LDA 29
A57E   LDA @176
A56D   LDA %01101101
```

ASCII-Buchstaben-Konstanten, in Anführungszeichen gesetzt, werden verwendet, um Zeichenketten in der ASCII-Darstellung in den Speicher einzufügen.

Zum Beispiel:

```
25      .BYT '%','1'
M',''''
49274D
==0211
..
A950    LDA #'P
A927    LDA #''
A935    LDA #'5
```

Beachten Sie, daß zwei Anführungszeichen nötig sind, um ein Anführungszeichen im Speicher darzustellen. Im letzten Feld der .BYT-Anweisung, stellt daher das erste ein einfaches Anführungszeichen dar und das letzte schließt die Kette ab.

5.1.2 Symbole

Symbole sind Namen, die verwendet werden, um Zahlenwerte darzustellen. Sie können eine Länge von einem bis sechs alphanumerischen Zeichen haben, das erste Zeichen muß alphabetisch sein. Die 56 gültigen Opcodes (in Tabelle 5-2 aufgelistet) und die reservierten Symbole A, X, Y, S und P haben eine besondere Bedeutung für das Assemblerprogramm und dürfen als Symbole nicht verwendet werden.

Zum Beispiel:

```
==0218 VARBLE
       =$2C
==0218 DATA1
'3      BYT $A3,VARB
_E
2C
==021A LAB190
AD1802 LDA DATA1
A52C   LDA VARBLE
```

5.1.3 Adresszähler

Der Adresspegel, dargestellt durch das Zeichen "*", ist ein Folgezähler, der durch das Assemblerprogramm verwendet wird, um seine laufende Position im Speicher zu verfolgen. Er darf in Ausdrücken innerhalb eines Programmes frei verwendet werden.

Zum Beispiel:

```
021F    .DBY *
A02102 LDA *
```

5.2 Operationssymbole

Zwei arithmetische Operationssymbole sind in der 3500 Assemblersprache erlaubt:

OPERATIONSSYMBOL	OPERATION
+	Addition
-	Subtraktion

Die Auswertung der Ausdrücke erfolgt streng von links nach rechts, Klammergruppierungen sind nicht erlaubt; alle Operationszeichen sind gleichwertig.

Zusätzlich gibt es zwei spezielle Operationszeichen:

ZEICHEN	OPERATION
>	Auswahl des höherwertigen Byte
<	Auswahl des niederwertigen Byte

Die Operationszeichen < und > trennen einen Zwei-Byte-Wert in ein höherwertiges Byte bzw. ein niederwertiges Byte.

Zum Beispiel:

```
==0224 HIGH
AB      .BYT >$ABCD,<
*.5+<HIGH-%10
:5
27
A541    LDA (*+)$1AC2
A51A    LDA %101+7+$7
+@7
A503    LDA >HIGH-$40
+<65
```

Ausdrücke, die ausgewertet negative Werte ergeben, sind ungültig. Eine Zweier-Komplement Darstellung einer negativen Zahl muß als eine vorzeichenlose Konstante ausgedrückt werden (vorzugsweise Hexadezimal z.B. Schreibe "-1" als "$FF").

Besondere Anmerkung: Ausdrücke werden zum Zeitpunkt des Assemblierens ausgewertet, nicht zum Zeitpunkt der Ausführung.

6. Assemblerprogramm-Primäranweisungen

Assemblerprogramm-Quellanweisungen bestehen aus bis zu vier Feldern:

(Kennsatz) (Opcode (Operand)) (;Kommentar)

Klammern um ein Feld zeigen an, daß es sich um ein Optionsfeld handelt. Daher muß, obwohl keines der Felder vorgeschrieben ist, ein Opcodefeld vor einem Operandfeld stehen. Die Eingabe in das Assemblerprogramm ist in freier Form; ein beliebiges Feld kann in einer beliebigen Spalte beginnen.

Insbesondere bemerken Sie, daß wegen der reservierten Opcodes, der Anwender vor Kennsätze Leerstellen setzen kann. Ist kein Kennsatz vorhanden, kann ein

Opcode in die erste Spalte gesetzt werden.

Felder in einer Anweisung brauchen nur durch eine einzelne Leerstelle getrennt zu werden. Werden die Felder in dieser Art getrennt, wird der Assembler die Felder in Spalten ordnen und eine lesbare Auflistung produzieren. Das Programm des Anwenders kann dann auf der Kassette in einer hochkonzentrierten Form aufgezeichnet werden.

Beachten Sie auch, daß das Kommentarfeld (comment field) nach einem Semikolon stehen sollte. Wird das Semikolon ausgelassen, wird das Kommentarfeld nicht in seiner richtigen Spalte in der Auflistung gesetzt.

6.1 Kennsätze (Labels)

Ein Kennsatz ist eine Zeichenkette, bestehend aus einem bis sechs alphanumerischen Zeichen. Er muß mit einem alphabetischen Zeichen beginnen und muß als das erste Feld einer Zeile erscheinen, obwohl er in einer beliebigen Spalte beginnen kann. Die Verwendung des Kennsatzes ist eine Methode, um den augenblicklichen Wert des Adresspegels dem Symbol zuzuordnen, bevor der Rest der Zeile vom Assemblerprogramm verarbeitet wird. Kennsätze werden mit Befehlen als Verzweigungsadressen und mit Speicherdatenzellen als Bezüge in Operanden verwendet.

Eine Zeile, die nur einen Kennsatz enthält, ist gültig, sodaß mehrere Kennsätze dem gleichen Speicherplatz zugeordnet werden können, indem wir jeden auf eine getrennte Zeile setzen:

```
==022D SAME1
==022D SAME2
==022D SAME3
AD2D02 LDA SAME3
```

6.2 Opcodes und Operanden

Zwei getrennte Klassen von Assemblerprogrammbefehlen stehen dem Programmierer zur Verfügung: Maschinenbefehle und Assemblerprogramm-Anweisungen.

6.3 Maschinenbefehle

Die 56 gültigen Maschinenbefehlmnemonics (s.Seite 31) stellen die Operationen dar, die mit den 6500-Mikroprozessoren durchgeführt werden können. Wenn sie assembliert sind, generiert jede Mnemonic ein Byte Maschinencode, wobei das tatsächliche Bit-Muster abhängig ist von sowohl der im Opcodefeld definierten Operation und der Adressierart, die durch das Operandfeld bestimmt wird. Das Operandfeld kann einen oder zwei Adress-Bytes generieren.

6500 Mikroprozessor Befehlssatz - Alphabetische Reihenfolge

	ADC	Add Memory to Accumulator with Carry	
	AND	AND Memory with Accumulator	
	ASL	Shift Left One Bit (Memory or Accumulator)	
	BCC	Branch on Carry Clear	
	BCS	Branch on Carry Set	
	BEQ	Branch on Result Zero	
	BIT	Test Bits in Memory with Accumulator	
	BMI	Branch on Result Minus	
	BNE	Branch on Result Not Zero	
	BPL	Branch on Result Plus	
*	BRK	Force Break	
	BVC	Branch on Overflow Clear	
	BVS	Branch on Overflow Set	
*	CLC	Clear Carry Flag	
*	CLD	Clear Decimal Mode	
*	CLI	Clear Interrupt Disable Bit	
*	CLV	Clear Overflow Flag	
	CMP	Compare Memory and Accumulator	
	CPX	Compare Memory and Index X	
	CPY	Compare Memory and Index Y	
	DEC	Decrement Memory by One	
*	DEX	Decrement Index X by One	
*	DEY	Decrement Index Y by One	
	EOR	Exclusive-OR Memory with Accumulator	
	INC	Increment Memory by One	
*	INX	Increment Index X by One	
*	INY	Increment Index Y by One	
	JMP	Jump to New Location	
	JSR	Jump to New Location Saving Return Address	
	LDA	Load Accumulator with Memory	
	LDX	Load Index X with Memory	
	LDY	Load Index Y with Memory	
	LSR	Shift Right One Bit (Memory or Accumulator)	
*	NOP	No Operation	
	ORA	OR Memory with Accumulator	
*	PHA	Push Accumulator on Stack	
*	PHP	Push Processor Status on Stack	
*	PLA	Pull Accumulator from Stack	
*	PLP	Pull Processor Status from Stack	
	ROL	Rotate One Bit Left (Memory or Accumulator	
	ROR	Rotate One Bit Right (Memory or Accumulator)	
*	RTI	Return from Interrupt	
*	RTS	Return from Subroutine	
	SBC	Subtract Memory from Accumulator with Borrow	
*	SEC	Set Carry Flag	
*	SED	Set Decimal Mode	
*	SEI	Set Interrupt Disable Status	
	STA	Store Accumulator in Memory	
	STX	Store Index X in Memory	
	STY	Store Index Y in Memory	
*	TAX	Transfer Accumulator to Index X	
*	TAY	Transfer Accumulator to Index Y	
*	TSX	Transfer Stack Pointer to Index X	
*	TXA	Transfer Index X to Accumulator	
*	TXS	Transfer Index X to Stack Pointer	
*	TYA	Transfer Index Y to Accumulator	

Befehle nur gültig bei implizierter Adressierung

7. Operand-Adressierungsarten

7.1 Absolute Adressierung

Die absolute Adressierung ist vom Konzept her die häufigste; die Daten, die auf den Maschinencode folgen, werden als die Adresse eines Speicherplatzes behandelt, der die tatsächlich zu behandelnden Daten während des Befehlsschrittes enthält. Diese Adresse soll in der umgekehrten Reihenfolge gespeichert werden - also niederwertiges, dann höherwertiges Byte - zur Erhöhung der Verarbeitungseffektivität während der Ausführung.

Zum Beispiel:

```
==0230 PIA
       =$4C1F
==0230 LATCH
       =$4DE2
==0230 BUFF1
       =$54B0
==0230 START
       =$A000
==0230 EXTRTN
       =$C800
2C1F4C BIT PIA
CDD7C2 CMP $C2D7
CEBA54 DEC BUFF1+10
4DB054 EOR BUFF1
4C00A0 JMP START
2000C8 JSR EXTRTN
==0242
AD2F6C LDA %11011000
01011111
6ECF11 ROR $11CF
EDCF54 SBC BUFF1+$1F
8DE24D STA LATCH
```

7.2 Page Zero Addressing (Null-Seiten-Adressierung)

In der Praxis wird die Null-Seite -Adressierung (vom Konzept her mit der absoluten Adressierung identisch) am häufigsten verwendet. Sie erlaubt den Ausdruck des Befehles in zwei Byte anstatt drei; das niederwertige Byte der Datenadresse wird vom Speicher geholt, und das höherwertige Byte wird als Null angenommen. Alle Befehle, die in der absoluten Adressierung gültig sind, sind auch in der Null-Seite -Adressierungsart gültig, mit Ausnahme von JMP und JSR Befehlen (siehe Tabelle 5-2); der Assembler generiert automatisch den kürzest möglichen Code. Es ist eine gute Programmierangewohnheit, die Seite Null des Speichers (Speicherstelle 0-255) für die Angabe von Variablen zu reservieren.

ANMERKUNG

Die Variablen auf Seite Null müssen definiert werden, bevor auf sie verwiesen wird.

Zum Beispiel:

```
==024E MODE
       =$06
==024E KEY
       =$0C
==024E COUNTR
       =$37
==024E TTYBUF
       =$6B
6537   ADC COUNTR
247A   BIT $7A
C436   CPY MODE
E638   INC COUNTR+1
A66B   LDX TTYBUF
469A   LSR $21+@171
050C   ORA KEY
86AA   STX TTYBUF+$3
F
```

7.3 Unmittelbare Adressierung

Die unmittelbare Adressierungsart wird mit dem Zeichen "#" kodiert, gefolgt von dem Byte-Ausdruck; der Code, der in den Speicher eingegeben wird, wird wie Daten behandelt, die entsprechend dem Maschinencode angewendet werden.

Zum Beispiel:

```
==025E DOLLAR
       =$24
==025E LAB1
6903   ADC #3
29B5   AND #%10110101
1
E024   CPX #DOLLAR
A945   LDA #'E
A05E   LDY #<LAB1
```

7.4 Implizierte Adressierung

Fünfundzwanzig der sechsundfünfzig Befehle, gültig nur in der implizierten Adressierung, erfordern keinen Operanden - ihre Durchführung kann vollständig ausgeführt werden, ohne weitere Information als die im Opcode enthaltene. Diese Befehle sind mit einem * versehen in Tabelle 5-2.

Zum Beispiel:

```
00   BRK
D8   CLD
C8   INY
EA   NOP
68   PLA
60   RTS
8A   TXA
```

7.5 Akkumulator-Adressierung

Befehle, die die vier Shift-Operationen durchführen, haben zusätzlich zu der Adressierung die sich auf den Speicher bezieht, eine spezielle Betriebsart, die eine Manipulation des Akkumulators erlaubt. Die Verwendung dieser Adressierungsart erzeugt einen Ein- Byte - Maschinen - code, ähnlich wie bei implizierter Adressierung.

Zum Beispiel:

```
0A    ASL A
4A    LSR A
2A    ROL A
6A    ROR A
```

7.6 Relative Adressierung

Acht bedingte Verzweigungsbefehle stehen dem Programmierer zur Verfügung; normalerweise folgen diese unmittelbar auf Laden, Vergleichen, arithmetische und Shiftbefehle. Verzweigungsbefehle benutzen ausschließlich die relative Adressierung. Die Verzweigungsadresse ist ein Ein-Byte positiver oder negativer Offset vom Laufzeitprogrammzähler, ausgedrückt in Zweier-Komplement-Schreibweise. Zu dem Zeitpunkt, in dem die Verzweigungsadressenrechnung durchgeführt wird, zeigt der Programmzähler die erste Speicherstelle hinter dem Verzweigungsbefehlscode an. Daher wird der Zugang zu Verzweigungsadressen innerhalb von 129 Bytes vorwärts und 126 Bytes rückwärts vom Anfang des Verzweigungscode durch den Ein-Byte-Offset begrenzt (Eine Ein-Byte-Zweier-Komplementzahl ist beschränkt auf den Bereich von -128 bis +127 einschließlich). Ein Fehler wird zum Zeitpunkt des Assemblierens gekennzeichnet, wenn das Verzweigungsziel außerhalb der Grenzen der relativen Adressierung liegt.

Zum Beispiel:

```
9080    BCC *-126
==0275 HERE
F0FE    BEQ HERE
30FC    BMI *-2
707F    BVS *+129
```

7.7 Indizierte Adressierung

Die indizierte Adressierung (mit Indexregister X oder Y) vereinfacht manche Arten der Tabellenverarbeitung. Die Adresse, die als Operand angegeben wird, wird als Basisadresse behandelt, zu der der Inhalt von entweder des X- oder Y-Registers addiert wird, um so zu der effektiven Adresse der Speicherstelle zu gelangen, die die Daten enthält, mit denen zu operieren ist. Alle Befehle zur Durchführung von absolut indizierter Adressierung mit dem X-Register erlauben auch die gleiche Adressierung in der Null-Seiten-Adressierung; manche Befehle (LDX, LDY, STX und STY) erlauben Null-Seite indizierte Adressierung mit dem Y-Register.

Zum Beispiel:

```
==027B ARRAY
       =$0B
==027B NUMBUF
       =$50
==027B TABLE
       =$2200
7F4F00 ADC NUMBUF-1,
Y
DDC022 CMP TABLE,X
D60B    DEC ARRAY,X
5DCC22 EOR TABLE+$C,
X
B60B    LDX ARRAY,Y
3622    ROL >TABLE,X
968F    STX NUMBUF+$3
F,Y
```

7.8 Indirekte Adressierung

Das Konzept der indirekten Adressierung beinhaltet eine Komplexitätsebene höher als die der absoluten Adressierung. Die Operandadresse bezieht sich nicht auf eine Speicherstelle die Daten enthält, sondern auf eine Folge von zwei Speicherstellen, die die Adresse enthalten - gespeichert in der Reihenfolge niederwertiges Byte, höherwertiges Byte - der Stelle, die die eigentlichen Daten enthält, die zu verarbeiten sind. Wirklich indirekte Adressierung wird nur mit dem JMP Befehl angeboten; in anderen Fällen wird indizierte indirekte Adressierung mit dem X-Register und indirekte indizierte Adressierung mit dem Y-Register durchgeführt. Im Falle von indizierter indirekter Adressierung wird die indizierte Adresse berechnet, bevor die indirekte genommen wird, die Reihenfolge der Auswertung ist bei indirekt indizierter Adressierung umgekehrt.

ANMERKUNG

> Normale indirekte Adressierung findet statt, wenn das Indexregister Null enthält. Der JMP indirekt verwendet einen Operanden mit absoluter Länge (Zwei-Byte); andere erfordern, daß die Operandenadresse auf der Null-Seite liegt, zwischen 0 und 254 einschließlich.

Zum Beispiel:

```
==028C INDADR
       =$02
==028C CURSOR
       =$57
==028C OLDPTR
       =$7E
==028C NEXT
       =$D9
2102   AND (INDADR,X)

D17E   CMP (OLDPTR),Y

6CD900 JMP (NEXT)
B157   LDA (CURSOR),Y
```

```
Y
E102    SBC (INDADR),X
)
8157    STA (CURSOR),X
)
```

8. Assembleranweisungen

Das Programm besitzt neun Assembleranweisungen. Diese werden verwendet, um Symbol- und Adresspegelwerte zu setzen (=), Speicherplätze zu reservieren und zu initialisieren (.BYTE, .WORD, .DBYTE) und um die Assemblereingabe/-ausgabe und das Assembleraufllistungsformat (.PAGE, .SKIP) zu steuern. Diese sind alle mehr Assemblerzeitbefehle als Ausführungszeitbefehle.

8.1 Equate-Anweisung

Die Equate ("=")-Anweisung weist entweder einem Symbol oder dem Adresspegel den Wert eine Ausdrucks zu, der keinen Vorwärtsbezug besitzt (Symbole werden in einem folgenden Abschnitt des Codes definiert):

```
==0299
        *=$1800
==1800 TABLE2
        =$C800
==1800 WRDPTR
        =$2A
==1800 NUMPTR
        =WRDPTR+2
```

Ein Kennsatz, der mit einer Equate-Anweisung, die den Adresspegel erhöht, verwendet wird, reserviert Arbeitsbereichspeicherstellen; dies ist besonders brauchbar, wenn am Anfang eines Programmes nicht etikettierte Speicherplätze nacheinander zugewiesen werden:

```
==1000
        *=0
;EQUATE DIRECTIVE
==0000 EOT
        *=*+1
;RESERVE MEMORY
==0001 EOTADR
        *=*+2
==0003 BUFFER
        *=*+72
==004B ENDFLG
        *=*+1
```

Symbole, denen Ein-Byte-Werte zugewiesen worden sind, können als Assemblerkonstante programmiert werden - Assemblerzeitwerte, die durchweg im gesamten Programm verwendet werden und die zu einem späteren Zeitpunkt, wenn das Programm erneut assembliert ist, geändert werden können. Der Quellcode ist so ausgelegt, daß eine Änderung nur die Neuzuweisung der entsprechenden Assemblerkonstanten erfordert. Dieses wird als eine gute Programmiergewohnheit angesehen und ist eine sehr viel bessere Alternative zu der Methode die jede Konstante verändert, wie sie im Laufe eines Programmes vorkommt:

```
==004C
        *=0
 EQUATE DIRECTIVE
==0000 STRTCH
        =$28
 ASSEMBLER CONSTANTS
==0000 ENDCH
        =$29
==0000 DELIM
        =$2C
==0000 LOWCH
        =$41
==0000 HIGHCH
        =$5A
==0000 KEYLEN
        =4
==0000 BUFLEN
        =72
```

8.2 .BYTE-Anweisung

Die .BYTE-Anweisung initialisiert Byte-Speicherstellen. Mehrfachargumente, durch Kommas getrennt, dürfen in einem einzigen .BYTE Befehl definiert werden,

um aufeinander-folgende Speicherstellen zu laden; entweder ASCII-Ketten oder Ausdrücke, die einen Acht-Bit-Wert auswerten, sind gültig. ASCII-Ketten in .BYTE-Anweisungen dürfen nicht mehr als 20 Zeichen erzeugen:

```
==0000 ASCII
4142    .BYT 'ABCD',''
EF',''JOE''S'
454648
4A4F
00      .BYT <ASCII,>
ASCII+2-*1,<**>*,2
01
0E
02
```

Beachten Sie die Verwendung der beiden Anführungszeichen innerhalb einer ASCII-Kette, um ein einfaches Anführungszeichen in den Speicher einzufügen.

8.3 .WORD-Anweisung

Die .WORD-Anweisung ist sehr brauchbar bei der Erstellung von Sprungtabellen und bei der Initialisierung von Zeigern. Ein Operandausdruck wird ausgewertet als eine Zwei-Byte-Adresse und in der Reihenfolge (niedrigwertiges Byte, höherwertiges Byte) gespeichert, in der der Mikroprozessor Adressen vom Speicher holt. Wie bei .BYTE sind Mehrfach-Operand-Felder, durch Kommas getrennt, erlaubt.

```
==0010 JMPTBL
04C8    .WOR $C804,$E
489,$F77A
B9E4
7AF7
0200    .WOR $2,<JMPT
BL,>JMPTBL,*,@6371
1000
0000
1C00
F90C
```

8.4 .DBYTE-Anweisung

Sollte gewünscht werden, einen 16-Bit Ausdruck-Wert in normaler höherwertiger, niederwertiger Byte-Reihenfolge zu erzeugen, muß die .DBYTE-Anweisung verwendet werden. Ihre Syntax-Regeln sind die gleichen wie für .WORD:

```
==0020 DATA
C804    .DBY #C804,$E
4B9,$F77A
E4B9
F77A
000E    .DBY $E,<DATA
,>DATA,*,%0101101111
01
0020
0000
002C
05BD
```

8.5 .PAGE-Anweisung

Die .PAGE-Anweisung verursacht das Drucken einer Überschriftszeile unter einer gestrichelten Linie. Ein Titel kann als eine ASCII-Kette im Operandfeld spezifiziert sein, und er kann mit einer Kette von einer oder mehreren Leerstellen gelöscht werden. Das Fehlen eines Operands verursacht auch die Löschung des Titels. Dieser Befehl wird bei der Eingabe in den Quellcode nicht ausgedruckt - nur die Ergebnisse erscheinen. Zum Beispiel, die Eingabe von:

```
.PAG  'ORIGINAL TITL
E'
.PAG
PAG   'NEW TITLE'
.PAG
```

würde das Erscheinen von Folgendem am oberen Rand jeder Seite verursachen:

```
ORIGINAL TITLE

NEW TITLE
```

8.6 .SKIP-Anweisung

Eine Leerzeile kann mit der .SKIP-Anweisung in das Programmprotokoll eingefügt werden.

```
*=0
.SKI
CURSOR *=*+2
EOT *=*+2
.SKI
TWO =EOT
```

Dieses verursacht den Ausdruck der folgenden Auflistung:

```
          *=0

=0000     CURSOR
          *=*+2
=0002     EOT
          *=*+2

=0004     TWO
          =EOT
```

8.7 .OPT-Anweisung

Die drei Alternativen der .OPT-Anweisung steuern die Erzeugung von Ausgabedateien und die Erweiterung von ASCII-Ketten in .BYTE-Anweisungen. Diese Alternativen werden wie folgt bestimmt:

```
.OPT LIST, GENERATE, ERRORS
```

und werden durch folgende Codierung eliminiert:

```
.OPT NOLIST, NOGENERATE, NOERRORS
```

Da nur die ersten drei Zeichen jeder Alternative abgetastet werden, können sie wie folgt geschrieben werden:

```
.OPT LIS, GEN, ERR
.OPT NOL, NOG, NOE
```

Von diesen Alternativen bleibt nur GEN/NOG nach dem Beginn des zweiten Durchlaufs unbestimmt; GEN/NOG hat einen Default-Wert von NOG, d.h. es wird automatisch NOG aufgenommen, falls nicht GEN spezifiziert wurde.
Die drei Alternativen haben folgende Funktionen:

1. LIST (NOLIST) steuert die Erzeugung des Programmprotokolls, das assemblierte Quelleingaben enthält, erzeugt den Maschinencode, Fehler und Warnungen.

2. GENERATE (NOGENERATE) steuert den Ausdruck des Maschinencodes für ASCII-Ketten in der .BYTE-Anweisung. Nur der Code für die ersten zwei Zeichen wird aufgelistet wenn NOG bestimmt ist; sonst wird das gesamte Literal erweitert.

3. ERRORS (NOERRORS) steuert nur die Auflistung von fehlerhaften Programmquellzeilen, zusammen mit den dazugehörig erzeugten Meldungen. Assembler-Tabellen - Überläufe werden auch in dieser Datei gemeldet.

8.8 .FILE-Anweisung

Bei großen Programmen ist es normalerweise bequemer, wenn man das Quellprogramm in logische Segmente aufteilt, die getrennt in den Textaufbereitungsprogrammpuffer geladen und aufbereitet werden können. Nach der Aufbereitung wird jede Datei vom Textpuffer in eine getrennte Datei auf der Kassette gespeichert. . Sollte das gesamte Programm assembliert werden, ist es allerdings notwendig, diese Dateien zusammenzubinden. Diese Funktion wird mit der .FILE-Assembler-Anweisung durchgeführt. Jede Datei (außer der letzten) enthält als letzte Aufzeichnung eine .FILE-Anweisung, die zu der nächsten Datei in der Kette zeigt.

```
.FILE NAME
```

Ist die erste Datei PRGM, dann wäre

```
.FILE QARC          die letzte Aussage in Datei PRGM
.FILE DEF       die letzte Aussage in Datei QARC
.FILE PATCH         die letzte Aussage in Datei DEF
```

8.9 .END-Anweisung

Die letzte Aussage der letzten Datei im Quellprogramm muß die .END-Anweisung sein. Zum Beispiel:

```
.END
```

ist die letzte Aussage in einem Ein-Datei-Programm und die letzte Aussage der letzten Datei in einem Mehrfach-Datei-Programm.

9 Bemerkungen (Comments)

Comments (Bemerkungen) können frei nach dem letzten Feld in einer Zeile in den Quellcode eingefügt werden. Wenn ein Opcode (und möglicherweise ein Operand) -Feld vorangeht, kann der Comment (die Bemerkung) wahlweise mit einem Semikolon (;) beginnen. Sonst ist das Semikolon zwingend notwendig. Ein Comment kann das einzige Feld auf einer Zeile sein.

Zum Beispiel:

```
==0000
        *=$0200
==0300
A900    LDA #0
 COMMENT FOLLOWING S
EMI-COLON
A900    LDA #0 COMMEN
T NOT FOLLOWING SEMI
-COLON
```

SIEMENS AKTIENGESELLSCHAFT

Bestell-Nr. 2135
Printed in Germany
KG 8792.