Ausgabe 1980/81

Assembler-Handbuch Personal-Computer PC 100

SIEMENS

Assembler-Handbuch Personal Computer PC 100

Ausgabe 1980/81

SIEMENS AKTIENGESELLSCHAFT

Liebe Leser!

Technische Dokumentation geben wir mit dem Ziel heraus, Sie beim Einsatz unserer Produkte zu unterstützen. Bei der Erarbeitung von Form und Inhalt der benötigten Information sind wir jedoch auch auf Ihre Hilfe angewiesen.

Wertvolle Mitarbeit bei der Verbesserung unserer Produktinformation können Sie durch Hinweise zu folgenden Fragen leisten:

1. Welche Begriffe oder Beschreibungen sind unverständlich?

2. Welche Ergänzungen und Erweiterungen schlagen Sie vor?

3. Wo haben sich inhaltliche Fehler eingeschlichen?

4. Welche Druckfehler haben Sie gefunden?

Antworten und sonstige Anregungen richten Sie bitte an:

Siemens Aktiengesellschaft
Unternehmensbereich Bauelemente
Vertrieb/Produktionsinformation
Balanstraße 73
8000 München 80

Zugehörige Druckschriften

Benötigen Sie zur Ergänzung Ihrer Informationen weitere technische Unterlagen, so fordern Sie bitte die aktuelle Angebotsliste „Produktinformation zum Thema Mikrocomputer" an.

Die halbjährlich neu erscheinende Angebotsliste mit anhängender Bestellkarte bekommen Sie bei ihrer nächstgelegenen Siemens-Dienststelle (siehe Geschäftsstellenverzeichnis).

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

Mit den Angaben 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 Landesgesellschaft im Ausland (siehe Geschäftsstellenverzeichnis).

Inhaltsverzeichnis

Inhaltsverzeichnis

1. Allgemeines

Der Übersetzungsvorgang einzelner Computerprogramm-Befehle, die in mnemonischer oder symbolischer Form (Quellprogramm) geschrieben sind, in tatsächliche Maschinen-Befehle (Maschinenprogramm), wird als Assemblierung bezeichnet. Das Computerprogramm, das die Übersetzung durchführt, ist das Assembler-Programm oder Assembler. Die mnemonischen Symbole und die Symbole, die beim Schreiben des Quellprogramms (Quellcode) verwendet werden, bezeichnet man als Assembler-Sprache. Ein Befehl in Assembler-Sprache läßt sich in einen Befehl in der Maschinensprache übersetzen. Das Maschinenprogramm wird im allgemeinen als Maschinencode bezeichnet.

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

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

Die Operation des Assembler-Programms geschieht innerhalb jeden Durchlaufs absolut automatisch, sobald Sie Informationen spezifiziert haben, wie z.B.: Herkunft des eingegebenen Quellcodes, Ziel des auszugebenden Maschinencodes, volle Assembler-Programmauflistung bzw. nur Fehlerauflistung.

Die Assembler-Befehle, die in dem Quellcode enthalten sind, liefern Zusatz- und Korrekturbefehle für die Erzeugung der Listen. Wird die Kasette als Eingabe verwendet, erlaubt eine Datei-Verbindung die Verwendung von Mehrfachdateien.

1.1 Einbau des Assembler-ROMs

Bevor Sie das Assembler-ROM in die Hand nehmen, sollten Sie, um Schäden am PC 100 oder an angeschlossenen Geräten zu vermeiden, folgende Vorsichtsmaßnahmen beachten:

- **Nicht abgedeckte Baugruppe!**

  Gegenstände die in die Baugruppe fallen oder auf ihr abgelegt werden, können den Drucker, die Anzeige und andere Bauteile beschädigen. Gelangt Flüssigkeit in die Baugruppe, können Kurzschlüsse entstehen.

- **MOS-Teile!**

  Mikrokomputerbauteile werden in Metall-Oxid-Silizium-Technologie (MOS) hergestellt. Unachtsames Anlegen hoher Spannungen können MOS-Beuteile zerstören.

- **Vorsichtsmaßnahmen:**

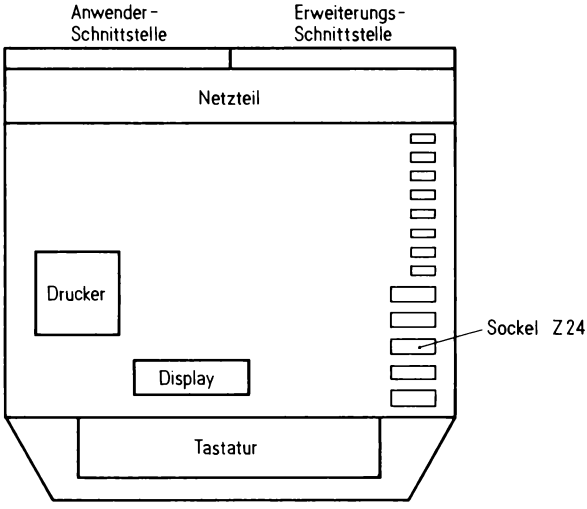
  - Entladen Sie Ihren Körper von statischen Aufladungen, indem Sie einen geerdeten Punkt berühren (etwa ein geerdetes Gerätegehäuse). Diese Vorsichtsmaßnahme ist besonders wichtig, falls Sie im Raum Teppichböden und eine niedrige relative Luftfeuchtigkeit haben.
  - Überprüfen Sie, ob alle Meßgeräte, alle peripheren Geräte und alle elektrischen Werkzeuge (z.B. Lötkolben) ordnungsgemäß geerdet sind.

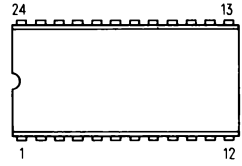
- **Offene Spannungen!**

  Die Versorgungsspannungen (5 V –; 24 V –) liegen an vielen Stellen der Baugruppe offen. Ein Kurzschluß dieser Stellen gegen Masse kann falsche Funktionen oder dauernden Schaden zur Folge haben.

  Schalten Sie die Spannungen des PC 100 ab. Prüfen Sie die Anschlüsse (PINs) des Assembler-ROM's ob sie gerade sind und ob keine Fremdkörper dazwischen stecken. Durch Gegenhalten unter den Sockel der Baugruppen einstecken.

  (ROM Nr. 3224 in den Sockel Z24). Beachten Sie die richtige Lage des Bausteins (PIN 1 in Sockelanschluß 1) und prüfen Sie, ob er korrekt im Sockel sitzt.





2. Symboltabelle

Für die Speicherung der Symboltabelle während des ersten Durchlaufs müssen im RAM Speicherstellen reserviert werden. Es müssen acht Bytes im RAM für jedes 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. Ist der zugeordnete Symboltabellenbereich nicht groß genug, beendet der Assembler den ersten Durchlauf, indem er folgende Fehlermeldung anzeigt:

Beispiel:

```
PASS 1

SYM TBL OVERFLOW
ASSEMBLER
```

Die Start- und Endadressengrenzen der Symboltabelle werden während des ersten Assembler-Durchlaufs eingegeben. Die tatsächliche Startadresse stimmt mit der eingegebenen Startadressengrenze überein. Die tatsächliche Endadresse wird niedriger als die Endadressengrenze sein.

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

Adresse	Parameter	Beispiel
$003A	Startadresse der Symboltabelle (LSB)[1])	$00
$003B	Startadresse der Symboltabelle (MSB)[1])	$03
$003C	Pointer auf das letzte abgespeicherte Symbol (LSB)	$88
$003D	Pointer auf das letzte abgespeicherte Symbol (MSB)	$03
$003E	Adressen-Obergrenze der Symboltabelle (LSB)	$FF
$003F	Adressen-Obergrenze der Symboltabelle (MSB)	$03
$000B	Anzahl der Symbole (HEX, LSB[1])	$00
$000C	Anzahl der Symbole (HEX, MSB[1])	$12

Die Adresse des letzten Symbols kann errechnet werden, indem man die Anzahl der Symbole in der Tabelle mit acht multipliziert und das Ergebnis zur Startadresse addiert. Es ist jedoch ebenso möglich, den Pointer auf das letzte abgespeicherte Symbol abzufragen.

[1]) LSB = niederwertiges Byte; MSB = höherwertiges Byte

Beispiel:

$Ø3ØØ + ($ØØ12 × 8) = $Ø3ØØ + $ØØ9Ø = $Ø39Ø

(Adresse des letzten Symbols)

Anmerkung:
Wenn der Quellcode, der Maschinencode und die Symboltabelle alle während des Assemblierens im RAM Platz finden sollen beachten Sie, daß Sie ein Überschreiben des Quellcodes entweder mit der Symboltabelle oder dem Maschinencode oder ein Überschreiben der Symboltabelle mit dem Maschinencode verhindern. Achten Sie besonders darauf, nicht den Quellcode zu überschreiben, er muß sonst noch einmal in den Textpuffer eingelesen werden. Es ist empfehlenswert, von Zeit zu Zeit den Quellcode auf Dauerspeichermedien (z.B. Kassette) zwischenzuspeichern, um einen unbeabsichtigten Verlust durch Überschreiben, Texteditor – Initialisierung oder durch Ausschalten zu verhindern. Bedenken Sie auch, daß der Assembler einen großen Teil der ZERO PAGE [1]) benutzt. Es kommt zum Überschreiben von Daten, wenn Sie versuchen, nach der Assemblierung eines Maschinenprogramms wieder in das vorher initialisierte BASIC zurückzukehren. (Siehe Kapitel 3.3)

[1]) ZERO PAGE ist der Speicherbereich von $ØØØØ bis $ØØFF

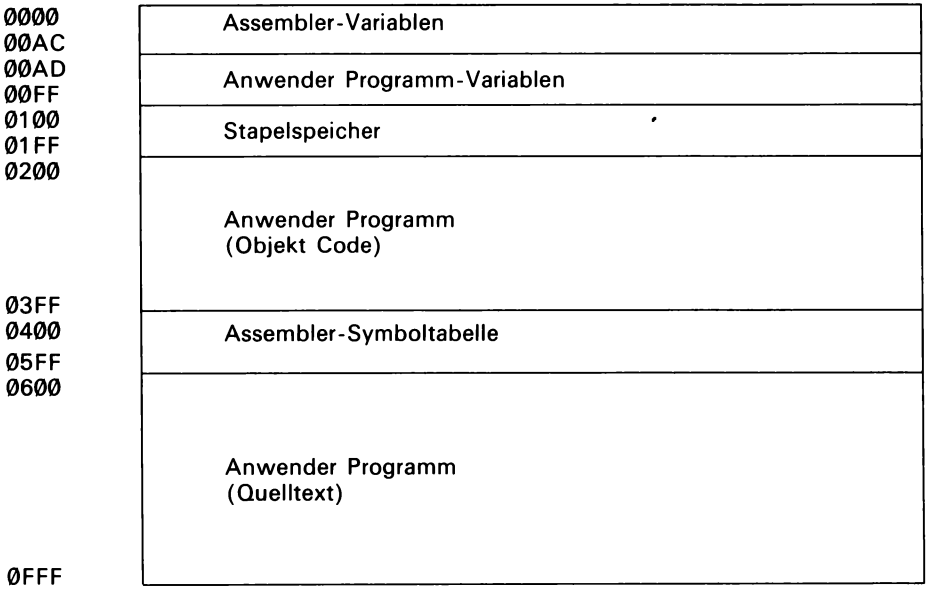
3. Assemblieren von Programmen

3.1 Speicher-zu-Speicher-Assemblierung

Die tatsächlichen Maschinencodeadressen können überprüft werden, indem man die Assemblerprogramm-Auflistung während des zweiten Durchlaufs ausdruckt, ohne den Maschinencode in den Speicher zu geben. Der Maschinencode kann entweder zur Kassette oder zu dem Blindbaustein (keine Ausgabe) adressiert 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. Wenn die Maschinencode-Adressen mit dem Quellcode kollidieren, sollte der Quellcode auf Kassette zwischengespeichert werden, bevor der Maschinencode geladen wird.

Wurde keine Maschinencode-Ausgabe generiert und zeigt die Überprüfung der Maschinencode-Adressen auf der Assembler-Programm-Auflistung keine Quellcode- oder Symboladresskollisionen, kann der Assembler neu gestartet werden, um den Maschinencode an den Speicher zu adressieren.

Beispiel:



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 ist es empfehlenswert den Quellcode vor der Assemblierung auf der Kassette zwischenzuspeichern. Durchlauf 1 und 2 sollten beide mit der Eingabe von der Kassette durchgeführt werden. Der ausgegebene Maschinencode wird also an eine andere Kassette adressiert (siehe Audio-Kassetten-Operation Kapitel 4.1.7). Die Größe des Programms wird jetzt nur durch die RAM Speicher beschränkt, um die Anzahl der Symbole im Quellcode aufzunehmen.

3.3 Speichern von Anwenderprogrammen

Der Assembler verwendet die Adressen $ØØØ4 bis $ØØDE (ZERO PAGE) und $Ø17Ø bis $Ø183 (Seite Eins). Aus diesem Grund sollten Sie während der Assemblierung in den Speicher (OBJ?N) keine Befehle oder Konstanten in diese Adressen assemblieren. Variablen des Anwenderprogramms können jedoch diesen Adressen zugewiesen werden. Nach der Assemblierung können Befehle oder Konstanten in diesen Adressen geladen werden.

4. Anwenden des Assemblerprogramms

4.1 Anwendungshinweise mit Beispielen

Verwenden Sie den PC 100 Assembler wie folgt:

4.1.1 Zum Aufruf des Assemblerprogramms betätigen Sie die Taste N nach der Anzeige des Monitorprogramm-Zeichens „<".

PC 100 antwortet mit:

```
ASSEMBLER
FROM=
```

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

Wurde zum Beispiel Ø3ØØ eingegeben, antwortet PC 100 mit:

```
FROM=300 TO=
```

Achtung!
Da fast die gesamte Seite Ø (ZERO PAGE) für Assemblerprogramm-Variablen und Seite 1 für Anwender- und Monitorprogramm-Stapel und für -Variablen des PC 100 reserviert sind, muß die Startadresse der Symboltabelle gleich oder größer als $Ø2ØØ sein.

4.1.3 Geben Sie die Symboltabellen-Endadresse hexadezimal ein. Beenden Sie die Adresseneingabe mit RETURN. Das Betätigen der Taste RETURN ohne die Eingabe eines Wertes bewirkt, daß der vorher eingetragene Wert als Endadresse eingegeben wird. Darauf wird entweder der neue oder vorher eingegebene Wert angezeigt. Wurde zum Beispiel Ø4ØØ eingegeben, antwortet PC 100 mit:

```
FROM=300  TO=400

IN=
```

4.1.4 Geben Sie den Code des Eingabebausteins ein, der den Quellcode enthält. Gültige Möglichkeiten sind:

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

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

Beachten Sie, daß der Quellcode in dem Augenblick angezeigt wird, in dem er von dem Eigabebaustein eingelesen wird.

- Wird M eingegeben, zeigt PC 100 „M". Gehen Sie zu Schritt 5 (Absatz 4.1.5.).
- Wird T eingegeben, fragt PC 100 nach dem Dateinamen:

```
IN=T F=
```

Anmerkung:
Bei Dateneingabe von Kassette muß darauf geachtet werden, daß der Kassettenrekorder eine Fernbedienung besitzt. Der Assembler erfaßt den Quellcode, indem er 80 Bytes auf einmal verarbeitet. Nach dem Einlesen des 1. Blockes (80 Bytes) wird der Kassettenrekorder vom PC 100 gestoppt. Er wird erst wieder neu gestartet, wenn der Assembler die 80 Bytes abgearbeitet hat.

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

Beispiel:

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

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

```
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, zeigt PC 100 die Suchmeldung und die Blockanzahl der gerade überlesenen Datei an. Wird der Dateiname PROG1 gelesen, antwortet PC 100 mit:

```
SRCH  F=PROG1  BLK=XX  (XX≙Blockzählerstand)
```

Sobald der eingegebene Dateiname auf dem Band gefunden ist, geht das Assemblerprogramm weiter mit Schritt 5 (Absatz 4.1.5.).

- Wurde L eingegeben,
  sollte das Einlesen des Quellcodes auf Lochstreifen vom TTY, eingeleitet werden.
- Wurde U eingegeben,
  wird der erste Durchlauf unter Verwendung der anwenderdefinierten Eingabe eingeleitet.

4.1.5 PC 100 fragt nun, ob die gesamte Assemblerprogramm-Auflistung oder nur eine Fehlerliste angezeigt oder ausgedruckt werden soll:

```
LIST?
```

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

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

4.1.6 PC 100 fragt, wohin die volle Assemblerprogramm- oder Fehlerauflistung gelenkt werden soll.

```
LIST-OUT=
```

Geben Sie eine der gültigen Möglichkeiten ein:

- RETURN oder Leertaste = Anzeige/Drucker
- P = Drucker
- T = Kassette (PC 100 Quellcode Format)
- U = Vom Anwender definiert
- L = TTY

4.1.7 PC 100 fragt nun, wohin der Maschinencode adressiert werden soll:

```
OBJ?
```

Soll der Maschinencode direkt in den Speicher geleitet werden, betätigen Sie Taste N. Gehen Sie weiter zu Schritt 9 (Absatz 4.1.9.).

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

Soll der Maschinencode an einen Ausgabebaustein und nicht an den Speicher geleitet werden, betätigen Sie Taste Y. PC 100 fragt nun nach dem Ausgabebaustein:

```
OBJ-OUT=
```

4.1.8 Betätigen Sie eine der gültigen Codemöglichkeiten:

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

Anmerkung:
Die ausgewählten OBJ-OUT = Option darf nicht mit der vorher gewählten LIST-OUT Option kollidieren; sonst wird beides, Auflistung und Maschinencodeausgabe, an den gleichen Ausgabebaustein in einer gemischten Form geleitet.

LIST-Option	OBJ-Ausgabe-Optionen					
	OBJ?N	OBJ?Y OBJ-OUT=				
	(M)	X	T	K	U	RETURN oder SPACE
RETURN oder SPACE	*	*	*	*	*	
P	*	*	*	*	*	
T	*	*			*	*
U	*	○	○	○	○	○

* Erlaubte OBJ-Ausgabe-Optionen
○ Nicht erlaubte Optionen bzw. hängt von der gewählten Eingabe ab.

4.1.9 PC 100 leitet den ersten Durchlauf ein.

```
PASS 1
```

Während des ersten Durchgangs generiert das Assemblerprogramm die Symboltabelle. Ist der zugewiesene Symboltabellen-Speicherbereich zu klein, um alle Symbole zu speichern, zeigt PC 100

```
SYM TBL OVERFLOW
```

an und kehrt an den Assemblerprogrammanfang zurück.

4.1.10 Ist der erste Durchlauf erfolgreich beendet, leitet PC 100 automatisch den zweiten Durchlauf ein, wenn die Eingabe (IN=) vom Speicher (M) oder anwenderdefiniert (U) ist. Kommt die Eingabe von Kassette (T) oder Lochstreifen (L), hält der Assembler an.

```
PASS 2
```

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

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

Alle Fehler, die während des zweiten Durchlaufs festgestellt werden, werden durch eine Zahl (Fehlernummer) identifiziert, die einem Fehlercode entspricht (siehe Kapitel 4.1 Assembler-Fehlermeldungen):

** ERROR XX (XX≙Ø1 bis 21)

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

ERRORS=XX (XXXX≙Anzahl der Fehler (dezimal)

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

4.2 Assembler-Fehlermeldungen

Fehler-nummer	Fehlermeldung und Bedeutung
(ohne)	**SYM TBL OVERFLOW** Während des ersten Durchlaufs 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 im ersten Durchlauf verändert.
01	**SYMBOL NICHT DEFINIERT** Der Assembler hat ein Symbol in einem Operanden-Ausdruck gefunden, das nicht im Quellcode definiert ist (Kennsatz oder als das Empfangsfeld eines vergleichbaren Befehls). Dieser Fehler tritt auch dann auf, wenn ein reservierter Name (A, X, Y, S oder P) als Symbol in einem Ausdruck verwendet wird.
02	**KENNSATZ BEREITS DEFINIERT** Das erste Feld, interpretiert als Symbol, wurde schon vorher mit einem Wert in der Symboltabelle definiert. Eine Bezugnahme auf ein in der ZERO PAGE definiertes Symbol, hat eine Verschiebung aller Adresswerte zwischen Durchlauf 1 und 2 verursacht.
03	**FALSCHER ODER FEHLENDER BEFEHL** Das Assemblerprogramm hat eine Zeile gefunden, die einen Label beinhaltet, gefolgt von einem Ausdruck, den es als Befehl zu interpretieren versuchte.
04	**ADRESSE UNGÜLTIG** Eine Adresse, auf die in einem Befehl oder in einem der Assemblerbefehle (.BYTE, .WORD oder .DBYTE) verwiesen wurde, ist ungültig.
05	**NICHT ERLAUBTE AKKUMULATOR-BEZUGNAHME** Nach einer gültigen Befehls-Mnemonic und einer oder mehrerer Leerstellen erscheint der Buchstabe A, gefolgt von einem oder mehreren Leerstellen (Kennzeichnung der Adressier-Betriebsart des Akkus). Der Assembler versuchte, den Akkumulator als Operanden zu verwenden. Die Ausgabe im Befehl erlaubt jedoch den Bezug zum Akkumulator nicht.
06	**VORWÄRTSVERWEIS AUF ZERO PAGE** Es wurde ein Operandenausdruck gefunden, der einen Vorwärtsverweis auf einen bisher nicht definierten „ZERO PAGE" Operanden enthält.
07	**ZEILENENDE ÜBERLAUFEN** Diese Fehlermeldung erscheint, wenn der Assembler ein benötigtes Feld sucht und über das Ende der Zeilenabbildung hinausläuft, bevor das Feld gefunden wird.

4.2 Assembler-Fehlermeldungen (Fortsetzung)

Fehler-nummer	Fehlermeldung und Bedeutung
08	**LABEL BEGINNT NICHT MIT EINEM BUCHSTABEN** Das erste nicht leere Feld, das weder ein Kommentar noch ein gültiger Befehl ist, wird als Label angenommen. Das erste Zeichen des Feldes beginnt mit einer Zahl (Ø bis 9). Das jedoch steht im Widerspruch mit den Symbolaufbauregeln.
09	**LABEL LÄNGER ALS SECHS ZEICHEN** Das erste Feld auf der Zeile ist eine Zeichenfolge, die mehr als sechs Zeichen enthält. Da das vorangestellte Semikolon (Kennzeichnung eines Kommentars) fehlt, wird angenommen, daß es sich um ein Symbol handelt, dessen zulässige Länge überschritten wurde.
10	**LABEL ODER BEFEHL ENTHÄLT EIN NICHT ALPHANUMERISCHES ZEICHEN** Das Kennsatz- oder Maschinencodefeld in einer Zeile enthält ein Zeichen, das nicht alphanumerisch ist.
11	**Vorwärtsbezugnahme in einer Wertzuweisung** Der Ausdruck auf der rechten Seite eines Gleichheitszeichens enthält ein Symbol, das vorher nicht definiert wurde.
12	**UNGÜLTIGER INDEX (X ODER Y WURDE ERWARTET)** Einem Operationscode folgt ein gültiger Operandausdruck. Diesem Ausdruck folgen ein Komma (Kennzeichnung einer indizierten Adressierung) und eine ungültige Zeichenfolge, wobei entweder „X" oder „Y" erwartet wurde. Diese Fehlermeldung erscheint, wenn eine indizierte Adressierbetriebsart für die entsprechende Befehlsmnemonic ungültig ist.
13	**UNGÜLTIGER AUSDRUCK** Bei der Auswertung eines Ausdrucks fand das Assemblerprogramm ein Zeichen, das es nicht interpretieren konnte.
14	**UNGÜLTIGE ASSEMBLERANWEISUNG** Ist das erste Zeichen in einem nicht leeren Feld ein Punkt, 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	**UNGÜLTIGES ZERO PAGE KOMMANDO** Ein ungültiger Null-Seiten-indizierter Operand wurde gefunden, z.B. STA #2Ø,X. Ein ungültiger nicht Null-Seiten-indizierter Operand, z.B. STA #2Ø, oder ein ungültiger Null-Seiten Operand ohne Indizierung ergibt Fehler 18.
16	Für Erweiterungen freigehalten

4.2 Assembler-Fehlermeldungen (Fortsetzung)

Fehler-nummer	Fehlermeldung und Bedeutung
17	**RELATIVE SPRUNGADRESSE LIEGT AUSSERHALB DES SPRUNGBEREICHS** Ein Sprungbefehl kann nur 127 Bytes vorwärts oder 128 Bytes rückwärts verzweigen. Diese Fehlermeldung zeigt eine Verzweigung außerhalb der Reichweite an.
18	**OPERANDENTYP FÜR DIESE ANWEISUNG UNZULÄSSIG** Wenn eine Befehlsmnemonic gefunden wird, 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	**INDIREKTE ADRESSE AUSSERHALB DER GRENZEN** 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, einen Wert zwischen Ø und 254 haben.
20	**VERWENDUNG EINES RESERVIERTEN NAMENS ALS SYMBOL** Einer der fünf reservierten Namen (A, X, Y, S und P) ist als Symbol verwendet worden.
21	**PROGRAMMZÄHLER NEGATIV** Der Versuch, auf eine negative Speicherzelle Bezug zu nehmen, bedingt diese Fehlermeldung und verursacht, daß der Programmzähler auf „Null" zurückgestellt wird.

5. Assemblerprogramm-Ausdrücke

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

Es gibt zwei Assemblerausdruck-Komponenten:

- Elemente und
- Operatoren

5.1 Elemente

Elemente werden in drei unterschiedlichen Arten klassifiziert:

- Konstanten
- Symbole
- Adresszähler

5.1.1 Konstanten

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

Vorangestelltes Zeichen	Basis
(keins)	10 (Dezimal)
$ (Dollarzeichen)	16 (Hexadezimal)
@ (kommerzielles a)	8 (Oktal)
% (Prozentzeichen)	2 (Binär)
' (Komma)	ASCII

Beispiel:

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

25        .BYT '%','I','M',''''
49
4D
==0210
27
A950      LDA #'P
A927      LDA #'''
A935      LDA #'5
```

Beachten Sie, daß, um ein Anführungszeichen im ASCII-Code im Speicher darzustellen, zwei Anführungszeichen nötig sind. Hinter einer „. BYT-Anweisung" stellt daher das erste Anführungszeichen ein „einfaches Anführungszeichen" dar und erst 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 (Tabelle 6.3) und die reservierten Symbole „A, X, Y, S und P" haben eine besondere Bedeutung für das Assemblerprogramm und dürfen als Symbole nicht verwendet werden.

Beispiel:

```
==0217 VARBLE
       =$2C
==0217 DATA1
A2     .BYT $A2,VARBLE
2C
==0219 MARKE1
AD1702 LDA DATA1
A52C   LDA VARBLE
A92C   LDA #VARBLE
```

5.1.3 Adresszähler

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

Beispiel:

```
0220   .DBY *
AD2202 LDA *
```

5.2 Operationssymbole

Zwei arithmetische Operationssymbole sind in der Assemblersprache erlaubt:

Symbol	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 Bytes
<	Auswahl des niederwertigen Bytes

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

Beispiel:

```
==0225 DATA2
AB     .BYT >$ABCD,<*,5+<DATA2-%10
26
28
A542   LDA <*+>$1AC2
A51A   LDA %101+7+$7+@7
A503   LDA >DATA2-$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").

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

6. Assemblerprogramm-Primäranweisungen

Assemblerprogramm-Quellanweisungen bestehen aus bis zu vier Feldern:

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

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

Anmerkung:
Wegen der reservierten Opcodes kann der Anwender vor Kennsätze Leerstellen setzen. Ist kein Kennsatz vorhanden, kann ein Opcode in die erste Spalte gesetzt werden.

Felder in einer Anweisung müssen durch mindestens eine Leerstelle getrennt werden. Der Assembler ordnet sie dann in Spalten und produziert eine lesbare Auflistung. Das Anwenderprogramm kann jetzt auf Kassette in hochkonzentrierter Form aufgezeichnet werden.

Beachten Sie, daß einem Kommentarfeld (comment field) ein Semikolon voranstehen muß. Läßt man das Semikolon aus, wird das Kommentarfeld nicht in seiner richtigen Spalte in der Auflistung gesetzt.

6.1 Kennsätze

Ein Kennsatz (Label) ist eine Zeichenkette, bestehend aus einem bis sechs alphanumerischen Zeichen. Der Satz muß mit einem alphabetischen Zeichen beginnen und als erstes 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 ist auch gültig, die nur einen Kennsatz enthält. Man darf mehrere Kennsätze dem gleichen Speicherplatz zuordnen, indem jedes in eine separate Zeile geschrieben wird.

Beispiel:

```
==022E MARKE2
==022E MARKE3
==022E MARKE4
AD1902 LDA MARKE1
```

6.2 Opcodes und Operanden

Zwei getrennte Assemblerprogrammbefehl-Klassen stehen dem Programmierer zur Verfügung:

- Maschinenbefehle (Mikroprozessor-Befehle)
- Assemblerprogramm-Anweisungen

6.3 Maschinenbefehle (Mikroprozessor-Befehle)

56 gültige Maschinenbefehl-Mnemonics (siehe auch Seiten 27, 52 und 83) stellen die Operationen dar, die mit den Mikroprozessoren durchgeführt werden. Wenn sie assembliert sind, generiert jede Mnemonic ein Byte Maschinencode. Das tatsächliche Bit-Muster ist abhängig von der im Opcodefeld definierten Operation und der Adressierart. Die Adressierungsart ist durch das Operandfeld bestimmt. Das Operandfeld kann einen oder zwei Adress-Bytes generieren.

6.3.1 Befehlsmnemonic (Operation/Funktion)

Befehls-mnemo-nic	Operation/Funktion	
ADC	Add Memory to Accumu-lator with Carry	Addiere Speicherzelle und Carry-Flag zum Akku-Inhalt
AND	AND Memory with Accu-mulator	Logische „UND"-Verknüpfung von Speicherzelle und dem Akku
ASL	Shift Left One Bit (Memory or Accumulator)	Schiebe Speicherzelle oder Akku um ein Bit nach links
BCC	Branch on Carry Clear	Relativer Sprung, wenn Carry-Flag = Ø
BCS	Branch on Carry Set	Relativer Sprung, wenn Carry-Flag = 1
BEQ	Branch on Result Zero	Relativer Sprung, wenn Zero-Flag = 1
BIT	Test Bits in Memory with Accumulator	Vergleiche die Bits einer Speicherzelle mit Akku
BMI	Branch on Result Minus	Relativer Sprung, wenn Negativ-Flag = 1
BNE	Branch on Result Not Zero	Relativer Sprung, wenn Zero-Flag = Ø
BPL	Branch on Result Plus	Relativer Sprung, wenn Negativ-Flag = Ø
* BRK	Force Break	Programm-Unterbrechung
BVC	Branch on Overflow Clear	Relativer Sprung, wenn Overflow-Flag = Ø
BVS	Branch on Overflow Set	Relativer Sprung, wenn Overflow-Flag = 1
* CLC	Clear Carry Flag	Lösche Carry-Flag
* CLD	Clear Decimal Mode	Lösche Dezimalmodus
* CLI	Clear Interrupt Disable Bit	Lösche Interrupt Sperr-Flag
* CLV	Clear Overflow Flag	Lösche Overflow-Flag
CMP	Compare Memory and Accumulator	Vergleiche Speicherzelle mit Akku
CPX	Compare Memory and Index X	Vergleiche Speicherzelle mit X-Register
CPY	Compare Memory and Index Y	Vergleiche Speicherzelle mit Y-Register
DEC	Decrement Memory by One	Erniedrige Speicherzelle um eins
* DEX	Decrement Index X by One	Erniedrige X-Register um eins
* DEY	Decrement Index Y by One	Erniedrige Y-Register um eins
EOR	Exclusive-OR Memory with Accumulator	Logische „Exclusiv-ODER"-Verknüp-fung von Speicherzelle mit Akku

* Befehle nur bei implizierter Adressierung gültig

Befehls-mnemonic	Operation/Funktion	
INC	Increment Memory by One	Erhöhe Speicherzelle um eins
*INX	Increment Index X by One	Erhöhe X-Register um eins
*INY	Increment Index Y by One	Erhöhe Y-Register um eins
JMP	Jump to New Location	Springe in anderen Speicherbereich
JSR	Jump to New Location Saving Return Address	Springe in anderen Speicherbereich und rette Rückkehradresse (Unterprogrammsprung)
LDA	Load Accumulator with Memory	Lade Akku mit Speicherzelle
LDX	Load Index X with Memory	Lade X-Register mit Speicherzelle
LDY	Load Index Y with Memory	Lade Y-Register mit Speicherzelle
LSR	Shift Right One Bit (Memory or Accumulator)	Schiebe Speicherzelle oder Akku um ein Bit nach rechts
*NOP	No Operation	Keine Operation
ORA	OR Memory with Accumulator	Logische „ODER"-Verknüpfung von Speicherzelle mit Akku
*PHA	Push Accumulator on Stack	Speichere Akku auf Stapelspeicher
*PHP	Push Processor Status on Stack	Speichere Prozessorstatus-Register auf Stapelspeicher
*PLA	Pull Accumulator from Stack	Lade Akku mit Inhalt Stapelspeicher
*PLP	Pull Processor Status from Stack	Lade Prozessorstatus mit Inhalt Stapelspeicher
ROL	Rotate One Bit Left (Memory or Accumulator	Rotiere Speicherzelle oder Akku um ein Bit nach links
ROR	Rotate One Bit Right (Memory or Accumulator)	Rotiere Speicherzelle oder Akku um ein Bit nach rechts
*RTI	Return from Interrupt	Kehre aus Interrupt zurück
*RTS	Return from Subroutine	Kehre aus Unterprogramm zurück
SBS	Subtract Memory from Accumulator with Borrow	Subtrahiere Speicherzelle und negiertes Carry-Flag vom Akku
*SEC	Set Carry Flag	Setze Carry-Flag
*SED	Set Decimal Mode	Setze Dezimal modus
*SEI	Set Interrupt Diseable Status	Setze Interrupt Sperr-Flag

*) Befehle nur bei implizierter Adressierung gültig

Befehls-mnemo-nic	Operation/Funktion	
STA	Store Accumulator in Memory	Speichere Akku in Speicherzelle
STX	Store Index X in Memory	Speichere X-Register in Speicherzelle
STY	Store Index Y in Memory	Speichere Y-Register in Speicherzelle
TAX[1])	Transfer Accumulator to Index X	Transferiere Akku in X-Register
TAY[1])	Transfer Accumulator to Index Y	Transferiere Akku in Y-Register
TSX[1])	Transfer Stack Pointer to Index X	Transferiere Adresse des Stapelzeigers in X-Register[2])
TXA[1])	Transfer Index X to Accumulator	Transferiere X-Register in Akku
TXS[1])	Transfer Index X to Stack Pointer	Setze Adresse des Stapelzeigers auf den Wert des X-Registers[2])
TYA[1])	Transfer Index Y to Accumulator	Transferiere Y-Register in Akku

[1]) Befehle nur bei implizierter Adressierung gültig
[2]) Das höherwertige Byte ist bei diesen Operationen immer eins, da sich der Stapelzeiger stets zwischen $100 und $1FF befindet.

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 Speicherplatzadresse behandelt, die die tatsächlich zu verarbeitenden Daten während des Befehlsschritts enthält. Diese Adresse wird zur Erhöhung der Verarbeitungseffektivität während der Ausführung in umgekehrter Reihenfolge gespeichert (zuerst niederwertiges-, dann höherwertiges Byte).

Beispiel:

```
==0231 UT1L
       =$A004
==0231 UT1H
       =$A005
==0231 START
       =$200
==0231 DATA3
       =0
==0231 COMIN
       =$E1A1
2C04A0 BIT UT1L
CD05A0 CMP $A005
C60A   DEC DATA3+10
4500   EOR DATA3
4C0002 JMP START
20A1E1 JSR COMIN
==0241
A5D8   LDA %11011000
6600   ROR 0
E51F   SBC DATA3+$1F
8D05A0 STA UT1H
```

7.2 Null-Seite-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 Befehls in zwei Bytes, anstatt drei; das niederwertige Byte der Datenadresse wird vom Speicher geholt, und das höherwertige wird als Null angenommen. Alle Befehle, die in der absoluten Adressierung gültig sind, gelten auch in der „Null-Seite-Adressierungsart", mit Ausnahme von „JMP- und JSR-Befehlen" (siehe Kapitel 6.3); der Assembler generiert automatisch den kürzestmöglichen Code. Es ist sinnvoll, die Seite Null des Speichers (Speicherstelle $ØØ–$FF) für die Angabe von Variablen zu reservieren.

Anmerkung:
Die Variablen auf Seite Null müssen definiert sein, bevor auf sie verwiesen wird.

Beispiel:

```
==024A DATA4
       =$06
==024A DATA5
       =$0C
==024A ZAEHLR
       =$37
==024A TTYBUF
       =$6B
6537   ADC ZAEHLR
247A   BIT $7A
C406   CPY DATA4
E638   INC ZAEHLR+1
A66B   LDX TTYBUF
469A   LSR $21+@171
050C   ORA DATA5
86AA   STX TTYBUF+$3F
==025A
```

7.3 Unmittelbare Adressierung

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

Beispiel:

```
==025A DATA6
       =$24
==025A MARKE5
6903   ADC #3
29B5   AND #%10110101
E024   CPX #DATA6
A945   LDA #'E
A024   LDY #<DATA6
```

7.4 Implizierte Adressierung

25 der 56 Befehle, gültig nur in der implizierten Adressierung, erfordern keinen Operanden. Für ihre Durchführung wird lediglich die im Opcode enthaltene Information benötigt. Diese Befehle sind mit einem „*" versehen (siehe Kapitel 6.3. „Tabelle").

Beispiel:

```
00     BRK
D8     CLD
C8     INY
EA     NOP
68     PLA
60     RTS
==026A
8A     TXA
```

7.5 Akkumulator-Adressierung

Befehle, die die vier Shift-Operationen durchführen, haben zusätzlich zu der Speicheradressierung eine spezielle Betriebsart, die eine Manipulation des Akkumulators erlaubt. Die Verwendung dieser Adressierungsart erzeugt einen Ein-Byte-Maschinencode (ähnlich wie bei implizierter Adressierung).

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. Diese Verzweigungs-Befehle folgen normalerweise unmittelbar auf Lade-, Vergleiche-, Arithmetik- und Shiftbefehle. Verzweigungs-Befehle benutzen ausschließlich die relative Adressierung. Die Verzweigungsadresse ist ein Ein-Byte-Offset (positiv oder negativ) vom Laufzeitprogrammzähler, in Zweier-Komplement-Schreibweise. Zu dem Zeitpunkt, in dem die Verzweigungsadressenrechnung durchgeführt wird, zeigt der Programmzähler auf die Speicherstelle des Verzweigungsbefehls. Daher wird der Zugang zu Verzweigungsadressen innerhalb von 129 Bytes vorwärts und 126 Bytes rückwärts vom Anfang des Verzweigungscodes durch den Ein-Byte-Offset begrenzt (eine Ein-Byte-Zweier-Komplementzahl ist auf den Bereich von −128 bis +127 einschließlich beschränkt. Ein Fehler wird zum Zeitpunkt des Assemblierens gekennzeichnet, wenn das Verzweigungsziel außerhalb der Grenzen der relativen Adressierung liegt.

Beispiel:

```
D002    BNE *+4
9080    BCC *-126
==0273  MARKE6
F0FE    BEQ MARKE6
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 dieser wird der Inhalt des X- oder Y-Registers addiert, um so zu der effektiven Speicherstelle-Adresse 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-Seite-Adressierung".

Beispiel:

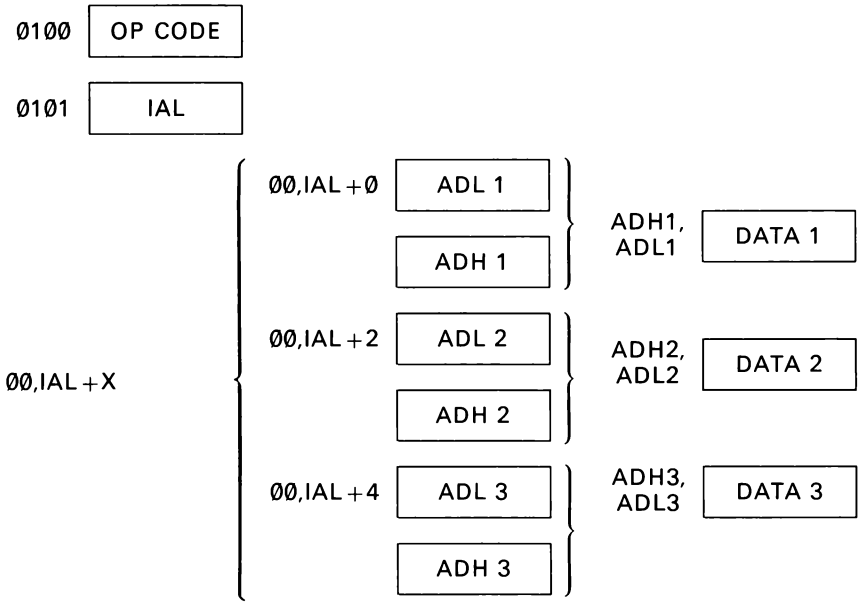
```
==0279 FELD1
       =$0B
==0279 ZAHLB
       =$50
==0279 TABELL
       =$F00
794F00 ADC ZAHLB-1,Y
DD000F CMP TABELL,X
D60B   DEC FELD1,X
5D0C0F EOR TABELL+$C,X
B60B   LDX FELD1,Y
360F   ROL >TABELL,X
967F   STX ZAHLB+$2F,Y
==028A
       .END
 ERRORS= 0000
```

7.8 Indirekte Adressierung

Das Konzept der indirekten Adressierung beinhaltet eine höhere Komplexitätsebene als die der absoluten Adressierung. Die Operandadresse bezieht sich nicht auf eine Speicherstelle, die Daten enthält, sondern auf eine Folge von zwei Speicherstellen. Diese enthalten die Adresse (Reihenfolge: niederwertiges – höherwertiges Byte) der Stelle, die die zu verarbeitenden Daten enthält. Wirklich indirekte Adressierung ist nur mit dem JMP-Befehl möglich; in anderen Fällen wird „indiziert indirekte Adressierung" mit dem X-Register, und „indirekt indizierte Adressierung" mit dem Y-Register durchgeführt.

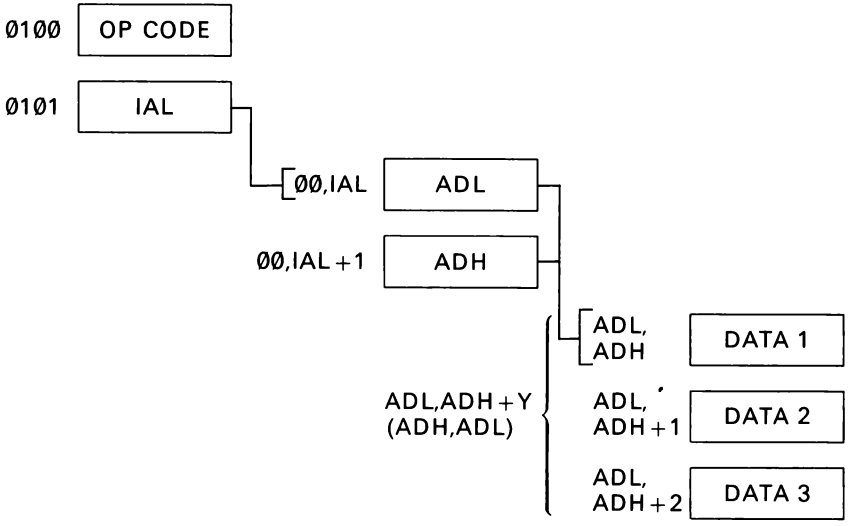
7.8.1 Indiziert indirekte Adressierung

Bei der „indiziert indirekten Adressierung" errechnet sich die Endadresse der zu bearbeitenden Variablen über eine in der ZERO PAGE aufgebauten Adressenliste. Die Summe aus dem Operanden und dem X-Register ergibt die Speicherzelle, in der das niederwertige Byte der Endadresse abgespeichert ist.



7.8.2 Indirekt indizierte Adressierung

Der Operand zeigt auf eine Speicherzelle in der ZERO PAGE. Die Summe aus dem Inhalt dieser Speicherzelle und dem Y-Register ergibt das niederwertige Byte (LSB) der Endadresse. Das höherwertige Byte (MSB) der Endadresse steht in der nächsten, durch den Operanden ausgewiesenen, ZERO PAGE-Adresse. Ergibt sich bei der Berechnung der Endadresse eine Seitenüberschneidung, wird diese mitberücksichtigt.



Anmerkung:
„Normale indirekte Adressierung" findet statt, wenn das Indexregister Null enthält. Nur der indirekte Befehl „JMP" verwendet einen Operanden mit absoluter Länge (Zwei-Byte). Die anderen erfordern, daß die Operandenadresse auf der Null-Seite liegt (zwischen $00 und $FF einschließlich).

Beispiel:

```
==028A DATA7
       =$02
==028A DATA8
       =$57
==028A DATA9
       =120
==028A DATA10
       =@70
2102   AND (DATA7,X)
D157   CMP (DATA8),Y
6C7800 JMP (DATA9)
B138   LDA (DATA10),Y
E157   SBC (DATA8,X)
8138   STA (DATA10,X)
```

8. Assembleranweisungen

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

8.1 EQUATE-Anweisung

Die EQUATE („=")-Anweisung weist einem Symbol oder dem Adresspegel den Wert eines Ausdrucks zu, der keinen Vorwärtsbezug besitzt:

Beispiel:

```
==0297
        *=$800
==0800 DATA11
        =$E000
==0800 ZAEHL1
        =$2A
==0800 ZAEHL2
        =ZAEHL1+2
```

Ein Kennsatz, der mit einer EQUATE-Anweisung, die den Adresspegel erhöht, verwendet wird, reserviert Arbeitsspeicher. Das ist besonders nützlich, wenn am Programmanfang Speicherplätze nacheinander zugewiesen werden:

Beispiel:

```
;RESERVIEREN VON SPEICHERZELLEN
==0800
        *=0
==0000 DATA12
        *=*+1
==0001 ADRR1
        *=*+2
==0003 PUFFER
        *=*+72
==004B FLAG
        *=*+1
```

Symbole, denen Ein-Byte-Werte zugewiesen wurden, können als Assemblerkonstante – Assemblerzeitwerte – programmiert werden, 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. Dies ist eine bessere Methode, als jede Konstante zu verändern.

Beispiel:

```
;ASSEMBLER KONSTANTEN
==004C DATA13
       =$28
==004C OUTPUT
       =$E97A
==004C RDRUB
       =$E95F
==004C T1LATC
       =%00110101
==004C CR
       =$0D
==004C LF
       =$0A
==004C DATA14
       =@127
==004C PUFFR1
       =5
```

8.2 .BYTE-Anweisung

Die .BYTE-Anweisung initialisiert Byte-Speicherstellen. In einem einzigen .BYTE-Befehl können Mehrfachargumente – durch Kommas getrennt – definiert werden, um aufeinanderfolgende Speicherstellen zu laden. Gültig sind ASCII-Ketten oder -Ausdrücke, die einen Acht-Bit-Wert beinhalten. ASCII-Ketten in .BYTE-Anweisungen dürfen nicht mehr als 20 Zeichen zwischen zwei Anführungszeichen erzeugen.

Beispiel:

```
==004C ASCII
4142    .BYT 'ABCD','EFGH','JOE''S'
4344
4546
4748
4A4F
452753
4C      .BYT <ASCII,>ASCII+2-%1,<*+>*,2
01
5B
==005C
02
5349    .BYT 'SIEMENS PC 100 ASSEM','BLER'
454D
454E
5320
5043
2031
3030
2041
5353
454D
==0071
424C
4552
```

Anmerkung:
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 sinnvoll bei Sprungtabellen-Erstellung und bei Zeiger-Initialisierung. Ein Operandausdruck wird als eine Zwei-Byte-Adresse ausgewertet und in der Reihenfolge: LSB, MSB gespeichert.

Wie bei .BYTE sind Mehrfach-Operand-Felder – durch Kommas getrennt – erlaubt.

Beispiel:

```
==0075 SPRGTB
A1E1    .WOR $E1A1,$E907,RDRUB
07E9
5FE9
0200    .WOR $2,<SPRGTB,>SPRGTB,*,@6371
7500
0000
8100
F90C
==0085
```

8.4 .DBYTE-Anweisung

Wenn ein Ausdruckwert von 16 Bits in normaler Reihenfolge (höherwertiges, niederwertiges Byte) erzeugt werden soll, muß die .DBYTE-Anweisung verwendet werden. Die Syntax-Regeln sind die gleichen wie für .WORD.

Beispiel:

```
==0085 DATA15
E1A1    .DBY $E1A1,$E907,RDRUB
E907
E95F
0002    .DBY $2,<DATA15,>DATA15,*,%0101101111101
0085
0000
0091
05BD
==0095
```

Anmerkung:
Sie finden zu diesem Kapitel ein Anwendungsbeispiel im Monitor-Programmlisting ab Zeile 465.

8.5 .PAGE-Anweisung

Die .PAGE-Anweisung verursacht den Druck einer Überschriftszeile unter einer gestrichelten Linie. Ein Titel kann als ASCII-Kette im Operandfeld spezifiziert sein und mit einer Kette von Leerstellen (eine oder mehrere) gelöscht werden. Das Fehlen eines Operanden verursacht ebenfalls die Löschung des Titels. Dieser Befehl wird bei der Eingabe in den Quellcode nicht ausgedruckt, es erscheinen nur die Ergebnisse. Die Eingabe von z.B.

```
.PAG 'ORIGINAL TITEL'
.PAG
.PAG 'NEUER TITEL'
.PAG
```

würde folgenden Ausdruck am oberen Rand jeder Seite verursachen:

Beispiel:

```
--------------------
ORIGINAL TITEL

--------------------

--------------------
NEUER TITEL

--------------------
```

8.6 .SKIP-Anweisung

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

Beispiel:

```
*=$100
      .SKI
CURSOR *=*+2
DATA17 *=*+2
      .SKI
DATA18 =DATA17
```

Das verursacht den Ausdruck:

```
==0095
        *=$100

==0100 CURSOR
        *=*+2
==0102 DATA17
        *=*+2

==0104 DATA18
        =DATA17
```

Es ist jedoch ebenso möglich, die gewünschte Leerzeile durch Einfügen von „SPACE/CR" im Textaufbereitungsprogramm zu erzeugen.

8.7 .OPT-Anweisung

Die sieben Alternativen der .OPT-Anweisung steuern die Ausgabedatei-Erzeugung und die ASCII-Ketten-Erweiterung in .BYTE-Anweisungen. Sie werden wie folgt bestimmt:

.OPT LIST, GENERATE, ERRORS, MEMORY, SYMBOL, COUNT, CROSS REFERENCE OUTPUT

und durch folgende Codierung eliminiert:

.OPT NOLIST, NOGENERATE, NOERRORS, NOMEMORY, NOSYMBOL, NO-COUNT

Da, bis auf besonders gekennzeichnete Kommandos, nur die ersten drei Zeichen jeder Alternative abgetastet werden, dürfen sie wie folgt geschrieben werden:

.OPT LIS, GEN, ERR, MEM, SYM, CNT, COU
.OPT NOL, NOG, NOE, NOM, NOS, NOC

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 Befehle SYM, CNT, COU, NOC, NOS werden vom Assembler zwar gelesen und akzeptiert, jedoch nicht abgearbeitet. Die nachträgliche Implementierung dieser Befehle wird durch folgendes Steuerprogramm ermöglicht:

Steuerprogramm[1]**):**

```
PAGE 01
LINE # LOC  CODE             LINE

0001  0000                    .OPT CNT
0002  0000                  ;**********************************
0003  0000                  ;**********************************
0004  0000                  ;***  PROGRAMM ZUM ERKENNEN DER ***
0005  0000                  ;***    SONDEROPTIONEN           ***
0006  0000                  ;***                             ***
0007  0000                  ;*** AUFRUF DURCH :              ***
0008  0000                  ;*** LIST=Y OUT=U                ***
0009  0000                  ;*** DIE VOM BENUTZER DEFFI-     ***
0010  0000                  ;*** NIERTE AUSGABEROUTINE RUFT ***
0011  0000                  ;*** RUFT DIE ABFRAGEROUTINE MIT***
0012  0000                  ;*** 'JSR EXTOPT' AUF.           ***
0013  0000                  ;**********************************
0014  0000                  ;**********************************

0015  0000                    .OPT SVM
0016  0000                    .OPT COU

0017  0000                  ;*** REGISTER & MONITOR-ROUTINEN ***

0018  0000                  IFLGS  =$FE

0019  0000                  ; DIESES REGISTER MUSS ABGEFRAGT WERDEN UM
0020  0000                  ; DIE SONDER OPTIONEN ZU ERKENNEN
0021  0000                  ; BEDEUTUNG:
0022  0000                  ; BIT 0 GESETZT NACH SVM-KOMMANDO
0023  0000                  ;       RUECKGESETZT NACH NOS-KOMMANDO
0024  0000                  ; BIT 1 GESETZT NACH CNT-KOMMANDO
0025  0000                  ;       RUECKGESETZT NACH NOC-KOMMANDO
0026  0000                  ; BIT 2 GESETZT NACH COU-KOMMANDO
0027  0000                  ; BIT 7 GESETZT NACH END-KOMMANDO
0028  0000                  ; --- BIT 7 KANN VERWENDET WERDEN,UM SICHERZUSTELLEN,
0029  0000                  ; --- DASS DAS SVM- ODER COU-KOMMANDO ERST NACH DER
0030  0000                  ; --- ASSEMBLIERUNG DES QUELLTEXTES ABGEARBEITET WIRD!


0031  0000                  OPRTBL =$0D
0032  0000                  SPC    =$20
0033  0000                  ISVM   =$2A
0034  0000                  OPSRCH =$D9EA
0035  0000                  OPTDRE =$DDB1
0036  0000                  PHXY   =$EB9E
0037  0000                  PLXY   =$EBAC


0038  0000                        *=$F90          ; PROGRAMM STARTADDRESSE

0039  0F90  20 9E EB    EXTOPT JSR PHXY
0040  0F93  A5 2D              LDA ISVM+3      ; DRITTES ZEICHEN IM BUFFER...
0041  0F95  C9 20              CMP #SPC        ; MUSS EIN (SPC) SEIN
0042  0F97  D0 17              BNE NOEXC
0043  0F99  A5 0D              LDA OPRTBL      ; RETTEN DER ASSEMBLER-REGISTER
0044  0F9B  48                 PHA
0045  0F9C  A5 0E              LDA OPRTBL+1
0046  0F9E  48                 PHA
0047  0F9F  A9 B1              LDA #<OPTDRE    ; INITIALISIEREN DER SUCHROUTINE
0048  0FA1  A0 DD              LDY #>OPTDRE
0049  0FA3  A2 14              LDX #$14
0050  0FA5  20 EA D9           JSR OPSRCH      ; ROUTINE,DIE OPTIONEN SUCHT
0051  0FA8  68                 PLA             ; ASSEMBLER-REGISTER ERSETZEN
0052  0FA9  85 0E              STA OPRTBL+1
```

Steuerprogramm[1]) **(Fortsetzung):**

```
PAGE 02
LINE # LOC  CODE             LINE

0053  0FAB  68                PLA
0054  0FAC  85 0D             STA OPRTBL
0055  0FAE  B0 04             BCS M00          ; C=0 DANN KOMMANDO GUELTIG

0056  0FB0  20 AC EB   NOEXC  JSR PLXV         ; RUECKSRUNG IN DAS AUFRUFENDE
0057  0FB3  60                RTS              ; ... AUSGABEPROGRAMM

0058  0FB4  E0 05       M00   CPX #5           ; (.END) FLAG SETZEN ?
0059  0FB6  D0 06             BNE M02          ; ...DAMIT DAS (SYM) ODER (COU)
0060  0FB8  A9 80             LDA #%10000000   ; ...KOMMANDO ERST NACH DER
0061  0FBA  05 FE             ORA IFLGS        ; ASSEMBLIERUNG ZUR AUSFUEHRUNG
0062  0FBC  D0 37             BNE H01          ;  ...GELANGT

0063  0FBE  8A          M02   TXA              ; BERECHNUNG DER ZIELADDRESSE
0064  0FBF  38                SEC
0065  0FC0  E9 0A             SBC #10          ; DIE SONDEROPTIONEN STEHEN IM
0066  0FC2  30 EC             BMI NOEXC        ; ASSEMBLER-ROM ZWISCHEN
0067  0FC4  C9 05             CMP #5           ; $0A UND $0E
0068  0FC6  B0 E8             BCS NOEXC
0069  0FC8  0A                ASL A
0070  0FC9  AA                TAX
0071  0FCA  BD D6 0F          LDA JUMP+1,X     ; ABSPEICHERN DER ZIELADDRESSE
0072  0FCD  48                PHA
0073  0FCE  BD D5 0F          LDA JUMP,X
0074  0FD1  48                PHA
0075  0FD2  A5 FE             LDA IFLGS
0076  0FD4  60                RTS              ; AUSFUEHRUNG DES KOMMANDOS

0077  0FD5             ;------- LISTE DER ZIELADDRESSEN -------

0078  0FD5  DE 0F      JUMP   .WOR SYMU-1
0079  0FD7  E4 0F             .WOR NOSYM-1
0080  0FD9  E9 0F             .WOR NOCU-1
0081  0FDB  EE 0F             .WOR CNTU-1
0082  0FDD  F2 0F             .WOR COUU-1

0083  0FDF             ;------- ROUTINE,DIE DAS OPT.-FLAG SETZT -------

0084  0FDF  09 01      SYMU   ORA #%00000001 ; SETZEN DES (SYM) FLAGS -
0085  0FE1  29 7F             AND #%01111111 ; ...& RUECKSETZEN DES (END) FLAGS
0086  0FE3  D0 10             BNE H01
0087  0FE5  29 FE      NOSYM  AND #%11111110
0088  0FE7  4C F5 0F          JMP H01
0089  0FEA  29 FD      NOCU   AND #%11111101
0090  0FEC  4C F5 0F          JMP H01
0091  0FEF  09 02      CNTU   ORA #%00000010
0092  0FF1  D0 02             BNE H01
0093  0FF3  09 04      COUU   ORA #%00000100
0094  0FF5  85 FE      H01    STA IFLGS
0095  0FF7  4C B0 0F          JMP NOEXC
0096  0FFA                    .END EXTOPT
```

[1]) Das Steuerprogramm ist so ausgelegt, daß hinter der .OPT-Anweisung nur die drei Kennbuchstaben der Sonderoptionen stehen dürfen. Diese dürfen auch nur die letzten, bzw. die einzigen Buchstaben in der aufrufenden .OPT-Anweisung sein. Die Verwendung einer Sonderoption als Kennsatz ist nicht erlaubt.

```
SYMBOL CROSS REFERENCE

SYMBOL DEFINED  REFERENCES


CNTU    0091   0081
COUU    0093   0082
EXTOPT  0039   0096
H01     0094   0062  0086  0088  0090  0092
IFLGS   0018   0061  0075  0094
ISYM    0033   0040
JUMP    0078   0071  0073
M00     0058   0055
M02     0063   0059
NOCU    0089   0080
NOEXC   0056   0042  0066  0068  0095
NOSYM   0087   0079
OPRTBL  0031   0043  0045  0052  0054
OPSRCH  0034   0050
OPTDRE  0035   0047  0048
PHXV    0036   0039
PLXV    0037   0056
SPC     0032   0041
SYMU    0084   0078

  SYMBOL TABLE

 SYMBOL  VALUE

 CNTU    0FEF   COUU    0FF3   EXTOPT  0F90   H01     0FF5   IFLGS   00FE
 ISYM    002A   JUMP    0FD5   M00     0FB4   M02     0FBE   NOCU    0FEA
 NOEXC   0FB0   NOSYM   0FE5   OPRTBL  000D   OPSRCH  D9EA   OPTDRE  DDB1
 PHXV    EB9E   PLXV    EBAC   SPC     0020   SYMU    0FDF
      .END EXTOPT
ERRORS= 0000
```

Die sieben Alternativen haben folgende Funktionen:

8.7.1 LIST (NOLIST)

steuert die Erzeugung des Programmprotokollausdrucks (Programm-Listing), der assemblierte Quelleingaben, erzeugten Maschinencode, Fehler und Warnungen enthält.

8.7.2 GENERATE (NOGENERATE)

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

8.7.3 ERRORS (NOERRORS)

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

8.7.4 MEMORY (NOMEMORY)

steuert die Ausgabe des Objekt-Codes in den Speicher. Der Ausgabebaustein, der vom Benutzer auf die Frage OBJ-OUT= bestimmt wurde, ist voreingestellt! Stößt der Assembler auf die Anweisung MEM so wird das Objekt-Modul im RAM-Speicherbereich abgelegt. Mit der Anweisung NOM wird die Ausgabe wieder auf den voreingestellten Ausgabebaustein zurückgelegt.

Anmerkung:
Die Benutzung dieses Befehls ist vor allem dann sinnvoll, wenn man den Assemblerablauf beim Assemblieren in den Speicher nicht zerstören will, indem man Daten oder Programmteile in der „ZERO PAGE" ablegt. Diese Daten können, nach dem Assemblerlauf, von dem betreffenden Eingabebaustein nachgeladen werden.

8.7.5 SYMBOL, COUNT, CROSS REFERENCE OUTPUT (NOSYMBOL, NOCOUNT)

Da aus Platzgründen diese Befehle nicht im Assembler-ROM enthalten sind, läßt sich jede beliebige Steuerroutine einfügen. Es ist jedoch empfehlenswert, den Befehl SYM zur Erzeugung einer Symboltabelle, CNT zur Steuerung einer Zeilennummer- bzw. Objekt Code-Adressenausgabe und COU zum Anlegen einer „Cross Reference Map" zu benutzen. Der Pointer auf die momentan aktive Objekt Code-Adresse befindet sich in den Zellen $32 (niederwertiges Byte) und $33 (höherwertiges Byte).

8.8 .FILE-Anweisung

Bei großen Programmen ist es im Normalfall bequemer, wenn man das Quellprogramm in logische Segmente aufteilt, die getrennt in den Textaufbereitungs-Programmpuffer geladen und aufbereitet werden. Nach der Aufbereitung wird jede Datei vom Textpuffer in eine getrennte Datei auf der Kassette gespeichert. .Soll 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 .FI-Anweisung, die zu der nächsten Datei in der Kette zeigt.

Beispiel:

```
.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.

Beispiel:

```
.END
```

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

Anmerkung:
Nach .END ist es übersichtlicher den Namen der letzten Datei nochmals in den Quellentext mitaufzunehmen. Das ist mit der .END NAME-Anweisung ohne Schwierigkeiten zu realisieren, da der Text, der nach der .END-Anweisung steht, vom Assembler nicht mehr verarbeitet wird.

9. Bemerkungen (Kommentare)

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

Beispiel:

```
A900   LDA #0
;KOMMENTAR NACH EINEM SEMIKOLON
A900   LDA #0 KOMMENTAR OHNE SEMIKOLON
```

Anmerkung:

- *Der Assembler-Lauf läßt sich zu jedem beliebigen Zeitpunkt durch Betätigen der Taste SPACE anhalten, sowie durch erneutes Drücken fortsetzen. Betätigen Sie während der Assemblierung die Taste ESC, so erfolgt ein Rücksprung in das Monitor-Programm.*
- *Beachten Sie, daß die Index-Register bei der vom Benutzer definierten Ein- oder Ausgabe nicht überschrieben werden dürfen bzw. zwischengespeichert werden müssen. Sie zerstören sonst die Pointer auf den nächsten abzuarbeitenden Buchstaben.*
- *Es ist möglich, daß der Assembler beim Erreichen einer bestimmten Datenmenge die Objekt Code-Ausgabe auf Band nicht korrekt abschließt. Man bemerkt diesen Fehler nicht beim Verifizieren sondern ausschließlich beim Einlesen des Objekt-Moduls in den PC 100. Der Einlesevorgang wird in diesem Fall nicht mehr abgeschlossen, und man muß den Wiedereintritt in den Monitor durch Betätigen der Tasten ‚E' und ‚R' erzwingen, wenn der letzte Datenblock (80 Bytes) eingelesen wurde. Das Programm wird trotzdem vollständig eingelesen. Sollten Sie diesen Fehler bemerken, fügen Sie am Ende des Programms einige NOP-Befehle ein. Der Block wird dann wie gewünscht abgeschlossen.*
- *Beachten Sie auch, daß beim Assemblieren auf Tonbandkassette das gewünschte Bandlaufwerk, vor dem Aufruf des Assemblers, eingeschaltet ist. Ist dies nicht der Fall, wird der erste Bandblock nicht mit aufgezeichnet.*
- *Stellen Sie außerdem die Anzahl der Synchroncharakter (Speicherzelle $A4Ø9) auf den, für Ihren Kassettenrekorder, günstigsten Wert ein. Dieser ist in der Regel etwas größer, als der Wert von $Ø8.*

10. Mikroprozessor-Befehle

56 gültige Maschinenbefehle stellen die Operationen dar, die mit dem Mikroprozessoren durchgeführt werden.

10.1 Zeichenvorrat

Folgende Zeichen werden benutzt:

Zeichen	Bedeutung
A	Akkumulator (Akku)
X,Y	Index Register
M	Speicherzelle
P	Prozessorstatus-Register
S	Stapelzeiger
V	Veränderung
–	Keine Veränderung
+	Addition
∧	Logisch „UND"
-	Subtraktion
∀	Logisch „EXCLUSIV ODER"
↑	Transferiere vom Stapelspeicher
↓	Transferiere auf den Stapelspeicher
→	Transferiere in
←	Transferiere in
∨	Logisch „ODER"
PC	Programmzähler
PCH	Programmzähler höherwertiges Byte (MSB)
PCL	Programmzähler niederwertiges Byte (LSB)
OPER	Operand
#	Direkte Adressierung (Immediate Adressierung)

10.2 Befehlsliste

Auf den folgenden Seiten werden alle 56 gültigen Mikroprozessor-Befehle in alphabetischer Reihenfolge, mit Angabe der erzeugten Objekt-Codes (OP-Code), der Anzahl der Abarbeitungszyklen (Zyklen) und des Speicherplatzbedarfs (Bytes) abgehandelt.

Am Kapitelende auf Seite 83 finden Sie nochmals alle Befehle in übersichtlicher Tabellenform.

ADC Addiere Speicherzelle und Carry-Flag zum Akku-Inhalt

Operation $A + M + C \rightarrow A, C$

Flags

N	Z	C	I	D	V
V	V	V	–	–	V

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	ADC # Oper	69	2	2
Zero Page	ADC # Oper	65	2	3
Zero Page, X	ADC # Oper, X	75	2	4
Absolute	ADC # Oper	6D	3	4
Absolute, X	ADC # Oper, X	7D	3	4[1])
Absolute, Y	ADC # Oper, Y	79	3	4[1])
(Indirect, X)	ADC # (Oper, X)	61	2	6
(Indirect), Y	ADC # (Oper), Y	71	2	5[1])

AND Logische „UND"-Verknüpfung von Speicherzelle mit Akku

Operation $A \wedge M \rightarrow A$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	AND # Oper	29	2	2
Zero Page	AND # Oper	25	2	3
Zero Page, X	AND # Oper, X	35	2	4
Absolute	AND # Oper	2D	3	4
Absolute, X	AND # Oper, X	3D	3	4[1])
Absolute, Y	AND # Oper, Y	39	3	4[1])
(Indirect, X)	AND # (Oper, X)	21	2	6
(Indirect), Y	AND # (Oper), Y	31	2	5[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

ASL Schiebe Speicherzelle oder Akku um ein Bit nach links

Operation C←|7|6|5|4|3|2|1|Ø|←Ø

Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Accumulator	ASL A	ØA	1	2
Zero Page	ASL Oper	Ø6	2	5
Zero Page, X	ASL Oper, X	16	2	6
Absolute	ASL Oper	ØE	3	6
Absolute, X	ASL Oper, X	1E	3	7

BCC Relativer Sprung, wenn Carry-Flag=Ø

Operation Springe, wenn C=Ø

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BCC Oper	9Ø	2	2[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn innerhalb einer Seite gesprungen wird und erhöht sich um zwei, wenn eine Seitengrenze übersprungen wird.

BCS Relativer Sprung, wenn Carry-Flag=1

Operation Springe, wenn C=1

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BCS Oper	BØ	2	2[1])

BEQ Relativer Sprung, wenn Zero-Flag=1

Operation Springe, wenn Z=1

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BEQ Oper	FØ	2	2[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn innerhalb einer Seite gesprungen wird und erhöht sich um zwei, wenn eine Seitengrenze übersprungen wird.

BIT Vergleiche die Bits einer Speicherzelle mit Akku

Operation $A \wedge \wedge M, M_7 \rightarrow N,\ M_6 \rightarrow V$

Flags

N	Z	C	I	D	V
M_7	V	–	–	–	M_6

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Zero Page	BIT Oper	24	2	3
Absolute	BIT Oper	2C	3	4

Bit 6 und 7 werden in das Prozessorstatusregister transferiert.
Wenn das Ergebnis der Operation $A \wedge M = Ø$, wird $Z = 1$, sonst ist $Z = Ø$.

BMI Relativer Sprung, wenn Negativ-Flag = 1

Operation Springe, wenn $N = 1$

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BMI Oper	3Ø	2	2[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn innerhalb einer Seite gesprungen wird und erhöht sich um zwei, wenn eine Seitengrenze übersprungen wird.

BNE Relativer Sprung, wenn Zero-Flag=Ø

Operation Springe, wenn Z=Ø

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BNE Oper	DØ	2	2[1])

BPL Relativer Sprung, wenn Negativ-Flag=Ø

Operation Springe, wenn N=Ø

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BPL Oper	1Ø	2	2[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn innerhalb einer Seite gesprungen wird und erhöht sich um zwei, wenn eine Seitengrenze übersprungen wird.

BRK Programm-Unterbrechung

Operation Erzwungener Interrupt PC+2↓P↓

Flags

N	Z	C	I	D	V
–	–	–	1	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	BRK	ØØ	1	7

Das BRK Kommando läßt sich nicht durch Setzen des Interrupt Flags maskieren.

BVC Relativer Sprung, wenn Overflow-Flag=Ø

Operation Springe, wenn V=Ø

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BVC Oper	5Ø	2	2[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn innerhalb einer Seite gesprungen wird und erhöht sich um zwei, wenn eine Seitengrenze übersprungen wird.

BVS Relativer Sprung, wenn Overflow-Flag=1

Operation Springe, wenn V=1

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Relative	BVS Oper	7Ø	2	2[1])

CLC Lösche das Carry-Flag

Operation Ø→C

Flags

N	Z	C	I	D	V
–	–	Ø	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	CLC	18	1	2

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn innerhalb einer Seite gesprungen wird und erhöht sich um zwei, wenn eine Seitengrenze übersprungen wird.

CLD Lösche Dezimalmodus

Operation Ø → D

Flags

N	Z	C	I	D	V
–	–	–	–	Ø	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	CLD	D8	1	2

CLI Lösche Interrupt Sperr-Flag

Operation Ø → I

Flags

N	Z	C	I	D	V
–	–	–	Ø	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	CLI	58	1	2

CLV Lösche Overflow-Flag

Operation $\emptyset \rightarrow V$

Flags

N	Z	C	I	D	V
–	–	–	–	–	Ø

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	CLV	B8	1	2

CMP Vergleiche Speicherzelle mit Akku

Operation A – M

Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	CMP # Oper	C9	2	2
Zero Page	CMP # Oper	C5	2	3
Zero Page, X	CMP # Oper, X	D5	2	4
Absolute	CMP # Oper	CD	3	4
Absolute, X	CMP # Oper, X	DD	3	4[1])
Absolute, Y	CMP # Oper, Y	D9	3	4[1])
(Indirect, X)	CMP # (Oper, X)	C1	2	6
(Indirect), Y	CMP # (Oper), Y	D1	2	5[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

CPX Vergleiche Speicherzelle mit X-Register

Operation X – M

Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	CPX # Oper	EØ	2	2
Zero Page	CPX # Oper	E4	2	3
Absolute	CPX # Oper	EC	3	4

CPY Vergleiche Speicherzelle mit Y-Register

Operation Y – M

Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	CPY # Oper	CØ	2	2
Zero Page	CPY # Oper	C4	2	3
Absolute	CPY # Oper	CC	3	4

DEC Erniedrige Speicherzelle um eins

Operation $M-1 \rightarrow M$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Zero Page	DEC Oper	C6	2	5
Zero Page, X	DEC Oper, X	D6	2	6
Absolute	DEC Oper	CE	3	6
Absolute, X	DEC Oper, X	DE	3	7

DEX Erniedrige X-Register um eins

Operation $X-1 \rightarrow X$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	DEX	CA	1	2

DEY Erniedrige Y-Register um eins

Operation $Y-1 \rightarrow Y$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	DEY	88	1	2

EOR Logische „Exclusiv-ODER"-Verknüpfung von Speicherzelle mit Akku

Operation $A \veebar M \rightarrow A$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	EOR # Oper	49	2	2
Zero Page	EOR # Oper	45	2	3
Zero Page, X	EOR # Oper, X	55	2	4
Absolute	EOR # Oper	4D	3	4
Absolute, X	EOR # Oper, X	5D	3	4[1])
Absolute, Y	EOR # Oper, Y	59	3	4[1])
(Indirect, X)	EOR # (Oper, X)	41	2	6
(Indirect), Y	EOR # (Oper), Y	51	2	5[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

INC Erhöhe Speicherzelle um eins

Operation $M+1 \rightarrow M$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Zero Page	INC Oper	E6	2	5
Zero Page, X	INC Oper, X	F6	2	6
Absolute	INC Oper	EE	3	6
Absolute, X	INC Oper, X	FE	3	7

INX Erhöhe X-Register um eins

Operation $X+1 \rightarrow X$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	INX	E8	1	2

INY Erhöhe Y-Register um eins

Operation $Y+1 \rightarrow Y$

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	INY	C8	1	2

JMP Springe in anderen Speicherbereich

Operation $(PC+1) \rightarrow PCL$
$(PC+2) \rightarrow PCH$

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

JSR Springe in anderen Speicherbereich und rette Rückkehradresse

Operation PC+2↓, (PC+1)→PCL
(PC+2)→PCH

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Absolute	JSR Oper	20	3	6

LDA Lade Akku mit Speicherzelle

Operation M→A

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	LDA # Oper	A9	2	2
Zero Page	LDA # Oper	A5	2	3
Zero Page, X	LDA # Oper, X	B5	2	4
Absolute	LDA # Oper	AD	3	4
Absolute, X	LDA # Oper, X	BD	3	4[1])
Absolute, Y	LDA # Oper, Y	B9	3	4[1])
(Indirect, X)	LDA # (Oper, X)	A1	2	6
(Indirect), Y	LDA # (Oper), Y	B1	2	5[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

LDX Lade X-Register mit Speicherzelle

Operation M → X

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	LDX # Oper	A2	2	2
Zero Page	LDX # Oper	A6	2	3
Zero Page, Y	LDX # Oper, Y	B6	2	4
Absolute	LDX # Oper	AE	3	4
Absolute, Y	LDX # Oper, Y	BE	3	4[1])

LDY Lade Y-Register mit Speicherzelle

Operation M → Y

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	LDY # Oper	AØ	2	2
Zero Page	LDY # Oper	A4	2	3
Zero Page, X	LDY # Oper, X	B4	2	4
Absolute	LDY # Oper	AC	3	4
Absolute, X	LDY # Oper, X	BC	3	4[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

LSR Schiebe Speicherzelle oder Akku um ein Bit nach rechts

Operation Ø→ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø→C

Flags

N	Z	C	I	D	V
Ø	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page, X	LSR Oper, X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute, X	LSR Oper, X	5E	3	7

NOP Keine Operation

Operation No Operation (2 cycles)

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	NOP	EA	1	2

ORA Logische „ODER"-Verknüpfung von Speicherzelle mit Akku

Operation AVM→A

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	ORA # Oper	Ø9	2	2
Zero Page	ORA # Oper	Ø5	2	3
Zero Page, X	ORA # Oper, X	15	2	4
Absolute	ORA # Oper	ØD	3	4
Absolute, X	ORA # Oper, X	1D	3	4[1])
Absolute, Y	ORA # Oper, Y	19	3	4[1])
(Indirect, X)	ORA # (Oper, X)	Ø1	2	6
(Indirect), Y	ORA # (Oper), Y	11	2	5[1])

PHA Speichere Akku auf Stapelspeicher

Operation A↓

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	PHA	48	1	3

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

PHP Speichere Prozessorstatus-Register auf Stapelspeicher

Operation P↓

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	PHP	Ø8	1	3

PLA Lade Akku mit Inhalt Stapelspeicher

Operation A↑

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	PLA	68	1	4

PLP Lade Prozessorstatus-Register mit Inhalt Stapelspeicher

Operation P↑

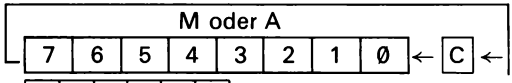
Flags

N	Z	C	I	D	V
*	*	*	*	*	*

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	PLP	28	1	4

ROL Rotiere Speicherzelle oder Akku um ein Bit nach links

Operation



Flags

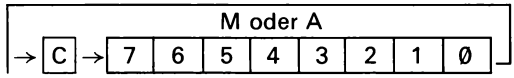
N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Accumulator	ROL A	2A	1	2
Zero Page	ROL Oper	26	2	5
Zero Page, X	ROL Oper, X	36	2	6
Absolute	ROL Oper	2E	3	6
Absolute, X	ROL Oper, X	3E	3	7

*) Vom Stapelspeicher.

ROR Rotiere Speicherzelle oder Akku um ein Bit nach rechts

Operation



Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Accumulator	ROR A	6A	1	2
Zero Page	ROR Oper	66	2	5
Zero Page, X	ROR Oper, X	76	2	6
Absolute	ROR Oper	6E	3	6
Absolute, X	ROR Oper, X	7E	3	7

RTI Kehre aus Interrupt zurück

Operation P↑PC↑

Flags

N	Z	C	I	D	V
*	*	*	*	*	*

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	RTI	4Ø	1	6

*) Vom Stapelspeicher.

RTS Kehre aus Unterprogramm zurück

Operation PC↑, PC+1→PC

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	RTS	6Ø	1	6

SBC Subtrahiere Speicherzelle und negiertes Carry-Flag vom Akku

Operation $A-M-\bar{C}\rightarrow A$

Flags

N	Z	C	I	D	V
V	V	V	–	–	V

$\bar{C}$ = negativer Übertrag

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Immediate	SBC # Oper	E9	2	2
Zero Page	SBC # Oper	E5	2	3
Zero Page, X	SBC # Oper, X	F5	2	4
Absolute	SBC # Oper	ED	3	4
Absolute, X	SBC # Oper, X	FD	3	4[1])
Absolute, Y	SBC # Oper, Y	F9	3	4[1])
(Indirect, X)	SBC # (Oper, X)	E1	2	6
(Indirect), Y	SBC # (Oper), Y	F1	2	5[1])

[1]) Die Anzahl der Ausführungszyklen erhöht sich um eins, wenn sich die Seitengrenzen überschneiden.

SEC Setze Carry-Flag

Operation 1 → C

Flags

N	Z	C	I	D	V
–	–	1	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	SEC	38	1	2

SED Setze Dezimalmodus

Operation 1 → D

Flags

N	Z	C	I	D	V
–	–	–	–	1	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	SED	F8	1	2

SEI Setze Interrupt Sperr-Flag

Operation $1 \rightarrow I$

Flags

N	Z	C	I	D	V
–	–	–	1	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	SEI	78	1	2

STA Speichere Akku in Speicherzelle

Operation $A \rightarrow M$

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Zero Page	STA Oper	85	2	3
Zero Page, X	STA Oper, X	95	2	4
Absolute	STA Oper	8D	3	4
Absolute, X	STA Oper, X	9D	3	5
Absolute, Y	STA Oper, Y	99	3	5
(Indirect, X)	STA (Oper, X)	81	2	6
(Indirect), Y	STA (Oper), Y	91	2	6

STX Speichere X-Register in Speicherzelle

Operation X → M

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Zero Page	STX Oper	86	2	3
Zero Page, Y	STX Oper, Y	96	2	4
Absolute	STX Oper	8E	3	4

STY Speichere Y-Register in Speicherzelle

Operation Y → M

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Zero Page	STY Oper	84	2	3
Zero Page, X	STY Oper, X	94	2	4
Absolute	STY Oper	8C	3	4

TAX Transferiere Akku in X-Register

Operation A→X

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	TAX	AA	1	2

TAY Transferiere Akku in Y-Register

Operation A→Y

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	TAY	A8	1	2

TYA Transferiere Y-Register in Akku

Operation Y→A

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	TYA	98	1	2

TSX Transferiere Adresse des Stapelzeigers in X-Register

Operation S→X

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied[1])	TSX	BA	1	2

[1]) Das höherwertige Byte ist bei diesen Operationen immer eins, da sich der Stapelzeiger stets zwischen $1Ø0 und $1FF befindet.

TXA Transferiere X-Register in Akku

Operation X→A

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied	TXA	8A	1	2

TXS Setze Adresse des Stapelzeigers auf den Wert des X-Registers

Operation X→S

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes	Anzahl Zyklen
Implied [1])	TXS	9A	1	2

[1]) Das höherwertige Byte ist bei diesen Operationen immer eins, da sich der Stapelzeiger stets zwischen $100 und $1FF befindet.

10.3 Sonderbefehle

Der Mikroprozessor kennt eine Anzahl von Sonderbefehlen, die weitgehend unbekannt sind, dem Benutzer aber eine wertvolle Hilfe bei der Programmerstellung sein können. Die gewählte Mnemonic in folgenden Tabellen ist lediglich eine Empfehlung die Befehle sinnvoll zu umschreiben.

Anmerkung:
Diese Befehle sind nicht Bestandteil der Spezifikation, sie können jederzeit ohne Mitteilung geändert werden. Die Sonderbefehle können nicht vom Assemblerprogramm entschlüsselt werden und müssen mit der .BYT-Anweisung programmiert werden (siehe Kapitel 10.2).

AAX Logische „UND"-Verknüpfung von Akku mit X-Register und Ergebnis-Abspeicherung

Operation A ∧ X → M bei Zero Page und (A) ∧ X ∧ $Ø2 → M bei Absolute

Flags

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes
Zero Page	AAX Oper	87	2
Zero Page, Y	AAX+16 Oper	97	2
X-Reg. ∧ $Ø2		9E	3
Absolute	AAX+23 Oper		
X-Reg. ∧ Accu ∧ $Ø2		9F	3
Absolute	AAX+24 Oper		

DCM Erniedrige Speicherzelle um eins und vergleiche Ergebnis mit Akku

Operation M – 1 → M und A – M

Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes
Zero Page	DCM Oper	C7	2

LAX Lade Akku und X-Register

Operation M→A und M→X

Flags

N	Z	C	I	D	V
V	V	–	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes
Immediate	LAX Oper	AB	2
Zero Page	LAX–4 Oper	A7	2

ISB Erhöhe Speicherzelle um eins und subtrahiere Ergebnisse vom Akku

Operation M+1→M und A–M→A

Flags

N	Z	C	I	D	V
V	V	V	–	–	–

Adressierungsart	Assembler-Mnemonic	OP-Code	Anzahl Bytes
Zero Page	ISB Oper	E7	2

Anmerkung:
Die Befehle LAX Immediate, AAX X-Reg. $Ø2 und AAX X-Reg. Accu $Ø2 werden nicht immer korrekt abgearbeitet.

10.4 Programmieren in Assembler-Sprache

Da die vorgenannten Sonderbefehle vom Assembler-Programm nicht entschlüsselt werden können, müssen sie mit Hilfe der .BYT-Anweisung programmiert werden.

Beispiel:

```
==0108 LAX
       =$AB
==0108 AAX
       =$87
==0108 ISB
       =$E7
==0108 DCM
       =$C7
A958   LDA #88
==010A EXTRA
97     .BYT AAX+16,$40
40
F0FB   BEQ *-3
E7     .BYT ISB,30
1E
A7     .BYT LAX-4,%00000111
07
C7     .BYT DCM,0
00
30F4   BMI EXTRA
       .END
 ERRORS= 0000
```

10.5 Mikroprozessor-Befehlssatz (Übersicht)

INSTRUCTIONS		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM.			IMPLIED			(IND. X)			(IND), Y			Z. PAGE, X			ABS, X			ABS, Y			RELATIVE			INDIRECT			Z. PAGE, Y			PROCESSOR STATUS CODES								
MNE-MONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	7 N	6 V	5 .	4 B	3 D	2 I	1 Z	0 C	MNE-MONIC
ADC	A + M + C → A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										N	V					Z	C	ADC
AND	A ∧ M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										N						Z	.	AND
ASL	C ← [7 … 0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													N						Z	C	ASL
BCC	BRANCH ON C = 0 (2)																															90	2	2								.								BCC
BCS	BRANCH ON C = 1 (2)																															B0	2	2								.	.	.	.	.	.	.	.	BCS
BEQ	BRANCH ON Z = 1 (2)																															F0	2	2								.	.					.		BEQ
BIT	A ∧ M				2C	4	3	24	3	2																																M_7	M_6	.				Z	.	BIT
BMI	BRANCH ON N = 1 (2)																															30	2	2								.								BMI
BNE	BRANCH ON Z = 0 (2)																															D0	2	2								.								BNE
BPL	BRANCH ON N = 0 (2)																															10	2	2								.	.	.	.	.	.	.	.	BPL
BRK	BREAK													00	7	1																										.				1		1			BRK
BVC	BRANCH ON V = 0 (2)																															50	2	2								.								BVC
BVS	BRANCH ON V = 1 (2)																															70	2	2								.							.	BVS
CLC	0 → C													18	2	1																										.							0	CLC
CLD	0 → D													D8	2	1																										.	.	.	.	.	0	.	.	.	CLD
CLI	0 → I													58	2	1																										.	.					0	.		CLI
CLV	0 → V													B8	2	1																										.	0						.	.	CLV
CMP	A — M	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										N						Z	C	CMP
CPX	X — M	E0	2	2	EC	4	3	E4	3	2																																N						Z	C	CPX
CPY	Y — M	C0	2	2	CC	4	3	C4	3	2																																N	.	.	.	.	.	Z	C	CPY
DEC	M — 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													N						Z	.	DEC
DEX	X — 1 → X													CA	2	1																										N						Z	.	DEX
DEY	Y — 1 → Y													88	2	1																										N						Z	.	DEY
EOR	A ⊻ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										N						Z	.	EOR
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													N	.	.	.	.	.	Z	.	INC
INX	X + 1 → X													E8	2	1																										N						Z	.	INX
INY	Y + 1 → Y													C8	2	1																										N	.					Z	.	INY
JMP	JUMP TO NEW LOC				4C	3	3																													6C	5	3				.								JMP
JSR	JUMP SUB				20	6	3																																			.							.	JSR
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										N	.	.	.	.	.	Z	.	LDA

10.5 Mikroprozessor-Befehlssatz (Fortsetzung)

INSTRUCTIONS		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM.			IMPLIED			(IND. X)			(IND), Y			Z. PAGE, X			ABS, X			ABS, Y			RELATIVE			INDIRECT			Z. PAGE, Y			PROCESSOR STATUS CODES								
MNEMONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	7 N	6 V	5 .	4 B	3 D	2 I	1 Z	0 C	MNEMONIC
LDX	M→X (1)	A2	2	2	AE	4	3	A6	3	2																BE	4	3										B6	4	2	N						Z	.	LDX
LDY	M→Y (1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3												N						Z	.	LDY
LSR	0→[7 0]→C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3												0						Z	C	LSR
NOP	NO OPERATION													EA	2	1																								.						.		NOP
ORA	AVM→A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3									N	.	.	.	.	.	Z	.	ORA
PHA	A→Ms S−1→S													48	3	1																								.						.		PHA
PHP	P→Ms S−1→S													08	3	1																								.						.		PHP
PLA	S+1→S Ms→A													68	4	1																								N	.	.	.	.	.	Z	.	PLA
PLP	S+1→S Ms→P													28	4	1																									(RESTORED)							PLP
ROL	[7 0]←[C]				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3												N	.	.	.	.	.	Z	C	ROL
ROR	[C]→[7 0]				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3												N	.	.	.	.	.	Z	C	ROR
RTI	RTRN INT													40	6	1																									(RESTORED)							RTI
RTS	RTRN SUB													60	6	1																								.	.	.	.	.	.	.	.	RTS
SBC	A−M−C→A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3									N	V					Z	(3)	SBC
SEC	1→C													38	2	1																								.						1		SEC
SED	1→D													F8	2	1																								.	.	.	.	1	.	.	.	SED
SEI	1→I													78	2	1																								.					1			SEI
STA	A→M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3									.								STA
STX	X→M				8E	4	3	86	3	2																													96	4	2	.								STX
STY	Y→M				8C	4	3	84	3	2													94	4	2																.						.		STY
TAX	A→X													AA	2	1																								N	.	.	.	.	.	Z	.	TAX
TAY	A→Y													A8	2	1																								N						Z	.	TAY
TSX	S→X													BA	2	1																								N						Z	.	TSX
TXA	X→A													8A	2	1																								N						Z	.	TXA
TXS	X→S													9A	2	1																								.						.		TXS
TYA	Y→A													98	2	1																								N	.	.	.	.	.	Z	.	TYA

(1) ADD 1 TO "N" IF BOUNDARY IS CROSSED
(2) ADD 1 TO "N" IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO "N" IF BRANCH OCCURS TO DIFFERENT PAGE
(3) CARRY NOT = BORROW
(4) IF IN DECIMAL MODE, Z-FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M_6 MEMORY PER STACK POINTER

\+ ADD
− SUBTRACT
∧ AND
∨ OR
⊻ EXCLUSIVE OR

M_7 MEMORY BIT 7
M_6 MEMORY BIT 6
n NO. CYCLES
NO. BYTES

11. Monitor-Befehle

11.1 Tabelle 1; Befehlsliste

Kategorie	Zeichen	Funktion/Bedeutung
Anzeige/ Verändere Register	*	Verändere Programmzähler
	P	Verändere Prozessorstatus
	A	Verändere Akkumulatorinhalt
	X	Verändere X-Registerinhalt
	Y	Verändere Y-Registerinhalt
	S	Verändere Stapelzeigeradresse
	R	Anzeige der Register
Anzeige/ Verändere Speicher	M	Ausgabe von 4 Speicherzelleninhalten
	SPACE	Ausgabe der nächsten 4 Speicherzellen
	/	Verändere Speicherinhalt
Befehlseingabe/ Disassemblierung	I	Eintritt in den mnemonischen Befehls-eingabemodus
	K	Disassembliere Speicherinhalt
Schnittstelle für Anwender-funktionen	F1	Anwenderfunktions-Taste 1
	F2	Anwenderfunktions-Taste 2
	F3	Anwenderfunktions-Taste 3
Ausführung/ Protokoll	G	Starte Maschinenprogrammausführung ab Programmzähleradresse
	Z	Ein-/Ausschalten des Befehlsprotokollmodus
	V	Ein-/Ausschalten des Registerprotokollmodus
	H	Ausgabe des Programmzählerprotokolls
Manipuliere Breakpoints (Anhaltepunkte)	?	Anzeige der Breakpoints
	#	Löschen aller Breakpoints
	B	Verändern der Breakpoints
	4	Ein-/Ausschalten der Breakpointfreigabe

11.2 RESET – Eingabe und Initialisierung des Monitorprogramms

Der RESET-Befehl führt eine Hardwarerückstellung der Peripherie-Bausteine durch und initialisiert den Monitor.

- Führen Sie einen „warmen" RESET durch, indem Sie die Tasten E/R betätigen.
- Führen Sie einen „kalten" RESET durch, indem Sie die Stromversorgung ausschalten, einige Sekunden warten, um die Stromversorgung dann wieder einzuschalten.
- Sie können einen „kalten RESET" auch durch die Tasten E/R erzeugen, wenn Sie nach jedem vorangegangenen RESET die Speicherzelle $A402 verändert haben. Programmteile werden hierdurch nicht zerstört. Ein TRACE-Betrieb ist jedoch erst wieder nach dem Drücken der Tasten E/R möglich.

Achtung:
Verändern Sie den DILINK-Vektor in Speicherzelle $A406/$A407 *niemals mit dem M-Befehl. Das Monitorprogramm würde sich sofort ‚verlaufen' und wäre dann nicht mehr unter Kontrolle zu bringen. Ein Wiedereintritt ist dann nur noch durch Abschalten der Stromversorgung möglich.*

11.2.1 *-Befehl – Verändere Programmzähler

Der *-Befehl verändert den Wert des Programmzählers. Es handelt sich hierbei um ein 16 Bit Register, das die Adresse des nächsten durchzuführenden Befehls enthält. Bei jeder Verwendung dieses Registers inkrementiert der Prozessor den Programmzähler. Der Prozessor führt also die Befehle der Reihe nach aus, es sei denn, ein Sprung- oder Verzweigungsbefehl setzt definitiv einen neuen Wert in den Programmzähler.

Verwenden Sie den *-Befehl wie folgt:

1. Geben Sie SHIFT und * gleichzeitig ein.

   Beispiel:

   ⟨*⟩= ^

2. Geben Sie den neuen Hexadezimalwert des Programmzählers ein. Beenden Sie die Eingabe mit RETURN oder SPACE.

   Beispiel:

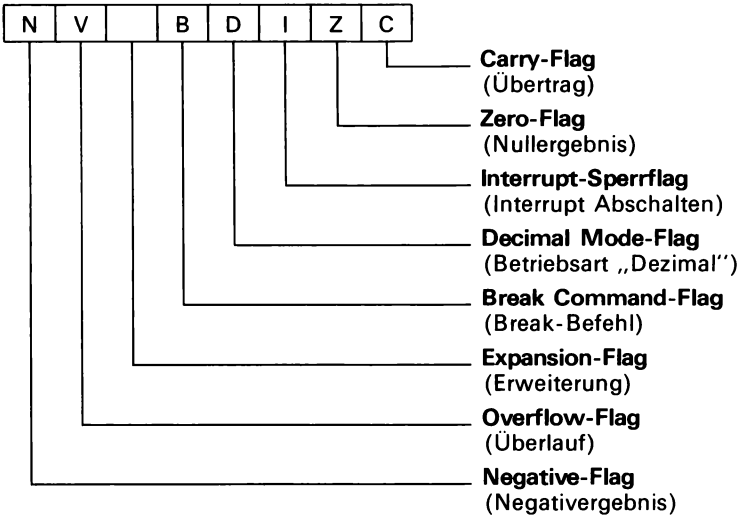
   ⟨*⟩=0300

In dem angeführten Beispiel wurde der Programmzähler in **$0300** geändert. Wenn der G-Befehl eingegeben wird, wird zuerst der Befehl in Speicherstelle **$0300** ausgeführt. Es beginnt die Ausführung bei der Programmadresse.

11.2.2 P-Befehl – Verändere Prozessorzustandsregister

Der P-Befehl verändert den Inhalt des Prozessorzustandsregisters.

11.2.2.1 Prozessorstatusregister



Anmerkung:
Im P-Register haben nur die einzelnen Bits Bedeutung. Sollten Sie diese Bits überprüfen oder verändern wollen, schlagen Sie Kapitel 11.2.2.2, Tabelle 2 auf.

Erläuterungen zum Register

Bit 7	**(N) = 1,** wenn das letzte Ergebnis eine 1 in seinem höchstwertigen Bit hatte, Ø wenn das letzte Ergebnis eine Ø in seinem höchstwertigen Bit hatte. Dieses Bit wird als Negative- oder Sign-Flag bezeichnet (Negativergebnis)
Bit 6	**(V) = 1,** wenn die letzte Rechenoperation einen Zweierkomplement-Überlauf, d.h. einen Übertrag in Bit 7 des Ergebnisses, produziert hatte. Dieses Bit wird als Overflow-Flag bezeichnet (Überlauf)
Bit 5	Expansions-Flag (Erweiterung)
Bit 4	**(B) = 1,** wenn der letzte Befehl BRK war, sonst Ø. Dieses Bit wird als Break Command-Flag bezeichnet (Break-Befehl)
Bit 3	**(D) = 1,** wenn der Prozessor in der Betriebsart „Dezimal" ist, sonst Ø. Dieses Bit wird als Decimal Mode-Flag bezeichnet (Betriebsart „Dezimal")
Bit 2	**(I) = 1,** wenn Interrupts (Programm-Unterbrechungen) nicht erlaubt sind, Ø wenn sie erlaubt sind. Dieses Bit wird als Interrupt-Sperrflag bezeichnet (Interrupt Abschalten)
Bit 1	**(Z) = 1,** wenn das letzte Ergebnis Null war, sonst Ø. Dieses Bit wird als Zero-Flag bezeichnet (Nullergebnis)
Bit Ø	**(C) = 1,** wenn das Ergebnis der letzten Rechenoperation größer als $FF war, sodaß es nicht mehr in einem Byte abgespeichert werden konnte. Dieses Bit wird als Carry-Flag bezeichnet (Übertrag)

11.2.2.2 Tabelle 2, Prozessorstatusanzeige

Anzeige wert	Bedeutung							
	Negativ	Overflow	....	Break	Dezimal	Interrupt	Zero	Carry
Ø								
1				B				C
2							Z	
3				B			Z	C
4		O				I		
5		O		B		I		C
6		O				I	Z	
7		O		B		I	Z	C
8	N				D			
9	N			B	D			C
A	N				D		Z	
B	N			B	D		Z	C
C	N	O			D	I		
D	N	O		B	D	I		C
E	N	O			D	I	Z	
F	N	O		B	D	I	Z	C

Zur Änderung des Prozessorzustandsregisters geben Sie P ein. PC 100 gibt aus:

<P>=Λ

Geben Sie den neuen Wert des Prozessorzustandsregisters in Form einer zweistelligen Hexadezimalzahl ein. Eine führende Null muß in die Stelle der linken Ziffer eingegeben werden, wenn der Wert der linken Ziffer Null ist.

Beispiel:

<P>=ØØ

In dem Beispiel wurde der Wert des Prozessorzustandsregisters zu $ØØ geändert.

11.2.3 A-Befehl – Verändere Akkumulator

Der A-Befehl verändert den Inhalt des Akkumulators.

Es handelt sich hierbei um ein 8-Bit-Register, das den Mittelpunkt der Prozessoroperationen darstellt. Es arbeitet in ähnlicher Weise wie das laufende Zwischenergebnis in einem Rechner.

Geben Sie A ein, um den Inhalt des Akkumulators zu verändern.

Beispiel:

⟨A⟩=^

Geben Sie den neuen Wert des Akkumulators als zweistellige Hexadezimalzahl ein. Eine führende Null muß in die Stelle der linken Ziffer eingegeben werden, wenn der Wert der linken Ziffer Null ist.

Beispiel:

<A>=01

In dem Beispiel wurde der Wert des Akkumulators zu $Ø1 verändert.

11.2.4 Index-Register X und Y

Es handelt sich hierbei um zwei 8-Bit-Register, die als Zähler oder Index verwendet werden können.

11.2.4.1 X-Befehl – Verändere X-Register

Der X-Befehl verändert den Inhalt des X-Registers.

Geben Sie X ein, um den Inhalt des X-Registers zu verändern.

Beispiel:

⟨X⟩=^

Geben Sie den neuen Wert des X-Registers als zweistellige Hexadezimalzahl ein. Eine führende Null muß in die Stelle der linken Ziffer eingegeben werden, wenn der Wert der linken Ziffer Null ist.

Beispiel:

<X>=02

In dem oben angegebenen Beispiel wurde der Wert des X-Registers zu $Ø2 verändert.

11.2.4.2 Y-Befehl – Verändere Y-Register

Der Y-Befehl verändert den Inhalt des Y-Registers.

Geben Sie Y ein, um das Y-Register zu verändern.

Beispiel:

⟨Y⟩=^

Geben Sie den neuen Wert des Y-Registers als zweistellige Hexadezimalzahl ein. Eine führende Null muß an Stelle der linken Ziffer eingegeben werden, wenn deren Wert Null ist.

Beispiel:

⟨Y⟩=03

In dem Beispiel wurde der Wert des Y-Registers zu $Ø3 verändert.

11.2.5 S-Befehl – Verändere Stapelzeiger

Der S-Befehl verändert den Wert des Stapelzeigers.

Es handelt sich hierbei um ein 8-Bit-Register, das die Adresse des Stapels auf Seite 1 des Speichers enthält. Wenn S $F3 enthält, befindet sich die nächst verfügbare Stapelstelle auf Adresse Ø1F3.

Geben Sie S ein, um den Wert des Stapelzeigers zu verändern.

⟨S⟩ =

Geben Sie den neuen Wert des Stapelzeigers als zweistellige Hexadezimalzahl ein. Eine führende Null muß in die Stelle der linken Ziffer eingegeben werden, wenn deren Wert Null ist.

Beispiel:

⟨S⟩=FF

In dem Beispiel wurde der Wert des Stapelzeigers zu $FF verändert. Beachten Sie, daß sich der Stapel immer in Seite 1 des Speichers befindet, so daß die Adresse des Stapels $Ø1FF ist.

11.2.6 R-Befehl – Anzeige der Registerinhalte

Der R-Befehl wird verwendet, um den augenblicklichen Inhalt aller 6 Register anzuzeigen.

Geben Sie R ein, um den Inhalt der Register anzuzeigen. PC 100 wird zwei Zeilen drucken. Die erste Zeile zeigt die Symbole für die Register, die zweite den augenblicklichen Inhalt. Die Register und ihre zugehörigen Symbole sind:

Register	Symbole
Programmzähler	****
Prozessorzustand	PS
Akkumulator	AA
X-Register	XX
Y-Register	YY
Stapelzeiger	SS

Beispiel:

```
<R>
 **** PS AA XX YY SS
 0300 00 01 02 03 FF
```

Die Register und ihre Inhalte sind in dem Beispiel:

Register	Symbole	Inhalt
Programmzähler	(****)	$0300
Prozessorzustand	(PS)	$00
Akkumulator	(AA)	$01
X-Register	(XX)	$02
Y-Register	(YY)	$03
Stapelzeiger	(SS)	$FF[1])

Der R-Befehl bietet auch Spalten-Überschriften als Bezug, wenn das Registerprotokollprogramm oder Breakpoints verwendet werden.

[1]) Das bedeutet, daß der Stapelzeiger auf Adresse $01FF gestellt ist, da er sich immer auf Seite 1 befindet.

11.3 Befehlseingabe und Disassemblierung

Zwei Befehle ermöglichen die leichte Eingabe von Mikroprozessor-Befehlen in den Speicher und die Überprüfung der Befehle, die schon im Speicher vorhanden sind.

Der I-Befehl codiert (assembliert) eingegebene symbolische Befehle in direkt ausführbaren Maschinencode, der im Speicher gespeichert wird. Der K-Befehl decodiert (oder disassembliert) Maschinencode vom Speicher in symbolische Befehle, zur Anwender-Überprüfung.

11.3.1 I-Befehl – Betriebsart mnemonische Befehlseingabe

Der I-Befehl gibt die Prozessor-Befehle direkt als Maschinencode in den Speicher ein. Mit der Tastatur werden symbolische Befehle eingegeben. Beginnend bei einer vom Anwender eingegebenen Adresse, werden Maschinencodes (OP-Codes), in Form von alphabetischen Abkürzungen mit einer Länge von drei Buchstaben (siehe 10. Befehle) in den Speicher assembliert. Ungültige OP-Codes und Operanden werden ignoriert, bewirken aber die Anzeige der ERROR-Meldung.

Verwenden Sie den I-Befehl wie folgt:

1. Geben Sie I ein. PC 100 antwortet mit der augenblicklichen Programmzähleradresse:

   Beispiel:

   ⟨I⟩
   XXXX

2. Die Programmzähleradresse kann durch das Eingeben von *, gefolgt von einer hexadezimalen Adresse, verändert werden. Wird die Adresse Ø3ØØ eingegeben, antwortet PC 100 mit:

   Beispiel:

   Ø3ØØ

3. Geben Sie die dreistellige alphabetische Abkürzung des Maschinencods ein. Ein Eingabefehler bei den beiden ersten Buchstaben kann durch das Betätigen der Taste DEL und erneuter berichtigter Eingabe korrigiert werden.

   Sollte der eingegebene Op-Code keinen Operanden erfordern, wird der Maschinencode berechnet, im Speicher gespeichert und zusammen mit der Programmzähleradresse und dem symbolischen Op-Code in Maschinencode-Form angezeigt. Der Programmzähler wird um 1 erhöht. Wollen Sie in nachfolgenden Adressen zusätzliche Befehle eingeben, kehren Sie zu Schritt 3 zurück. Ist die Befehlseingabe beendet, kehren Sie durch Betätigen von ESC in den Monitor zurück.

   Sollte der Op-Code einen Operanden erfordern, fahren Sie mit Schritt 4 fort.

   Ist der Op-Code ungültig, wird eine „ERROR"-Nachricht angezeigt. Der richtige Op-Code kann dann eingegeben werden, ohne die Programmzähleradresse zu verändern, da diese nicht erhöht wurde.

Wurde ein gültiger, aber unerwünschter, Op-Code eingegeben, kann er durch zwei Methoden korrigiert werden:

- O Erfordert der Op-Code einen Operanden, geben Sie RETURN vor Eingabe des Operanden ein, oder tasten Sie absichtlich einen ungültigen Operanden ein. Damit wird eine „ERROR"-Meldung erzeugt und der gesamte Befehl kann neu eingegeben werden, da die Programmzähleradresse nicht verändert wurde.
- O Sollte der Op-Code keinen Operanden erfordern, wurde der Maschinencode in den Speicher eingegeben und der Programmzähler erhöht. In diesem Fall stellen Sie die vorherige Programmzähleradresse wieder her (siehe Schritt 2).

4. Geben Sie den Operanden hexadezimal gemäß der Adressierungsformate ein. In manchen Fällen ist eine verkürzte Form erlaubt. Die Anzeige meldet allerdings die Standardform, außer von bedingten Verzweigungs-Befehlen (absolute Adresse; im Gegensatz zur relativen Adresse). Die Form der Operanden-Eingabe, in der entsprechenden Adressierung, wird nachstehend gezeigt.

Adressierung		**Operanden-Format**
Accumulator	(Akkumulator)	A
Immediate	(Unmittelbare)	#HH
Zero Page	(Null Seite)	HH
Zero Page, X	(Null Seite X)	HH, X oder HHX
Zero Page, Y	(Null Seite Y)	HH, Y oder HHY
Absolute		HHHH
Absolute, X		HHHH, X oder HHHHX
Absolute, Y		HHHH, Y oder HHHHY
Relative[1])		HH oder HHHH
(Indirect, X)	(Indirekte, X)	(HH, X) oder (HHX) oder (HH, X) oder (HHX)
(Indirect, Y)	(Indirekte, Y)	(HH), Y oder (HH) Y
(Indirect)	(Indirekt)	(HHHH)

Anmerkungen:

1. *Unmittelbare Nullseiten- oder relative-Adressen erfordern die Eingabe von zwei Ziffern (HH).*
2. *Absolute Adressen erfordern die Eingabe von vier Ziffern (HHHH).*
3. *Das $-Symbol vor hexadezimalen Ziffern ist nicht erlaubt, da alle Eingaben hexadezimal definiert sind.*
4. *Die relative Adresse vom Programmzähler darf, im Falle von bedingten Verzweigungen, als zweistellige relative Adresse eingegeben werden, oder als eine vierstellige absolute Adresse, wobei der richtige Wert der relativen Adresse automatisch berechnet wird.*

H≙ Hexadezimale Daten
[1]) Siehe Anmerkung 4.

Beenden Sie die Eingabe der Operanden mit RETURN oder SPACE. Der Op-Code und der Operand werden berechnet und im Speicher abgelegt. Die Programmzähleradresse, der Op-Code Maschinencode und die symbolische Form des Op-Code und des Operanden werden gemeldet. Wurde SPACE verwendet, wird eine zweite Zeile angezeigt. Diese Zeile enthält den Programmzähler und die Maschinencode Form von Op-Code und Operand.

War der Operand ungültig, wird eine „ERROR-Nachricht" erzeugt und der gesamte Befehl muß neu eingegeben werden.

Ein Fehler in der Operand-Eingabe vor Betätigen von RETURN oder SPACE kann durch Taste DEL und erneuter Daten-Eingabe korrigiert werden. Ein Fehler in der Operand-Eingabe, nach RETURN oder SPACE, kann korrigiert werden, indem man die gewünschte Programmzähleradresse wiederherstellt und den gesamten Befehl neu eingibt.

Für die Eingabe von zusätzlichen Befehlen gehen Sie zu Schritt 2 zurück. Ist die Befehlseingabe beendet, kehren Sie durch ESC in das Monitorprogramm zurück.

Beispiel:

```
<I>
 0200     *=0300
 0300 EA NOP
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
 0309     *=0310
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
 0318
```

11.3.2 K-Befehl – Disassembliere Speicher

Der K-Befehl disassembliert Maschinencode vom Speicher in symbolische Befehle. Beginnend an einer bestimmten Adresse wird jedes Byte des Speichers disassembliert, bis ein gültiger Op-Code entziffert ist. Sobald ein gültiger Op-Code gefunden ist, wird die entsprechende Anzahl der nachfolgenden Bytes disassembliert, um den Befehlsoperanden zu bestimmen und anzuzeigen. Ungültige Op-Codes werden mit Fragezeichen gekennzeichnet. Vergleichen Sie die ungültigen Op-Codes mit den gültigen.

Verwenden Sie den K-Befehl wie folgt:

1. Tasten Sie K ein.

   Beispiel:

   ⟨K⟩* =

2. Geben Sie die Startadresse hexadezimal ein und betätigen Sie RETURN. Geben Sie Ø3ØØ ein.

   Beispiel:

   ⟨K⟩* =Ø3ØØ

3. Bestimmen Sie die Anzahl der Befehle, die disassembliert werden sollen, wie folgt:
   - Eingabe einer dezimalen Zahl von Ø1 bis 99
   - RETURN (ein Befehl), oder
   - „." oder SPACE (kontinuierliche Disassemblierung; ØØ≙1ØØ Befehle)

Der PC 100 disassembliert nun alle Befehle die vorgegeben wurden. Ein Unterbrechen der Disassemblierung ist zu jedem Zeitpunkt durch R/E oder ESC möglich.

Die Disassemblierung kann durch SPACE unterbrochen werden. (Zur Wiederaufnahme der Disassemblierung betätigen Sie eine beliebige Taste.)

Beispiel:

```
<K>*=0300
/05
 0300 EA NOP
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
<K>*=0310
/04
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
```

11.4 Ausführung/Protokoll Programmbefehle

Vier Befehle ermöglichen die Ausführung und die genaue Prüfung eines Anwender-Programms. Der G-Befehl führt das Anwenderprogramm in der Betriebsart aus, die durch die Stellung des RUN/STEP-Schalters bestimmt wird. In der Betriebsart RUN wird das Programm in Echtzeit durchgeführt, wobei die komplette Steuerung die CPU an das Anwenderprogramm abgegeben wurde.

In der Betriebsart STEP wird die Programmausführung nach jedem Befehl zwecks Durchführung des Befehlsprotokollprogramms, des Registerprotokollprogramms und der Breakpointüberprüfung angehalten. Der Z-Befehl steuert das Befehlsprotokollprogramm, während der V-Befehl das Registerprotokollprogramm regelt. Die Breakpointsteuerung wird in Kapitel 11.5, Abschnitt B beschrieben. Nachdem die Ausführung beendet und die Steuerung wieder an das Monitorprogramm zurückgegeben worden ist, kann ein Protokollprogramm des Programmzählers, mittels H-Befehl, eingeleitet werden.

11.4.1 G-Befehl – Beginn der Ausführung bei Programmzähleradresse

Der G-Befehl beginnt die Ausführung eines Anwenderprogramms an dem augenblicklichen Wert des Programmzählers.

5. Der PC 100 wird das Programm abarbeiten, bis eine Abschlußbedingung angetroffen wird:
   A. In der Betriebsart STEP wird der nächste durchzuführende Befehl disassembliert und gedruckt, wenn die Betriebsart Befehlsprotokoll (Trace) (Z-Befehl) eingeschaltet ist. Der Inhalt der sechs Register wird vor der Ausführung des nächsten Befehls ausgedruckt, wenn die Betriebsart Registerprotokollpro-

gramm (Z-Befehl) eingeschaltet ist. Die Durchführung wird beendet und die Kontrolle an das Monitorprogramm zurückgegeben, wenn die eingegebene Anzahl der Befehle erreicht, ein BRK-Befehl durchgeführt, oder eine Breakpointadresse erreicht ist (falls Breakpoints freigegeben).

B. In der Betriebsart RUN dauert die Befehlsdurchführung so lange, bis ein BRK-Befehl kommt. Gleichzeitig geht die Kontrolle zurück an das Monitorprogramm. Die Ausführung des Befehls kann aber auch durch den RUN/STEP-Schalter in STEP-Stellung beendet werden. Wenn der G-Befehl mittels RETURN-Taste eingeleitet ist, wird in der STEP-Betriebsart nur ein Befehl vor Rückgabe an das Monitorprogramm ausgeführt.

Anmerkung:
Falls die CPU versucht, einen nicht implementierten Op-Code oder einen Sprung an eine unrichtige Adresse durchzuführen, kann sie mittels R/E-Schalter aufgehalten werden. Die Kontrolle übernimmt dann wieder das Monitorprogramm.

Verwenden Sie den G-Befehl wie folgt:

1. Legen Sie den Schalter RUN/STEP in die gewünschte Stellung.
2. Bei STEP-Stellungswahl führen Sie folgende Anweisungen aus:
   - O Initialisieren Sie den Programmzähler-Wert mittels *-Befehl.
   - O Setzen Sie den gewünschten Befehlsprotokollprogramm-Zustand mittels Z-Befehl.
   - O Setzen Sie den gewünschten Registerprotokollprogramm-Zustand mittels V-Befehl.
   - O Legen Sie die gewünschten Breakpointadressen mittels B-Befehl fest.
   - O Geben Sie die Breakpointadressen frei oder blockieren Sie die Breakpointadressen mittels 4-Befehl.
   - O Zeigen Sie die Registerüberschriften und -inhalte mittels R-Befehl an.
3. Betätigen Sie Taste G.

   Beispiel:

   ⟨G⟩/

4. In der Betriebsart STEP geben Sie mit folgenden Eingaben die Anzahl der Befehle ein, die durchgeführt werden sollen:
   - O Eingabe einer dezimalen Zahl von Ø1 bis 99, oder
   - O RETURN (ein Befehl), oder
   - O „." oder SPACE (kontinuierliche Befehlsdurchführung).

Wenn der Abschluß der Programmausführung durch eine der Abschlußbedingungen des G-Befehles ausgelöst wurde, kann die Ausführung an der augenblicklichen Programmzähleradresse wieder aufgenommen werden, indem man Teile von Schritt 2 wiederholt, ohne den Programmzähler neu zu initialisieren. Verwenden Sie den R-Befehl zur Überprüfung des Wertes des Programmzählers vor Wiederaufnahme der Ausführung.

Beispiel 1:
Betriebsart STEP, Befehlsprotokollprogramm „Ein", Registerprotokollprogramm „Ein".

```
<Z>ON
<V>ON
<*>=0300
<R>
 **** PS AA XX YY SS
 0300 A0 00 FF 01 FF
<G>/
 0301 A0 00 FF 01 FF
 0301 A2 LDX #FE
 0303 A0 00 FE 01 FF
 0303 E8 INX
 0304 A0 00 FF 01 FF
 0304 D0 BNE 0303
 0303 A0 00 FF 01 FF
 0303 E8 INX
 0304 22 00 00 01 FF
 0304 D0 BNE 0303
 0306 22 00 00 01 FF
 0306 4C JMP 0310
 0310 22 00 00 01 FF
 0310 A0 LDY #02
 0312 20 00 00 02 FF
 0312 88 DEY
 0313 20 00 00 01 FF
 0313 D0 BNE 0312
 0312 20 00 00 01 FF
 0312 88 DEY
 0313 22 00 00 00 FF
 0313 D0 BNE 0312
 0315 22 00 00 00 FF
 0315 4C JMP 0301
 0301 22 00 00 00 FF
 0301 A2 LDX #FE
 0303 A0 00 FE 00 FF
 0303 E8 INX
 0304 A0 00 FF 00 FF
 0304 D0 BNE 0303
 0304 D0 BNE 0303
```

Beispiel 2:

Betriebsart STEP, Befehlsprotokollprogramm „Ein", Registerprotokollprogramm „Aus".

```
<Z>OFF
<Z>ON
<V>OFF
<*>=0300
<R>
 **** PS AA XX YY SS
 0300 A0 00 FF 00 FF
<G>/
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0303 E8 INX
 0304 D0 BNE 0303
 0306 4C JMP 0310
 0310 A0 LDY #02
 0312 88 DEY
 0313 D0 BNE 0312
 0312 88 DEY
 0313 D0 BNE 0312
 0315 4C JMP 0301
 0301 A2 LDX #FE
 0303 E8 INX
 0304 D0 BNE 0303
 0304 D0 BNE 0303
```

Beispiel 3:

Betriebsart STEP, Befehlsprotokollprogramm „Aus", Registerprotokollprogramm „Ein".

```
<Z>OFF
<V>ON
<*>=0300
<R>
 **** PS AA XX YY SS
 0300 A0 00 FF 00 FF
<G>/
 0301 A0 00 FF 00 FF
 0303 A0 00 FE 00 FF
 0304 A0 00 FF 00 FF
 0303 A0 00 FF 00 FF
 0304 22 00 00 00 FF
 0306 22 00 00 00 FF
 0310 22 00 00 00 FF
 0312 20 00 00 02 FF
 0313 20 00 00 01 FF
 0312 20 00 00 01 FF
 0313 22 00 00 00 FF
 0315 22 00 00 00 FF
 0301 22 00 00 00 FF
 0303 A0 00 FE 00 FF
 0304 A0 00 FF 00 FF
 0303 A0 00 FF 00 FF
 0304 22 00 00 00 FF
 0306 22 00 00 00 FF
 0310 22 00 00 00 FF
 0310 22 00 00 00 FF
```

11.4.2 Z-Befehl – Ein/Ausschalten des Befehlsprotokollprogramms

Der Z-Befehl steuert das Befehlsprotokollprogramm, wenn der RUN/STEP-Schalter in der STEP-Stellung ist und wenn die Befehle nicht innerhalb des ROM-Adressbereichs stehen. Das Befehlsprotokollprogramm zeigt die Disassemblierung jedes Befehls, bevor der Befehl ausgeführt wird.

Um den Z-Befehl zu verwenden, betätigen Sie die Taste Z. PC 100 antwortet mit dem Status des Befehlsprotokollprogramms:

⟨Z⟩ ON oder
⟨Z⟩ OFF

Beispiel:

```
<Z>ON
<V>OFF
```

In dem oben angeführten Beispiel schaltete der erste Z-Befehl das Befehlsprotokollprogramm ein.

Der zweite Z-Befehl schaltete das Befehlsprotokollprogramm aus.

11.4.3 V-Befehl – Ein/Ausschalten des Registerprotokollprogramms

Der V-Befehl steuert das Registerprotokollprogramm, wenn der RUN/STEP-Schalter in der STEP-Stellung ist und wenn die Befehle nicht innerhalb des ROM-Adressbereiches stehen. Das Registerprotokollprogramm zeigt den Inhalt jedes Registers, nach der Durchführung jedes Befehls, im gleichen Format wie beim R-Befehl (Abschnitt 11.2.6).

Um den V-Befehl zu verwenden, betätigen Sie die Taste V. PC 100 antwortet mit dem Status des Registerprotokollprogramms:

⟨V⟩ ON oder
⟨V⟩ OFF

11.4.4 H-Befehl – Protokollprogramm für Programmzähler

Der H-Befehl zeigt die Adressen der letzten vier Befehle an, die durchgeführt wurden und die Adresse des nächsten Befehls, der durchgeführt werden soll. Protokollfähigkeit existiert nur, nachdem der PC 100 Befehle im STEP-Mode ausgeführt hat.

Verwenden Sie den H-Befehl wie folgt:

1. Führen Sie den gewünschten Befehl mittels G-, F1-, F2- oder F3-Befehl in der Betriebsart STEP aus.
2. Nach dem Monitorprogramm-Stichwort betätigen Sie die Taste H. PC 100 antwortet mit:

   ⟨H⟩

   XXXX (Der erste der letzten vier durchgeführten Befehle).

   XXXX

   XXXX

   XXXX (Adresse des gerade ausgeführten Befehls).

   XXXX (Adresse des nächsten auszuführenden Befehls).

Beispiel:

```
<H>
 0303
 0304
 0306
 0310
 0312
```

Das angeführte Beispiel zeigt ein Programm, das aus einer Gruppe von aufeinanderfolgenden Nichtsprung-Befehlen besteht, beginnend bei $0303, mit einem JMP $0310, RTS, RTI oder einem Sprung-Befehl bei $0306.

11.5 Manipulieren der Breakpoints

Vier Befehle sind möglich:

- Überprüfen
- Löschen
- Setzen oder Löschen (selektiv)
- Freigeben der Breakpoint-Blockierung

Diese Befehle werden in Verbindung mit dem G-Befehl in der Betriebsart STEP verwendet, um die Befehlsausführung bei bestimmten Breakpointadressen anzuhalten. Die Befehle können daher während des Austestens von Programmen verwendet werden, um sicherzustellen, daß das Programm der erwarteten Adressen weiterläuft oder um Zwischendaten im Speicher bei bestimmten Adressen zu überprüfen.

11.5.1 ?-Befehl – Anzeige der Breakpoints

Der ?-Befehl zeigt die Adresse jedes der vier Breakpoints an. Der am weitesten links stehende vierstellige Hexadezimalwert ist die Adresse des Breakpoints 0, während der am weitesten rechts stehende Wert die Adresse des Breakpoint 3 ist. 0000 meldet, daß der Breakpoint gelöscht ist, d.h. es ist keine Breakpointadresse gesetzt.

Um den ?-Befehl zu verwenden, betätigen Sie Tasten SHIFT und ? gleichzeitig.

Beispiel:

```
<?>
AAAA AAAA AAAA AAAA
```

Beispiel:

```
<?>
 0312 0000 0000 0000
```

In dem angeführten Beispiel sind die Breakpointzahlen und ihre zugehörigen Adressen:

Breakpointnummer	Breakpoint Adresse
Ø	$Ø312
1	nicht gesetzt
2	nicht gesetzt
3	nicht gesetzt

11.5.2 #-Befehl – Löschen der Breakpoints

Alle Breakpoints können mittels #-Befehl gelöscht werden. Breakpoints werden gelöscht, wenn die PC 100 Stromversorgung eingeschaltet wird. RESET verändert die Breakpointadressen nicht.

Betätigen Sie, um den #-Befehl zu verwenden, SHIFT und # gleichzeitig.

Beispiel:

〈#〉 OFF

Hierdurch wird angezeigt, daß alle Breakpoints auf $ØØØØ gesetzt wurden.

Beispiel:

〈#〉OFF

11.5.3 B-Befehl – Setzen/Löschen von Breakpoints

Der B-Befehl setzt oder löscht die Adresse irgendeines der vier Breakpoints (Breakpoint Ø bis Breakpoint 3).

Um die Breakpoints überprüfen zu können, muß der PC 100 in der Betriebsart STEP sein und die Breakpoints müssen mit dem 4-Befehl freigegeben sein.

Sind der STEP-Mode und die Breakpoints freigegeben, hält der Prozessor jedesmal an, wenn ein Befehl in den Adressbereich $ØØØ1 bis $9FFF geholt wird. Es findet eine Programmübergabe an den Monitor durch den NMI-Unterbrechungsvektor statt (es sei denn, die NMI-Vektoradresse in Stelle $A4Ø2 ist geändert worden). Eine Überprüfung, ob die Breakpoints freigegeben sind, wird durchgeführt (siehe 4-Befehl). Sind die Breakpoints freigegeben, wird jede feste Breakpointadresse mit der Adresse des Befehls, der vor der Durchführung steht, verglichen. Stimmt die Adresse einer der gesetzten Breakpoints mit der Adresse des Befehls überein, der vor der Durchführung steht, wird die Ausführung des Programms angehalten und die Kontrolle an den Monitor zurückgegeben.

Verwenden Sie den B-Befehl wie folgt:

1. Betätigen Sie B.

   Beispiel:

   〈B〉 BRK/

2. Nach dem /-Stichwort bestimmen Sie durch Eingabe einer Zahl zwischen Ø und 3 den Breakpoint der gesetzt/gelöscht werden soll. PC 100 antwortet, indem er die Nummer des eingegebenen Breakpoints ausdruckt und ein =-Stichwort. Ist z.B. eine Ø eingegeben worden:

   ⟨B⟩ BRK/Ø=

3. Um einen Breakpoint zu setzen, geben Sie die Hexadezimaladresse ein, an dem das Programm anhalten soll. Um einen Breakpoint zu löschen, geben Sie Ø ein.
4. Nachdem die Adresse eingegeben worden ist, betätigen Sie RETURN. Die Kontrolle wird an den Monitor zurückgegeben. Geben Sie den B-Befehl erneut ein, um zusätzliche Breakpoints zu setzen oder zu löschen.

Beispiel:

```
<B>BRK/0=0310
<B>BRK/1=0312
<B>BRK/3=0
```

In dem Beispiel wurde Breakpoint Ø an die Stelle $Ø31Ø gesetzt, Breakpoint 1 an die Stelle $Ø312, Breakpoint 2 wurde nicht verändert und Breakpoint 3 an die Stelle $ØØØØ gesetzt (d.h. gelöscht).

11.5.4 4-Befehl – Ein/Ausschalten der Breakpointfreigabe

Der 4-Befehl schaltet die Breakpointfreigabe ein oder aus. Ist die Breakpointfreigabe EIN und die Betriebsart STEP gewählt, werden die Breakpoints in einem Programm überprüft.

In der normalen Betriebsart werden die Breakpointadressen mittels B-Befehl zugeteilt, dann werden die Breakpoints mittels 4-Befehl freigegeben, falls der Anwender sie zur Ausführung bringen will.

Der 4-Befehl erlaubt auch eine zeitweilige Blockierung der Breakpointadressen, ohne daß eine spätere Neueingabe erforderlich wird.

Die Breakpointfreigabe ist automatisch „AUS" geschaltet, wenn die Stromversorgung des PC 100 eingeschaltet wird. Spätere RESET's beeinflussen die Breakpointfreigabe nicht.

Betätigen Sie die Taste 4, um den 4-Befehl zu verwenden. Das System wird die Breakpointfreigabe ein- oder ausschalten und das Ergebnis anzeigen:

Beispiel:

⟨4⟩ ON oder

⟨4⟩ OFF

Beispiel:

```
<4>ON
<4>OFF
```

In dem Beispiel wurden die Breakpoints freigegeben (eingeschaltet), als der erste 4-Befehl eingegeben wurde. Die Breakpoints wurden blockiert (ausgeschaltet), als der zweite 4-Befehl eingegeben wurde.

11.6 Laden/Ausgabe des Speichers

Zwei Befehle ermöglichen das Laden von Maschinencode von einer Eingabe-Baugruppe in den Speicher oder die Ausgabe vom Speicher an eine Ausgabe-Baugruppe.

11.6.1 L-Befehl – Lade Speicher

Der L-Befehl lädt Maschinencode von einem beliebigen System-Baustein in den Speicher.

Verwenden Sie den L-Befehl wie folgt:

1. Betätigen Sie die Taste L.

   Beispiel:

   ⟨L⟩ IN=

2. Schreiben Sie den Code des Eingabe-Bausteins, von dem der Maschinencode geladen werden soll:

Gewünschter Eingabebaustein	Einzugebender Baustein-Code
Kassettenrekorder PC 100-Format	⟨T⟩
Kassettenrekorder KIM-1-Format	⟨K⟩
TTY-Lochstreifen	⟨L⟩
Anwenderdefiniert	⟨U⟩

3. PC 100 wird den Maschinencode von dem definierten Baustein in den Speicher laden. Wenn der gesamte Code geladen ist, druckt PC 100 das Monitorprogramm-Stichwort.

Enthält irgendeiner der gelesenen Aufzeichnungen einen Checksum-Fehler, oder sollte irgend ein Teil des Speichers nicht schreiben, dann wird die Nachricht MEM FAIL ausgedruckt, womit auf den ersten Block der Aufzeichnung hingewiesen wird, der den Fehler verursacht hat.

11.6.2 D-Befehl – Ausgabe des Speichers

Der D-Befehl wird verwendet, um den Inhalt des Speichers an einen Ausgabebaustein auszugeben. Der Speicherinhalt, der ausgegeben wird, ist in Maschinencode-Format und kommt von der Adresse, die nach FROM= bestimmt wird, bis zu der Adresse, die nach TO= bestimmt wird. Mehrfachausgaben von verschiedenen Gebieten des Speichers können durchgeführt werden, indem neue Start- und Endadressen eingegeben werden, nachdem man auf das MORE?-Stichwort Y antwortet. Um die Ausgabe ordnungsgemäß zu beenden, ist eine N-Antwort erforderlich.

Verwenden Sie den D-Befehl wie folgt:

1. Betätigen Sie die Taste D. PC 100 antwortet, indem er die Ausgabe-Startadresse erfragt:

   Beispiel:

   ⟨D⟩
   FROM=

2. Geben Sie die Startadresse der Ausgabe hexadezimal ein. Ein Eingabebefehl kann mittels DEL korrigiert werden, oder indem man bis zu 11 Zahlen eingibt; PC 100 akzeptiert nur die letzten 4 eingegebenen Zahlen. Beenden Sie die Eingabe mit RETURN oder SPACE.

   Geben Sie Ø3ØØ ein. PC 100 antwortet, indem er die Ausgabe-Endadresse erfragt:

   Beispiel:

   FROM=Ø3ØØ TO=

3. Geben Sie die Endadresse der Ausgabe hexadezimal ein. Ein Eingabefehler kann in der gleichen Art wie bei der Startadresse korrigiert werden. Beenden Sie die Eingabe mit einem RETURN oder SPACE. Wird Ø34Ø eingegeben, antwortet PC 100:

   Beispiel:

   FROM=Ø3ØØ TO=Ø34Ø
   OUT=

4. Geben Sie den Code des Ausgabebausteins ein, an den die Ausgabe gerichtet ist:

Gewünschter Ausgabebaustein	Einzugebender Baustein Code
PC 100-Anzeige/Drucker	⟨RETURN⟩ oder ⟨SPACE⟩
PC 100-Drucker	⟨P⟩
Kassettenrekorder PC 100-Format	⟨T⟩
Kassettenrekorder KIM-1-Format	⟨K⟩
TTY-Lochstreifen	⟨L⟩
Anwenderdefiniert	⟨U⟩
Blind-Ausgabe (keine)	⟨X⟩

5. Der Speicherinhalt wird an den angegebenen Ausgabebaustein in Maschinencodeformat ausgegeben. Wenn der Speicherinhalt bis zu der angegebenen Endadresse ausgegeben wurde, zeigt PC 100:

   Beispiel:

   MORE?

6. Soll ein anderer Abschnitt des Speichers ausgegeben werden, geben Sie eine Y-Antwort (yes) ein. PC 100 wird die neuen Start- und Endadressen erfragen. Soll kein Speicher mehr ausgegeben werden, geben Sie eine N-Antwort (no) ein.

7. Nach einer N-Antwort wird PC 100 den Abschlußcode ausgeben.

Anmerkung:
Ist die Ausgabe nicht mir einer N-Antwort beendet, wird die letzte Dateiaufzeichnung, die die Dateiaufzeichnungs-Summe enthält, nicht aufgezeichnet. Dies verursacht nachfolgend ein falsches Laden oder Kassettenverifizieren.

Beispiel 1:

```
<D>
FROM=0200 TO=0366
OUT=T F=DUMP1 T=1
MORE?N
```

Dieses Beispiel gibt die Speicherstellen $200 bis $366 aus an eine Kassettendatei mit dem Namen DUMP 1, die auf dem Kassettenrekorder Nr. 1 steht. Keine anderen Teile wurden ausgegeben.

Beispiel 2:

```
<D>
FROM=0200 TO=0366
OUT=T F=DUMP2 T=2
MORE?Y
FROM=0300 TO=038D
MORE?N
```

Dieses Beispiel gibt die Speicherstellen $200 bis $366 aus an eine Kassettendatei mit dem Namen DUMP 2, die auf Kassettenrekorder Nr. 2 untergebracht ist. Es sollten weitere Speicherstellen ausgegeben werden, sodaß die Frage MORE? mit Y beantwortet wurde. Die Ausgabe war von der Stelle $0300 bis Stelle $0380. Es wurden keine weiteren Teile an diese Datei ausgegeben.

Beispiel 3:

```
<D>
FROM=0300 TO=0316
OUT=

:170300EAA2FEE8D0FD4
C10033027A2563633342A
00288D0FD4C010AF6
MORE?Y
FROM=0380 TO=03BD

:180380AB8353015040 7
4482F69370CF515632DD
742143D6E4D5F09086B
:18039871 6A53A0770B7
A25EE1EF3133B0D77806
A46571124012F040863
:0E03B01B47768BD90E6
7A1EA54570AEE4006E0
MORE?N:0000050005
```

Dieses Beispiel zeigt eine tatsächliche Speicherausgabe von Stelle $Ø3ØØ bis $Ø316 und von $Ø38Ø bis $Ø3BØ. Die Frage OUT= wurde mit RETURN beantwortet. Die Ausgabe erfolgt über Drucker und Anzeige.

Anmerkung:
Wenn der Speicherinhalt in KIM-1-Format an den Kassettenrekorder ausgegeben wird (OUT=K), muß die eingegebene TO-Adresse ein Byte größer sein als die letzte auszugebende Adresse.

11.7 Schnittstellen mit vom Anwender definierten Funktionen

Drei Befehle erlauben die Ausführung von 3 getrennten vom Anwender definierten Funktionen (Programmen) durch das PC 100-Monitorprogramm, mittels Tasten F1, F2 und F3. Um eine Funktionstaste verwenden zu können, muß die Verbindung zur Anwenderfunktion vorhanden sein. Diese Verbindung stellt ein JMP-Befehl zu der Startadresse des Programms dar. Der JMP-Befehl sollte an der Funktions-Verbindungsstelle untergebracht sein (siehe spezifische Funktionsnummer für eigentliche Adresse, Kapitel 11.7.1., Abschnitt 1). Nach Beendigung der vom Anwender definierten Funktion, kann die Kontrolle am Ende des vom Anwender definierten Funktionsbefehls an den PC 100-Monitor mit einem RTS-Befehl zurückgegeben werden.

Der JMP-Befehl an die Funktions-Verbindungsadresse kann mittels I-Befehl (mnemonischer Eingabe), dem Assembler, oder dem man den JMP-Befehl hexadezimal mittels M-Befehle (verändere Speicher) eingibt, festgelegt werden.

11.7.1 F1-, F2-, F3-Befehl – vom Anwender definierte Funktionen 1, 2 und 3

Die F1-, F2- und F3-Befehle werden verwendet, um vom Anwender definierte Funktionen einzugeben.

Um den F1-, F2- oder F3-Befehl zu verwenden, verfahren Sie wie folgt:

1. Programmieren Sie einen JMP-Befehl auf die Anfangsadresse Ihres Programms, in Adresse $Ø1ØC für die Startadresse der Funktion 1, in $Ø1ØF für Funktion 2 oder in $Ø112 für Funktion 3.
2. Starten Sie die Funktion, indem Sie die F1-, F2- oder F3-Taste betätigen. PC 100 antwortet mit folgender Anzeige und wird die Funktion 1, 2 oder 3 starten:

Taste	Stichwort
F1	<[>
F2	<]>
F3	<^>

3. Kehren Sie zu dem Monitorprogramm zurück, indem Sie ohne ein vorhergehendes JRS in dem Funktionsprogramm einen RTS-Befehl ausführen, oder betätigen Sie die Tasten R/E.

Beispiel F1:

```
<I>
 0200     *=010C
 010C 4C JMP 0220
 010F     *=0220
 0220 60 RTS
 0221
<I>
```

Beispiel F2:

```
<I>
 0200     *=010F
 010F 4C JMP 0240
 0112     *=0240
 0240 60 RTS
 0241
<I>
```

Beispiel F3:

```
<I>
 010F      *=0112
 0112 4C JMP 0260
 0115      *=0260
 0260 60 RTS
 0261
<^>
```

12. Tabellenanhang

12.1 ASCII – Zeichen – Codes

Dezimal	Zeichen	Dezimal	Zeichen	Dezimal	Zeichen
000	NUL	043	+		
001	SOH	044	,	086	V
002	STX	045	–	087	W
003	ETX	046	.	088	X
004	EOT	047	/	089	Y
005	ENQ	048	0	090	Z
006	ACK	049	1	091	[
007	BEL	050	2	092	\
008	BS	051	3	093	]
009	HT	052	4	094	^
010	LF	053	5	095	←
011	VT	054	6	096	'
012	FF	055	7	097	a
013	CR	056	8	098	b
014	SO	057	9	099	c
015	SI	058	:	100	d
016	DLE	059	;	101	e
017	DC1	060	<	102	f
018	DC2	061	=	103	g
019	DC3	062	>	104	h
020	DC4	063	?	105	i
021	NAK	064	@	106	j
022	SYN	065	A	107	k
023	ETB	066	B	108	l
024	CAN	067	C	109	m
025	EM	068	D	110	n
026	SUB	069	E	111	o
027	ESCAPE	070	F	112	p
028	FS	071	G	113	q
029	GS	072	H	114	r
030	RS	073	I	115	s
031	YS	074	J	116	t
032	SPACE	075	K	117	u
033	!	076	L	118	v
034	"	077	M	119	w
035	#	078	N	120	x
036	$	079	O	121	y
037	%	080	P	122	z
038	&	081	Q	123	{
039	'	082	R	124	|
040	(	083	S	125	}
041	)	084	T	126	~
042	*	085	U	127	DEL

LF = Line-Feed (Zeilenvorschub)
FF = Form-Feed (Papiervorschub)
CR = Carriage Return (Wagenrücklauf)
DEL = (Rubout am TTY)

12.2 Potenzen von 2

2^n	n	2^{-n}
1	0	1,0
2	1	0,5
4	2	0,25
8	3	0,125
16	4	0,062 5
32	5	0,031 25
64	6	0,015 625
128	7	0,007 812 5
256	8	0,003 906 25
512	9	0,001 953 125
1 024	10	0,000 976 562 5
2 048	11	0,000 488 281 25
4 096	12	0,000 244 140 625
8 192	13	0,000 122 070 312 5
16 384	14	0,000 061 035 156 25
32 768	15	0,000 030 517 578 125
65 536	16	0,000 015 258 789 062 5
131 072	17	0,000 007 629 394 531 25
262 144	18	0,000 003 814 697 265 625
524 288	19	0,000 001 907 348 632 812 5
1 048 576	20	0,000 000 953 674 316 406 25
2 097 152	21	0,000 000 476 837 158 203 125
4 194 304	22	0,000 000 238 418 579 101 562 5
8 388 608	23	0,000 000 119 209 289 550 781 25
16 777 216	24	0,000 000 059 604 644 775 390 625

12.3 Potenzen von 16

16^n	n	16^{-n}
1	0	$0{,}10000\ 00000\ 00000\ 00000 \times 10$
16	1	$0{,}62500\ 00000\ 00000\ 00000 \times 10^{-1}$
256	2	$0{,}39062\ 50000\ 00000\ 00000 \times 10^{-2}$
4 096	3	$0{,}24414\ 06250\ 00000\ 00000 \times 10^{-3}$
65 536	4	$0{,}15258\ 78906\ 25000\ 00000 \times 10^{-4}$
1 048 576	5	$0{,}95367\ 43164\ 06250\ 00000 \times 10^{-6}$
16 777 216	6	$0{,}59604\ 64477\ 53906\ 25000 \times 10^{-7}$
268 435 456	7	$0{,}37252\ 90298\ 46191\ 40625 \times 10^{-8}$
4 294 967 296	8	$0{,}23283\ 06436\ 53869\ 62891 \times 10^{-9}$
68 719 476 736	9	$0{,}14551\ 91522\ 83668\ 51807 \times 10^{-10}$
1 099 511 627 776	10	$0{,}90949\ 47017\ 72928\ 23792 \times 10^{-12}$
17 592 186 044 416	11	$0{,}56843\ 41886\ 08080\ 14870 \times 10^{-13}$
281 474 976 710 656	12	$0{,}35527\ 13678\ 80050\ 09294 \times 10^{-14}$
4 503 599 627 370 496	13	$0{,}22204\ 46049\ 25031\ 30808 \times 10^{-15}$
72 057 594 037 927 936	14	$0{,}13877\ 78780\ 78144\ 56755 \times 10^{-16}$
1 152 921 504 606 846 976	15	$0{,}86736\ 17379\ 88403\ 54721 \times 10^{-18}$

12.4 Hexadezimal – Dezimal – Umwandlung

Hexadezimale Spalten											
6		5		4		3		2		1	
Hex.	Dez.	Hex.	Dez.	Hex.	Dez.	Hex.	Dez.	Hex.	Dez.	Hex.	Dez.
0	0	0	0	0	0	0	0	0	0	0	0
1	1 048 576	1	65 536	1	4 096	1	256	1	16	1	1
2	2 097 152	2	131 072	2	8 192	2	512	2	32	2	2
3	3 145 728	3	196 608	3	12 288	3	768	3	48	3	3
4	4 194 304	4	262 144	4	16 384	4	1 024	4	64	4	4
5	5 242 880	5	327 680	5	20 480	5	1 280	5	80	5	5
6	6 291 456	6	393 216	6	24 576	6	1 536	6	96	6	6
7	7 340 032	7	458 752	7	28 672	7	1 792	7	112	7	7
8	8 388 608	8	524 288	8	32 768	8	2 048	8	128	8	8
9	9 437 184	9	589 824	9	36 864	9	2 304	9	144	9	9
A	10 485 760	A	655 360	A	40 960	A	2 560	A	160	A	10
B	11 534 336	B	720 896	B	45 056	B	2 816	B	176	B	11
C	12 582 912	C	786 432	C	49 152	C	3 072	C	192	C	12
D	13 631 488	D	851 968	D	53 248	D	3 328	D	208	D	13
E	14 680 064	E	917 504	E	57 344	E	3 584	E	224	E	14
F	15 728 640	F	983 040	F	61 440	F	3 840	F	240	F	15

12.5 Relative Verzweigungsadressen

12.5.1 Verzweigung „vorwärts"

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

12.5.2 Verzweigung „rückwärts"

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

MSD ≙ Höherwertiges $^1/_2$ Bytes
LSD ≙ Niederwertiges $^1/_2$ Bytes

Notizen

Bestell-Nr. B/2327
Printed in Germany
KG 10805.