ATARI®

ATARI PROGRAMM

DXG 8126
Diskette

MACRO ASSEMBLER

ATARI®
A Warner Communications Company

Wichtige Informationen

Lieber Computerfreund, lieber Kunde, lieber Händler!

Jeder, der sich einmal selbst damit beschäftigt hat, ein Computerprogramm zu fertigen, weiß, welche Arbeit und geistige Mühe aufgewendet werden muß, um eine Problemlösung zu finden und sie anwenderfreundlich zu programmieren. Die Erfüllung dieser Voraussetzungen erfordert viel Erfahrung und hohe finanzielle und zeitliche Investitionen. Das Ergebnis sind gute und erfolgreiche Computerprogramme, die von interessierten Anwendern nachgefragt werden und deshalb für den Händler verkäuflich sind.

Diese Tatsache machen sich einige dadurch zunutze, daß sie die mit hohen Voraufwendungen geschaffenen erfolgreichen Programme der Firma Atari kopieren oder ihren Kunden die Möglichkeit anbieten, die gewünschten Programme auf Diskette zu überspielen. Sie meinen, damit ihren Kunden ein gutes und billiges Angebot zu machen. Die Kunden wissen jedoch meist nicht, daß sie lediglich ein vermeintlich gutes und billiges Angebot erhalten.

Abgesehen davon, daß das Angebot zur Überspielung von Programmen und das Anbieten und Verkaufen illegal kopierter Programme strafrechtlich verboten ist, weil es sich dabei um Verletzungen des Urheberrechtes (COMPUTERPROGRAMM PIRATERIE) handelt, die von Atari gegenüber jedermann ohne Ansehen der Person gerichtlich verfolgt wird, so ist auch die Annahme falsch, das Angebot sei günstig oder billig:

- Gestohlene Ware ist immer billig. Der Dieb hat keine Voraufwendungen. Er eignet sich nur fremdes Eigentum an, für die der Käufer keine Gewährleistung erhält.
- Der Händler, der das Kopieren von Programmen anbietet, anstatt Originale zu verkaufen schmarotzt an fremder Leistung.
- Der interessierte Kunde wird bald keine guten Programme mehr kaufen können und illegale Programme wird der Handel bald auch nicht mehr anbieten können.

Letzteres deswegen, weil niemand mehr bereit und in der Lage sein wird, gute verkaufsfähige Programme zu entwickeln, wenn nicht die Möglichkeit besteht, die hohen Voraufwendungen durch Verkäufe wieder zu verdienen. Die Piraten sind geistig weder in der Lage noch überhaupt bereit, sich der Mühe zu unterziehen, Programme zu entwickeln. Sie können und wollen nur durch Diebstahl fremder guter Leistung eine schnelle bequeme Mark verdienen.

Wer also Interesse daran hat, daß das Angebot an guten Computerprogrammen wächst, sollte die illegalen „billigen" Angebote meiden und mit dazu beitragen, daß den Totengräbern der Computer-Programmentwicklung und damit des Computerhandels das Handwerk gelegt wird.

Wir danken für Ihr Verständnis und freuen uns über jeden Hinweis von Ihnen.

Atari Elektronikvertriebsges. mbH

ATARI ELEKTRONIK Vertriebsgesellschaft mbH · Bebelallee 10 · 2000 Hamburg 60 · Tel. 040/511 80 91

Inhalt

1 EINFUEHRUNG

Der ATARI MACRO ASSEMBLER ist eine Softwareentwicklung, geschrieben und als Hilfsmittel gedacht, fuer den Assembler des 6502 Processors fuer das ATARI HEIMCOMPUTER SYSTEM. Die Moeglichkeiten dieses Assemblers umfassen Macros, bedingte Assemblierung, Codevervielfachung, Zugriff zu freien Definitionen,Kontrolle des Programmlistings und cross-reference Tafeln. Er bietet schnelle Uebersetzung und benutzt die Standard 6502 Mnemonics.

Moeglichkeiten dieses Programmpaketes

Macros

Der Macro erlaubt einem Codewoerter zu definieren, die viele Befehle beinhalten. Dies macht es leicht, eine Serie von Codes mehrmals innerhalb eines Programms zu verwenden.

Bedingte Assemblierung und Mehrfachverwendung des Codes

Bedingte Assemblierung erlaubt die Erzeugung von Objektcode unter bestimmten Voraussetzungen. Verbunden mit den Macros ergibt dies eine leistungsfaehiges und vielseitiges Assembler Programm. Die ECHO-Pseudooperation ermoeglicht es, einen Teil des Codes zu wiederholen (aehnlich wie Macros, nur sind keine Parameter erlaubt).

Systext Files

Oft moechte man Macros entwerfen und in einer freiverfuegbaren Datei speichern. Einmal entworfen, koennen diese Macros in mehreren Quellprogrammen verwendet werden. Das erleichtert die Programmentwicklung.

Steuerung des Programmausdruckes

Der Pseudobefehl LIST ermoeglicht es, Programme auszudrucken, und so Programme leichter und exacter zu erstellen und genau das herauszufinden, was man braucht. Dieser Pseudo-Befehl erleichtert die Dokumentation durch Programmausdrucke und Seitenueberschriften.

Cross-Reference Tafeln

Der Macro-Assembler beinhaltet wahlfreie Cross-Reference Tafeln, so dass Labels und Variable schnell im Quellprogramm verwendet werden koennen.

ATARI und 6502 Standardmnemonics

In der Macro Assembler Diskette ist eine Datei enthalten, in der die Hardware Adressen und die OS Shadow Adressen des ATARI Computers enthalten sind. Mit Hilfe dieser Datei kann man in den Programmen die ATARI Standardmnemonics verwenden. Siehe hierzu "Optionen innerhalb einer Commandozeile " in Kapitel 2.

In diesem Assembler sind die Codes der Standard MOS Technologie des 6502 Mikroprozessors verwendet worden. Die Befehle entsprechen somit der Standardkonvention.

Inhalt dieses Software Paketes
Der Macro Assembler beinhaltet:

*Eine Diskette mit dem Macro Assembler und dem Text Editor
*Dieses Handbuch fuer den ATARI Macro Assembler
*Ein Befehlshandbuch fuer den ATARI Text Editor

Vorgehensweisen

Laden des Programms

1. Verbinden des ATARI Computers mit dem Fernseher nach den entsprechenden Gebrauchsanweisungen.
2. Verbinden der ATARI Diskettenstation mit dem Computer nach der Gebrauchsanweisung der Diskettenstation. Die Diskettenstation muss als 1. Station angeschlossen sein.
3. Herausnahme aller ROM's aus dem Schacht. Es darf kein ROM im Schacht des Computers sein.
4. Einschalten des Fernsehers.
5. Einschalten der Diskettenstation. Beide roten LED'S leuchten auf (die BUSY und die PWR ON LED).
6. Wenn die BUSY LED ausgegangen ist , die Klappe der Diskettenstation oeffnen.
7. Die Diskette mit dem Macro Assembler und dem Text Editor in die Diskettenstation schieben und die Klappe der Diskettenstation schliessen.
8. Den Computer einschalten.
Das DOS II Menue erscheint auf dem Bildschirm.

Erstellung eines Quellprogramms

Bei Benutzung des Editors, wird auf das Handbuch des Text Editors verwiesen.
Nach dem das Quellprogramm erstellt wurde, wird der Text Editor mit dem Befehl verlassen, der zum DOS zurueck fuehrt.

1. Druecken der <OPTION> Taste.
2. EXIT eingeben und die <START> Taste druecken.(Dies fuehrt zum DOS).

Zum Assemblieren des Quellprogramms
1. Den Buchstaben L eingeben und die <RETURN> Taste druecken.
2. AMAC eingeben und die <RETURN> Taste druecken.

Asseeblieren eines Quellprogramms

1. Nachsehen unter " Aufbau einer Kommandozeile " (in Kapitel 2) ueber den Aufbau und die Optionen einer Kommandozeile. Nach Eingabe der Kommando Zeile die <RETURN> Taste druecken.
2. Nach dem Asseeblieren, die <RETURN Taste druecken, um wieder in das DOS zurueckzukehren.Ist man wieder im DOS Menue und waehlt die Directory Option, wird man erkennen das eine Objektcode Datei mit der Extension OBJ entstanden ist.

Ziel dieser Anleitung

Diese Anleitung wurde geschrieben, um zu zeigen wie mit dem Macro Assembler gearbeitet wird. Wenn man beabsichtigt den Text Editor zum Erstellen von Quellprogrammen zu verwenden, ist es besser, erst das ATARI Text Editor Handbuch zu lesen, bevor man versucht Programme zu erstellen.

Zur Benutzung des Macro Assemblers ist es noetig, den Assembler und das ATARI DOS II zu kennen. Die Literaturangaben weiter unten sollen bei dem Selbststudium des Assemblers helfen. Wenn man mit den speziellen Moeglichkeiten des ATARI Computers vertrauter werden moechte, hilft hierbei "ATARI Technical Users Notes".

Literaturangaben

Wir empfehlen folgende Buecher:

* MOS Programming Manual von MOS Microcomputers
* SY6500/MCS6500 Microcomputer Family Programming Manual von SYNERTEK
* 6502 Assembly Language Programming von Lance Leventhal (gibt es auch in deutsch) * 6502 Software Design von Leo Scanlon
* 6502 Software Gourmet Guide and Cookbook von Robert Findley

ATARI Veroeffentlichungen:

* ATARI DOS II Reference Manual
* ATARI Technical Users Notes

2 Einsatz des Assemblers

Aufbau einer Kommandozeile

Der Macro Assembler wird ueber das DOS II Menue mit der Option L aufgerufen. Nach dem die Option L gewaehlt wurde, erscheint die Frage nach dem Dateinamen, hierauf AMAC eingeben und die <RETURN> Taste druecken.

Wenn das Programm in den Speicher geladen wurde, erscheint der Satz "Enter source filename and options". Der Quelldateiname muss immer eigegeben werden. Nach der Eingabe des Dateinamen folgt, durch ein Komma oder durch ein Freizeichen getrennt, die gewuenschte Option.

Die Kommandozeile wird mit einem <RETURN> und einem Sprung in die naechste Zeile abgeschlossen. Die Kommandozeile kann nicht mit den Cursorsteuerzeichen geaendert werden.

Der Aufbau einer Kommandozeile, der unbedingt eingehalten werden muss, ist : <Dateispezifikation> Option1,...Option. Wobei <Dateispezifikation> eine Quelldatei bezeichnet, die assembliert werden soll und von der Form <Geraet>:<Dateiname>.<Extension> ist. Diese Kommandozeile kann beliebig in grossen oder kleinen Buchstaben geschrieben werden, denn sie wird vor der Bearbeitung ,vom Assembler in grosse Buchstaben umgesetzt.

Optionen innerhalb einer Kommandozeile

Die opt1,...optn sind moegliche Parameter und koennen, in beliebiger Reihenfolge, nach folgender Liste ausgewaehlt werden:

H=Dn:	Erzeuge ein Object-Codefile auf der angegebenen Diskettenstation, wobei n=1,2,3 oder 4 sein kann. Wird kein Dateiname angegeben, erhaelt das Objekt-Codefile denselben Namen wie das Quell-Codefile und die Extension OBJ. (Fehlt die Angabe wird die Object-Code-Datei auf dieselbe Diskette geschrieben, auf der sich der Quellcode befindet.)
H=<filespezi-fikation>	Schreibe den Objektcode in <filespecifikation>
H=O	Erzeuge keinen Objektcode
L=P:	Ausgabe auf den Drucker
L=Dn:	Ausgabe auf die bezeichnete Diskettenstation (n=1,2,3 oder 4). Dieses File hat den Filenamen des Quellprogramms mit der Extension PRN.
L=S:	Ausgabe auf den Bildschirm.
L=O	Fertige kein Listing fuer dieses assemblierte Programm an. (L=O wird angenommen, wenn kein anderes Listfile angegeben wird.)

O=n	Stelle dem Objektprogramm den Wert der Startadresse voran. Die Angabe "O=n" in der Kommandozeile hat exakt die gleiche Bedeutung wie "END n" am Ende eines Assemblerprogramms.
O=0	Setze den Wert der Startadresse auf Null. (O=0 wird angenommen, wenn keine anderere Startadresse definiert wird.)
PS=n	Gib die Anzahl der Zeilen je Seite an. Es duerfen nicht mehr als 126 Zeilen je Seite angegeben werden. Wenn die Zeilenzahl je Seite kleiner als zehn ist, werden weder Ueberschriften und Untertitel noch Seitenzahlen in dem Listfile angegeben, ausserdem ist die vollstaendige Cross-Reference Tafel nicht anwendbar. (PS=63 wird angenommen, wenn keine andere Seitenlaenge angegeben wird.)
PS=0	Gib keine Ueberschriften, Untertitel und Seitenzahlen in diesem Listfile bei der Assemblierung an.
S=<filespezi-fikationen>	Spezifiziere das System Textfile. Die S-Option kann wiederholt werden. Der Benutzer kann soviele System Textfiles spezifizieren wie er moechte, solange die Anzahl der System Textfiles und der Hilfsprogramme zusammen die Zahl 40 nicht uebersteigt.
S	Benutze wenn nicht anders angegeben D:SYSTEXT.AST.
S=0	Keine Spezifizierung fuer diese Assemblierung. (S=0 wird angenommen, wenn S=... fehlt.)
R=F	Erzeuge vollstaendige Referenzuebersichtstafel. Liste alle allgemeingueltigen Symbole und deren Auftreten auf die Filespezifikation L Parameter.
R=S	Erzeuge die kurze Referenzuebersichtstafel. Liste alle allgemeingueltigen Symbole und nur deren Werte auf die Filespezifikation der L Parameter.
R=0	Erzeuge keine Referenzuebersichtstafel. (Wird angenommen, wenn R=... fehlt.)

SL=n — Setze die Laenge der Zeilen fest. Die maximale Zeilenlaenge des Listfiles hat <n> Symbole. Der Rest der Zeile geht verloren, wenn maximale Zeilenlaenge des angesprochenen Geraetes ueberschritten wird. (Wird SL=... nicht angegeben, so ist SL=80 fuer P: bzw. SL=38 fuer S:)

Alle numerischen Werte (bei O=n, PS=n und SL=n) koennen nach den allgemeinen Zahlenregeln benutzt werden. Speziell kann eine Basis angegeben (dezimal, binaer, octal oder hexadezimal) koennen verwendet werden. Siehe auch hierzu "Zahlen" in Kapitel 4.

Alle kleinen Buchstaben in einem Kommando werden in grosse Buchstaben umgesetzt, bevor das Kommando bearbeitet wird.

Beispiele fuer Kommandozeilen

D:TESTIT.ASM

Liest das Eingabefile D1:TESTIT.ASM(D: ist gleichbedeutend mit D1:), es wird keine Liste hergestellt und das ATARI Binaerformat des Objekt-Codefile heisst D1:TESTIT.OBJ.

D:TESTIT.ASM H=0 R=F L=S:

Assembliert D1:TESTIT.ASM, erzeugt keinen Objektcode, und listet die vollstaendige Referenzuebersichtstafel auf dem Bildschirm.

D2:TESTIT.ASM H=D: L=D2: R=F O=$200

Der Assembler assembliert das File D2:TESTIT.ASM, erzeugt das Objektfile D1:TESTIT.OBJ, und stellt eine Liste der vollstaendigen Referenzuebersichts- tafel her, und speichert sie in der Form D2:TESTIT.PRN ab. Ausserdem wird die Startadresse auf $200 gesetzt.

D2:TESTIT.ASM S S=D2:MSYS.AST L=P: R=F H=D: O=$1700

Der Assembler verwendet die beiden System Textfiles D1:SYSTEXT.AST und D2:MSYS.AST, assembliert das File D2:TESTIT.ASM, produziert das Objektfile D1:TESTIT.OBJ mit der Startadresse $1700, und listet die vollstaendige Referenztafel auf dem Drucker.

Hinweise zum Assemblieren

Die Assemblierung kann vorzeitig durch Druecken der <BREAK> Taste unterbrochen werden. Die Ausgabe auf den Bildschirm kann unterbrochen werden, indem die <CTRL> und Ziffer 1 gleichzeitig gedrueckt werden. Wiederholt man diesen Vorgang wird die Ausgabe auf dem Bildschirm fortgesetzt.

Falls beim Schreiben auf die Diskette ein Fehler auftritt(normalerweise wenn die Diskette oder die Directory voll ist), ist das fehlerhafte File (Objekt- oder Listfile) geloescht, auf dem Bildschirm erscheint eine Fehlermeldung und weitere Schreibversuche scheitern. Die Assemblie- rung wird normal fortgesetzt. Fehler, die beim Assemblieren auftreten, werden sowohl auf dem Bildschirm als auch in das Listfile geschrieben.

3 Dateistrukturen

Quell-Codefiles (Eingabe)

Das Quell-Codefile kann folgendermassen spezifiziert werden:

* Argument der ersten Kommandozeile
* Argument des System-Textfiles (S Parameter)
* LINK-Pseudobefehl
* INCLUDE-Pseudobefehl

Alle Quell-Codefiles muessen dem Text Editor-Format entsprechen. Sie bestehen aus einer oder mehreren Zeilen mit ATASCII Symbolen, beendet werden sie mit <EOL> End-of-Lines.

System Textfiles

Ein System Textfile (systext) ist ein Assemblierungsfile mit Symbolen und Macrodefinitionen. Der Programmierer kann hier Symbole erstellen, die er in vielen verschiedenen Programmen verwenden kann. Hier einige Beispiele:

* ATASCII Steuerzeichen (BS, TAB, ESC, EOL,...)
* Adressen (Eingabestellen zum CIO, SIO, und Kanalbezeichnungen)
* Macros

Wenn bei der Assemblierung waehrend der Durchsicht des System Textfiles unerwartet ein Fehler auftritt, bricht der Assembler ab und gibt eine Fehlermeldung aus.

Objekt-Codefile (Ausgabe)

Das Objekt-Codefile wird vom Assembler erzeugt, und wenn nichts anderes angegeben, mit der Extension OBJ versehen. Das File liegt im ATARI Binaercode vor. Fuer naehere Angaben siehe auch ATARI DOS II Reference Manual.

Assemblerprotokoll

Die Assemblerprotokolle des Quell-Codefiles, die vom Assembler erzeugt werden haben, wenn nichts anderes angegeben ist, die Extension PRN.

Der Macro Assembler hat einen anpassungsfaehigen Satz von Listing-Pseudobefehlen, die dem Benutzer erlauben, ein Listing nur indem Umfang zu erzeu- gen den er benoetigt.

Seitenueberschriften (wenn sie nicht durch PS=0 unterdrueckt wurden) sind in dieser Assemblerversion enthalten, und Seitenzahlen koennen soviel angegeben werden, wie der Benutzer sie zur Information benoetigt (siehe auch unter den Pseudobefehlen TITLE und SUBTTL).

Der Pseudobefehl LIST (oder L in der Kommandozeile) listet den Quellcode. Jedes Listing beginnt mit 20 Spalten Informationen, die vom Assembler ausgegeben werden.

Die erste Spalte ist reserviert fuer Fehlermeldungen; ist das assemblierte Programm frei von Fehlern, bleibt die erste Spalte frei. Die Anzahl der Fehler wird am Ende der Assemblierung angezeigt(siehe hierzu "Fehlermeldungen" in Kapitel 10).

Struktur des Quellprogramms

```
          1         2
123456789.123456789.
E addr# hhhhhhhhhh      Zeile mit dem erzeugten Code.
R addr  = vvvv          EQU, SET, IF, etc.
R-                      Zeile wird ueberschlagen.
O addr  = vvvv#         Ursprungs- und Adressenzaehler stimmen
                        nicht ueberein.
R addr
        + hhhhhhhhhh    Erzeugte Macrozeile.
  addr  hhhh ^addr      Zieladresse des Programmzaehlers bei
                        einem relativer Sprung.
```

Spalte	Beschreibung
1	Fehlerflag oder Leerzeichen. Fuer die Bedeutung der Fehlerflags siehe Kapitel 10.
2	Leerzeichen.
3-6	Adressenangabe der Anweisung(Wert des Zaehlers).
6	- Zeichen heisst: Zeile wurde wegen dem IF...ELSE Befehl nicht assembliert. Zeile wird nur bei der Funktion LIST F gelistet.
7	# Zeichen heisst: Ursprungs- und Adressenzaehler stimmen nicht ueberein.
8	+ Zeichen heisst: Vom Assembler erzeugte Zeile. Zeile wird bei der Funktion LIST M gelistet.
9-18	hhhhhhhhhh der Ergebniscode. Wird bis zum fuenften Byte gelistet. Bei der Funktion LIST G oder LIST D werden alle Zeilen jeweils bis zum fuenften Byte gelistet.
11-14	vvvv = Wert eines Ausdruckes.
19-20	Sind immer Leerzeichen.
21-80	Quellcode.

Programmbeispiel

```
                    I/O Zuweisungen.
                =009B       EOL =       $9B
                =0030       IOCB3 =     $30
                =0340       ICHID =     $0340
                =0341       ICDNO =     ICHID+1
                =0342       ICCOM =     ICDNO+1
                =0343       ICSTA =     ICCOM+1
                =0344       ICBAL =     ICSTA+1
                =0345       ICBAH =     ICBAL+1
                =0346       ICPTL =     ICBAH+1
                =0347       ICPTH =     ICPTL+1
                =0348       ICBLL =     ICPTH+1
                =0349       ICBLH =     ICBLL+1
                =034A       ICAX1 =     ICBLH+1
                =034B       ICAX2 =     ICAX1+1
                =0003       OPEN =      $03
                =0005       GETREC =    $05
                =0009       PUTREC =    $09
                =000C       CLOSE =     $0C
                =0004       OREAD =     $04
                =0008       OWRIT =     $08
                =0088       EOF =       $88
                =E456       CIOV =      $E456
                =0040       IOCB4 =     $40
                            ;
                            ;Als erstes die Ein- und
                            ;Ausgabe aktivieren.
                            ;
0000#           =5000                   ORG $5000

                            ;Datenteil
5000            44323A5445;NAME1        DB          'D2:TEST1',EOL
                =0050       BUF1SZ      =           80
                =5009       BUF1        =           *
5009            =5059                   ORG         *+BUF1SZ
5059            50323A9B    NAME2       DB          'P2:',EOL
505D            A230        START       LDX         #IOCB3
505F            A900                    LDA         #LOW NAME1
5061            9D4403                  STA         ICBAL,X
5064            A950                    LDA         #HIGH NAME1
5066            9D4503                  STA         ICBAH,X
5069            A900                    LDA         #0
506B            9D4B03                  STA         ICAX2,X
                            ;
                            ;"Oeffne" die Diskettenstation
                            ;
```

```
506E    A903            LDA     #OPEN
5070    9D4203          STA     ICCOM,X
5073    2056E4          JSR     CIOV
5076    BC4303          LDY     ICSTA,X
5079    1003 ^507E      BPL     L1
507B    4CA250          JMP     ERR2
                ;
                ;Ueber Kanal 4 wird der Drucker
                ;angesprochen
                ;
507E    A240    L1      LDX     #IOCB4
5080    A959            LDA     #LOW NAME2
5082    9D4403          STA     ICBAL,X
5085    A950            LDA     #HIGH NAME2
5087    9D4503          STA     ICBAH,X
508A    A908            LDA     #WRIT
508C    9D4A03          STA     ICAX1,X
508F    A900            LDA     #0
5091    9D4B03          STA     ICAX2,X
                ;
                ;"Oeffne" den Drucker
                ;
5094    A903            LDA     #OPEN
5096    9D4203          STA     ICCOM,X
5099    2056E4          JSR     CIOV
509C    BC4303          LDY     ICSTA,X
509F    1000 ^50A5      BPL     TP10
                ;
                ;Fehlerabbruch
                ;
50A1    00      ERR1    BRK
50A2    00      ERR2    BRK
50A3    00      ERR3    BRK
50A4    00      ERR4    BRK
                ;
                ;Vorbereitung zum Lesen des Daten-
                ;satzes
                ;
50A5    A230    TP10    LDX     #IOCB3
50A7    A905            LDA     #GETREC
50A9    9D4203          STA     ICCOM,X
50AC    A909            LDA     #LOW BUF1
50AE    9D4403          STA     ICBAL,X
50B1    A950            LDA     #HIGH BUF1
50B3    9D4503          STA     ICBAH,X
                ;
                ;Lesen des Datensatzes
                ;
50B6    A950    LOOP    LDA     #LOW BUF1SZ
50B8    9D4803          STA     ICBLL,X
50BB    A900            LDA     #HIGH BUF1SZ
50BD    9D4903          STA     ICBLH,X
50C0    2056E4          JSR     CIOV
50C6    1004
```

```
                        ;
                        ;Schliesse den Lesekanal-EOF
                        ;
50C8        C088        TP20        CPY         #EOF
50CA        D0D7 ^50A3              BNE         ERR3
                        ;
                        ;Drucke den Datensatz
                        ;
50CC        BD4803      PRNTR       LDA         ICBLL,X
50CF        A240                    LDX         #IOCB4
50D1        9D4803                  STA         ICBLL,X
50D4        A230                    LDX         #IOCB3
50D6        BD4903                  STA         ICBLH,X
50D9        A240                    LDX         #IOCB4
50DB        9D4903                  STA         ICBLH,X
50DE        A909                    LDA         #PUTREC
50E0        9D4203                  STA         ICCOM,X
50E3        A909                    LDA         #LOW BUF1
50E5        9D4403                  STA         ICBAL,X
50E8        A950                    LDA         #HIGH BUF1
50EA        9D4503                  STA         ICBAH,X
50ED        2056E4                  JSR         CIOV
50F0        BC4303                  LDY         ICSTA,X
50F3        1003 ^50F8              BPL         L3
50F5        4CA450                  JMP         ERR4
50F8        A230        L3          LDX         #IOCB3
50FA        BC4303                  LDY         ICSTA,X
50FD        C088                    CPY         #EOF
50FF        F003 ^5104              BEQ         L2
5101        4CA550                  JMP         TP10
5104        A90C        L2          LDA         #CLOSE
5106        9D4203                  STA         ICCOM,X
5109        2056E4                  JSR         CIOV
510c        A90C                    LDA         #CLOSE
510E        A230                    LDX         #IOCB3
5110        9D4203                  STA         ICCOM,X
5113        2056E4                  JSR         CIOV
5116        00                      BRK
5117                                END

Keine Fehler,39 Labels,$A3E6h frei
```

BUF1	5009	1#36	2/28	2/30	2/60	3/ 2		
BUF1Sz	0050	1#35	1/37	2/35	2/37			
CIOV	E456	1#25	1/51	2/12	2/39	3/ 4	3/18	3/23
CLOSE	000C	1#21	3/16	3/20				
EOF	0088	1#24	2/45	3/12				
EOL	009B	1# 3	1/34	1/38				
nERR1	50A1	2#18						
ERR2	50A2	1/54	2#19					
ERR3	50A3	2#20	2/46					
ERR4	50A4	2#21	3/ 8					
GETREC	0005	1#19	2/26					
ICAX1	034A	1#15	1/16	2/ 4				
ICAX2	034B	1#16	1/45	2/ 6				
ICBAH	0345	1#10	1/11	1/43	2/ 2	2/31	3/ 3	
ICBAL	0344	1# 9	1/10	1/41	1/60	2/29	2/61	
ICBLH	0349	1#14	1/15	2/38	2/54	2/56		
ICBLL	0348	1#13	1/14	2/36	2/50	2/52		
ICCOM	0342	1# 7	1/ 8	1/50	2/11	2/27	2/59	3/17
		3/22						
ICDNO	0341	1# 6	1/ 7					
ICHID	0340	1# 5	1/ 6					
ICPTH	0347	1#12	1/13					
ICPTL	0346	1#11	1/12					
ICSTA	0343	1# 8	1/ 9	1/52	2/13	2/40	3/ 6	3/11
IOCB3	0030	1# 4	1/39	2/25	2/53	3/10	3/21	
IOCB4	0040	1#26	1/58	2/51	2/55			
L1	507E	1/53	1#58					
L2	5104	3/13	3#16					
L3	50F8	3/ 7	3#10					
nLOOP	50B6	2#35						
NAME1	5000	1#34	1/40					
NAME2	5059	1#38	1/59	1/61				
OPEN	0003	1#18	1/49	2/10				
nOREAD	0004	1#22						
OWRIT	0008	1#23	2/ 3					
PRNTR	50CC	2/41	2#50					
PUTREC	0009	1#20	2/58					
nSTART	505D	1#39						
TP10	50A5	2/14	2#25	3/14				
nTP20	50C8	2#45						

Formate der Symbole

Wenn die Option R=S gewaehlt wurde, wird die kurze Referenzuebersichtstafel am Ende des Programms ausgegeben. Fuer irgendeinen Namen in dem Programm wird folgendes ausgegeben:

sa Symbol hhhh, heissen:
<s>ist ein Leerzeichen oder "s" steht fuer einen Namen der im System Textfile eingefuehrt wurde.
<a>ist entweder ein Leerzeichen oder
U= undefiniert oder
D= doppelte Definition oder
n= nicht benutzt
<symbol> ist der Name des Symbols.

<hhhh> ist der Wert des Symbols in hexadezimal, oder wenn der Name "mac" ist, ist es der Macro. Vier Symbole werden in jeder Zeile ausgegeben, wenn die Zeilenlaenge nicht ueberschritten wird.

Wenn die Option R=F gewaehlt wurde, wird die vollstaendige Referenzuebersichtstafel ausgegeben. In jeder Zeile werden zusaetzlich zu der Information wie sie bei R=S geliefert wird, Querverweise ausgegeben. Jeder Verweis hat folgende Form:

ppp/II

wobei <ppp> die Seitennummer und <II> die Zeilennummer bedeutet. Fuer die Definition wird / durch # ersetzt.

Namen, die mit einem Doppelpunkt (lokale Symbole) oder einem Fragezeichen (gewoehnlich in einem Macro eingefuehrt) beginnen, erscheinen in keiner der beiden Symbol-Tabellen.

Symbole, die in einer System-Datei definiert wurden, erscheinen in der Querverweisliste nur dann, - angezeigt durch s - wenn sie bei der Assemblierung auch benutzt wurden.

4 Aufbau der Sprache

Ein Macro Assembler Quellprogramm besteht aus einer Anzahl von Befehlen, Kommentaren und Definitionen. Befehle sind die fundamentalen Einheiten von Assemblerprogrammen. Kommentare werden vom Assembler nicht beachtet und erzeugen keinen Objektcode. Definitionen koennen bedingt assembliert, zum spaeteren Assemblieren gespeichert oder wiederholt werden.

Alle Buchstaben des Programms werden in Grossbuchstaben umgewandelt, ausgenommen die des Kommentarfeldes.

Befehle

Befehle sind in drei Felder unterteilt: Markenfeld, Operationsfeld und Variablenfeld.

Markenfeld

Das Markenfeld beginnt mit dem ersten Zeichen eines Befehls und endet mit einem Leerzeichen oder dem Ende des Befehls. Ein Doppelpunkt am Ende einer Marke wird ignoriert.

Beispiel:

```
        SYMBX:   ADC       MEM,X                ;Kommentar
```

SYMBX ist die Marke.

Operationsfeld

Das Operationsfeld beginnt mit dem ersten, vom Leerzeichen verschiedenen Zeichen, hinter dem Markenfeld und endet mit dem naechsten Leerzeichen. Maschinen-Mnemoniks, Pseudo-Operationen und Macro Aufrufe koennen im Operationsfeld erscheinen. Wenn dieses Feld leer ist, muss das Variablenfeld ebenfalls leer sein.

Beispiel:

```
        SYMBX:   ADC       MEM,X                ;Kommentar
```

ADC ist das Maschinen-Mnemonik

Variablenfeld

Das Variablenfeld beginnt mit dem ersten Zeichen hinter dem Operationsfeld und endet mit dem Ende des Befehls. Variable, Ausdruecke und andere Argumente, die vom Operationsfeld benutzt werden, erscheinen in diesem Feld.

Beispiel:

```
        SYMBX:   ADC       MEM,X                ;Kommentar
```

MEM,X ist die Variable.

Beendigung des Befehls

Ein Befehl wird beendet durch

- den Beginn eines Kommentars (;) oder
- durch das Zeilenende oder
- durch das Zeichen fuer das logische Ende eines Befehls (!).

```
SYMBX:   ADC       MEM,X                ;Kommentar
SYMBY:   ADC       MEM,X
SYMBZ:   ASL       ! ASL     ! ASL      ! ASL ;4Befehle
```

Im letzten Beispiel (SYMBZ) enthaelt eine Quellzeile vier Befehle. Drei von ihnen werden durch ein Ausrufezeichen beendet, der letzte durch ein Semicolon. Der gleiche Objektcode wuerde erzeugt werden, wenn das Ausrufezeichen durch ein Zeilenende ersetzt werden wuerde.

Wenn Ausrufezeichen und Semicolon in Anfuehrungszeichen stehen, wirken sie nicht als Trennungszeichen.

Kommentare

Ein Kommentar folgt dem Variablenfeld, beginnend mit einem Semicolon. Ein Kommentar wirkt weder auf die Operation des Assemblers noch auf den erzeugten Objektcode. Kommentare, die in Spalte 1 beginnen werden Ganzzeilenkommentare genannt. Sie beginnen entweder ebenfalls mit einem Semicolon oder mit einem Stern (ein Stern markiert nur dann einen Kommentar, wenn er in Spalte 1 steht. Spalte 1 der Eingabe erscheint im Ausgabe-Listing in Spalte 21).

```
Ein Kommentar wird beendet durch das Ende der Zeile.
LABEL:    LDA     120             ;Dies ist ein Kommentar
;Dies ist ein Ganzzeilenkommentar
*Dies ist auch ein Ganzzeilenkommentar
FROG:     STA     MEM,X           Dies ist als Kommentar verboten
                                  ;(in obiger Zeile fehlt ein ;)
```

Definitionen

Definitionen beginnen mit besonderen Typen von Befehlen (MACRO, ECHO, IF). Ihr Ende haengt davon ab, wie sie begonnen haben: ENDM ist das Ende einer MACRO- und einer ECHO-Definition, ENDIF beendet einen IF-Bereich.

Symbole und Namen

Ein Symbol besteht aus einer Anzahl von Zeichen, die einen Wert oder ein Macro bezeichnen. Das erste Zeichen darf keine Ziffer sein.

Symbole duerfen beliebig lang sein, die ersten sechs Zeichen werden aber nur beachtet. Symbole die laenger sind, muessen sich also in den ersten sechs Zeichen voneinander unterscheiden. Folgende Zeichen duerfen fuer ein Symbol verwendet werden:

A-Z	Die Grossbuchstaben des Alphabets
a-z	Die Kleinbuchstaben des Alphabets (werden in Grossbuchstaben umgewandelt.
:	Darf nur das erste Zeichen sein und zeigt ein lokales Symbol an.
?	Nur als erstes Zeichen. Solche Symbole werden nicht in die Symbol-Tabelle uebernommen.
§	Zusaetzliches Alpha-Zeichen. Darf nicht als erstes
0-9	Ziffern. Nicht als erstes Zeichen.

Das Unterstreichungszeichen (_) darf verwendet werden, wird aber ignoriert. Kleinbuchstaben werden in die entsprechenden Grossbuchstaben umgewandelt. Taucht ein Doppelpunkt am Anfang eines Namens auf, bezeichnet dies ein lokales Symbol innerhalb einer Prozedur (siehe PROC Pseudooperation, in Kapitel 6). Ein Doppelpunkt am Ende wird als End-Zeichen der Marke angesehen, an jeder anderen Stelle ignoriert.

Beispiel:

```
                                  ;_ wird ignoriert, Marke heisst ERROR5
          JMP       RESTART       ;die ersten sechs Zeichen
                                  ;werden benutzt: 'RESTAR'
TEST      LDA       COUNT
          BNE       Error5        ;umgewandelt in ERROR5
:LOCAL:   DEC                     ;:LOCAL: ist ein lokales Symbol
```

Zahlen

Zahlen koennen in einer der folgenden Formen angegeben werden.

Vorzeichen	Basis	
%	2	Binaer
@	8	Oktal
$	16	Hexadezimal

Fehlen des Vorzeichen bedeutet Dezimal

Ziffern, die groesser als die Basis sind, sind verboten. Das Unterstreichungszeichen wird ignoriert.

Der Macro Assembler kann reelle Zahlen in die 6-Byte Darstellung des ATARI-Basic umwandeln. Reelle Zahlen sind keine richtigen Argumente in Ausdruecken fuer Variablen-Felder (siehe REAL6 in Kapitel 6).

Beispiel:

```
BINVAL  EQU     %10 001 010
OCTVAL  EQU     @212
HEXVAL  EQU     $8A
```

Zeichenketten

Der Assembler nimmt ATASCII-Zeichen von $20 bis $7E als gueltige Zeichen an. Eine Zeichenkette besteht aus einer Anzahl beliebiger Zeichen, eingerahmt in einfache Anfuehrungszeichen ('n...n').

Innerhalb einer Zeichenkette wird ein einzelnes Anfuehrungszeichen durch zwei aufeinander folgende Anfuehrungszeichen dargestellt. Zeichenketten werden in TITLE- und SUBTTL-Befehlen, als Unterfelder in DC und DB, und als Operanden von Relationen gebraucht. Die Operation LSTR liefert die Laenge einer Zeichenkette an (siehe 'Ausdruecke' in diesem Kapitel).

Beispiele:

```
TITLE       'Probe Ausdruecke'
DB          'Dies ist eine Zeichenkette.',$9B
DB          'Control-Zeichen sind in einer Zeichenkette verboten'
DB  $9B     ;unsichtbare Zeichen koennen durch ihre ATASCII-Werte
            ;dargestellt werden. z.B. $96 fuer EOL
DW  $2766,  'bp','BP'; 2 Byte-Werte
LDA #43     ;eine Dezimalzahl
ADC #'C'    ;ein ATASCII-Zeichen
CMP #''     ;ein ATASCII-Zeichen
```

Ausdruecke

Ein Ausdruck besteht aus Operanden, verbunden durch Operatoren, die einen Wert erzeugen. Operatoren gleicher Prioritaet, werden nacheinander von links nach rechts ausgefuehrt. Zum aendern der Prioritaet koennen eckige Klammern verwendet werden, da runde Klammern vom 6502 fuer indirekte Adressierung belegt sind. Ausdruecke werden mit der 16-Bit-Zweierkomplement-Arithmetik berechnet. Ein Ueberlauf wird ignoriert. Reelle Zahlen sind daher keine gueltigen Argumente in Ausdruecken.

Beispiele:

```
DB          'hier einige denkbare Ausdruecke:'
DB          43+22 shl 3 mod 6
DB          'Q'+REF1 xor [99 und REF2]
AND         low[['ZZ'-['A'xor'a'+['A'xor'a'] shl 8]]
DW          rev[*0-*L]
```

Operanden

Ein Operand ist entweder ein Symbol, ein in eckige Klammern gesetzter Ausdruck, eine Zahl, eine Zeichenkette oder eines der folgenden speziellen Elemente:

```
*     = laufender Adressenzaehler
*L    = dasselbe wie *
*O    = laufender Wert des Ursprungszaehlers
*P    = laufender Wert des Bit-Zaehlers
```

Fuer weitere Informationen ueber *L siehe LOC-, ueber *O siehe ORG-, ueber *P siehe VFD-Pseudooperationen.

Die Relationen (<,>,...) liefern einen Wert von 0 fuer 'nicht wahr' und einen Wert von $FFFF fuer 'wahr'.

Zahlenvergleiche behandeln jeden Wert als nicht vorzeichenbehaftet, d.h. positiv, so dass (-1<0) die Antwort 'nicht wahr' liefert. Vergleiche von Zeichenketten werden mit Hilfe der ATASCII-Werte durchgefuehrt.

OPERATOREN

+	Addition oder positives Vorzeichen
-	Subtraktion oder negatives Vorzeichen
*	Multiplikation
/	Division
NOT	Einer-Komplement
AND, &	logisches Produkt, Konjunktion
OR	logische Summe, Disjunktion
XOR	logische Differenz, Antivalenz, exklusives Oder
= , EQ	gleich
<>, NE	ungleich
< , LT	kleiner
<=, LE	nicht groesser
> , GT	groesser
>=, GE	nicht kleiner
SHL n	logische Verschiebung um n Bits nach links
SHR n	logische Verschiebung um n Bits nach rechts
HIGH	High-Byte des 16-Bit Wertes, x/256
LOW	Low-Byte des 16-Bit Wertes, x MOD 256
MOD	Modulo-Funktion, Divisionsrest
REV	[[LOW x]SHL 8]+[HIGH x]
DEF	Definition eines Symbols testen
LSTR	Laenge einer Zeichenkette feststellen

Rechenprioritaeten

```
hoechste        eckige Klammern
                HIGH  LOW  DEF  REV  LSTR
                 *     /   MOD  SHL  SHR
                 +     -   (Vorzeichen)
                 +     -   (Verknuepfungen)
                 =    <>   <    <=   >    >=
                NOT
                AND
niedrigste       OR    XOR
```

5 Macros

Ein Macro ist eine Folge von Quell-Befehlen, die zunaechst gespeichert, und spaeter - beim Macro Aufruf - , assembliert werden. Ein Macroaufruf ist ein Verweis im Operationsteil eines Befehls, zum Macronamen. Oft enthaelt ein Macroaufruf aktuelle Parameter, die gegen die formellen Parameter im Macro ausgetauscht werden, so dass der Objektcode eines Macros von Aufruf zu Aufruf verschieden sein kann.

Definition eines Macros

Ein Macro besteht aus drei Teilen: Kopf, Rumpf und Endbefehl.

Kopf

Ein Macro beginnt mit dem Namen des Macros, dem Schluesselwort MACRO und den aktuellen Parametern im Variablenfeld.

Rumpf

Der Rumpf beginnt mit dem ersten Befehl hinter dem Kopf, und enthaelt verschiedene Anweisungen. Alle Anweisungen ausser END, einschliesslich anderer Macrodefinition und -aufrufe sind innerhalb des Rumpfes erlaubt. Aber eine Definition innerhalb einer Definition ist ungueltig solange die aeussere Definition nicht aufgerufen wurde. Die innere Definition kann also nicht direkt aufgerufen werden. Formale Parameter werden im Rumpf mit einem %-Zeichen und der Nummer aufgerufen, in der sie im Kopf stehen: %1 = erster Parameter
%2 = zweiter Parameter
...
%9 = neunter Parameter
%K = vierstellige Hex-Zahl, die den aktuellen Aufruf des Macros darstellt
%L = das Markenfeld des Macros
%M = der Name des Macros
%% = wird ersetzt durch ein einzelnes Prozentzeichen

Endbefehl

Ein Macro wird durch die Pseudooperation ENDM beendet. Der Assembler zaehlt die Schachtelungstiefe von MACRO/ECHO - ENDM - Paaren, die in einem Macrorumpf auftauchen, so dass eine Definition nur von dem entsprechenden ENDM beendet wird.
Wichtig: ENDM muss ein Tab-Zeichen vorangestellt werden.
Dazu <ESC><CLR SET TAB> druecken.

Aufruf eines Macros

Ein zuvor definiertes Macro wird aufgerufen, wenn sein Name im Operationsfeld eines Befehls auftaucht. Wenn aktuelle Parameter im Aufruf vorhanden sind, werden sie ohne Berechnung gegen die entsprechenden formalen Parameter im Macrorumpf ausgetauscht. Erst nachdem das im ganzen Rumpf geschehen ist, wird die Assemblierung wieder aufgenommen. Daher koennen Befehle, die ein Macro erzeugen, ihrerseits wieder Macro Aufrufe oder Definitionen enthalten. Die Schachtelungstiefe wird nur durch den verfuegbaren Speicherplatz begrenzt.

Wichtig: Es muss beim Schreiben recursiver Macros auf eine richtige Abbruchbedingung geachtet werden. Ein Macro, das sich unendlich oft selbst aufruft, veranlasst den Assembler (eventuell) mit der Nachricht " Memory Overflow " abzubrechen.

Codevervielfachung

Die Pseudooperation ECHO wird benutzt um Code-Reihen zu wiederholen. Er wird ebenso geschrieben wie der Macro-Befehl mit folgenden Unterschieden:

Im Kopf heisst es ECHO statt MACRO; statt der formalen Parameter steht im Variablenfeld ein Ausdruck, der angibt wie oft der Rumpf wiederholt werden soll. ENDM wird auch benutzt um ein ECHO zu beenden.

Verschachtelung

ECHO-, MACRO- und IF-Bloecke duerfen in jeder denkbaren Weise verschachtelt werden, mit der einzigen Auflage, dass jeder Block vollstaendig innerhalb des uebergeordneten Blocks liegt.

6 Pseudooperationen

Der Macro Assembler vefuegt ueber einen umfassenden Satz von Pseudooperationen, die eine Einflussnahme auf die Assemblierung erlauben.

Zum leichteren Verstaendnis werden in diesem Handbuch folgende Abkuerzungen verwendet:

iglab	das Marken- oder Labelfeld wird ignoriert.
<exp>	ein Ausdruck muss erscheinen
[exp]	ein Ausdruck darf erscheinen

ASSERT

Ueberprueft Assemblierungsbedingungen

```
iglab     ASSERT    <exp>
dabei sind:    iglab = ignoriertes Marken- oder Labelfeld
               exp   = irgendein richtiger Ausdruck:
                       ungleich null heisst 'wahr'
                       gleich null heisst 'unwahr'
```

ASSERT erlaubt es, nach nicht logischen Bedingungen zu suchen und sie anzuzeigen,z.B. unrichtige Werte von Parametern, zu grosse Programmlaengen, nicht definierte Symbole usw. Der Ausdruck wird berechnet, und ein P-Fehler gemeldet, wenn der Ausdruck unwahr d.h. null ist.

Der Ausdruck wird im ersten Durchlauf des Assemblers nicht beachtet, so dass jede Bedingung richtig geprueft werden kann. Vorwaertsverweise im Ausdruck werden also auch richtig berechnet.

DB

Byte erzeugen

LABEL: DB <exp>,...,<exp>

<exp>=irgendein richtiger Ausdruck,Wert oder eine Zeichenkette

DB erlaubt es, direkt den Inhalt von Speicherzellen anzugeben.

Eine Zeichenkette erzeugt soviele Bytes wie sie Zeichen hat, das erste Zeichen wird das erste erzeugte Byte. Jedes Zeichen erzeugt seinen 7-Bit ATASCII-Code ohne Paritaet.

DB wird benutzt, um Codes mit Textketten und Datentabellen einzustreuen.

Das Label vor DB gibt die Adresse des ersten Bytes an. Beispiele:

```
PNCHRS:   DB   ',./;@@<>?+!''#$%&''()_*=+(tm):-[]@',0
          DB   $80
          DB   LAB,LAB2,3,$46,$0AF,'xX',17+QVAL*4,'coffee'
```

DC

Erzeuge Zeichen

LABEL: DC <exp>,...,<exp>

<exp> irgendein richtiger Ausdruck, Wert, oder eine Zeichenkette.

DC arbeitet genauso wie DB, nur das hoechste Bit (Paritaetsbit) des letzten Bytes jeden Ausdruckes wird erzeugt.

DC wird genauso verwendet wie DB, der einzige Unterschied ist das Paritaetsbit des letzten Bytes eines Terms.

Beispiel:

```
TBLHDR    DC   'Dies ist ein Text'
ADRLST    DC   128,$36,$15,@21,159
```

DS

Erzeuge Zwischenraum

LABEL: DS <exp16>

<exp16> = irgendein richtiger 16-Bit Ausdruck, Wert oder eine Zeichenkette

DS erlaubt es, groessere Bloecke von Speicherplaetzen zu reservieren. Der Ausdruck <exp16> wird berechnet und als positive 16-Bit Zahl aufgefasst. Um diese Zahl wird der Stand von Ursprungs- und Adressenzaehler erhoeht. Uebersprungener Speicherplatz wird nicht geaendert, und enthaelt also bei der Ausfuehrung des Programms unbekannte Werte. Das Label vor DS bezeichnet das erste uebersprungene Byte.

DS wird benutzt um Datenspeicherplatz fuer den Gebrauch bei Ausfuehrung des Programms zu reservieren, oder um ein ROM-Bereich zu ueberspringen.

Beispiel:

```
STORG:    DS   256                    ;256 Bytes zuteilen
```

DW

Erzeuge Datenworte

LABEL: DW <exp16>,...,<exp16>

<exp16> = irgendein Ausdruck oder Wert oder eine Zeichenkette von ein bis zwei Zeichen

DW bestimmt den Inhalt von Bloecken von Speicherplatz. Werte und Ausdruecke im Variablenfeld werden in 16-Bit Werte verwandelt und im Speicherplatz als Datenworte untergebracht. Der Assembler legt zuerst das Low- und dann das High-Byte ab.

Das Label erhaelt den Wert der Adresse des ersten erzeugten Bytes.

DW ist zum Erzeugen von Tabellen von 16-Bit-Werten bestimmt.

Beispiele:

```
                                        Adressentabelle:
   DW   PWRON                           ;Einschalten
   DW   MSTRST                          ;Hauptreset
   DW   SYSCAL                          ;Systemanpassung
   DW   RECAL                           ;Wiederanpassung
   DW   PWRDN                           ;Ausschalten
   DW   BUTTON                          ;Knopf druecken
   DW   EMERG                           ;Notstillegung
   DW   ACTN1, ACTN2, ACTN3             ;Aktionsnummern 1,2,3
   DW   0                               ;Ende der Tabelle
```

ECHO...ENDM
Echoblock

```
   LABEL:    ECHO <exp>
             ...
             ENDM
```

<exp> = numerischer Ausdruck

ECHO...ENDM gibt die Moeglichkeit zur einfachen Codevervielfachung. Codes zwischen ECHO und ENDM werden so oft assembliert wie <exp> angibt.

Das Label adressiert den Zaehlerstand von #0 zu dem Zeitpunkt, an dem der ECHO-Befehl aufgefunden wird.

Eine ECHO...ENDM Konstruktion darf 255 Wiederholungen nicht ueberschreiten.

Null Wiederholungen heisst, der ECHO...ENDM Code wird uebersprungen. ECHO...ENDM ist eine bequeme Einrichtung zum Wiederholen von Codes. Einen ECHO-Block zu schreiben und zu speichern ist erheblich einfacher, als z.B. 127 mal eine 6 Zeilen lange Prozedur.

Wichtig: Der ENDM-Pseudooperation muss ein Tab-Zeichen vorausgehen.

Beispiel:

```
   ;folgendes Beispiel erzeugt eine Tabelle mit 20 Durchgaengen
   ;zu je 4 Bytes und setzt jeden Durchgang auf den Wert
   ;$10 37 00 00
   TAFEL:    ECHO 20                   ;20 mal
             DB   $10, $37, $00, $00
             ENDM                      ;Ende der Tabelle
```

EJECT
Seitenvorschub

```
   iglab     EJECT
```

EJECT erzwingt einen Seitenvorschub im Listing, wenn das Listing in dem Augenblick eingeschaltet ist. Es kann ueberall im Quellprogramm erscheinen. Die TITLE-Pseudooperation aendert die Seitenueberschrift und erzwingt ebenfalls ein EJECT.

Beispiel:

```
   EJECT
```

END
Ende des Programms

```
   LABEL:    END  [exp]
```

END teilt dem Assembler mit, wo er mit dem Assemblieren aufhoeren und die Symboltabelle anfangen soll. Der moegliche Ausdruck im Variablenfeld gibt die Startadresse fuer das Objektprogramm an.
END muss die letzte Anweisung der letzten LINK-Datei in einer Assemblierung sein. Dem Label vor END wird der Wert des *O-Zaehlers beim Bearbeiten des END-Befehls zugewiesen.
Beispiel:

```
FREESP:   END                    ;Ende des Programms
```

EQU oder =
Wertzuweisung zu einem Symbol

```
LABEL:    EQU  <exp16>
LABEL     =    <exp16>
<exp> = 16-Bit-Ausdruck oder -Wert oder eine Kette von ein
                oder zwei Zeichen
```

EQU weist dem LABEL auf der linken Seite den 16-Bit-Wert auf der rechten Seite zu. EQU legt ein Symbol (LABEL) fuer den Gebrauch in anderen Assembleranweisungen fest. Anders als SET, setzt EQU einen festen Wert fuer das Symbol fest, der bei der Assemblierung nicht wieder geaendert werden kann.
Der Operand <exp16> muss zum Zeitpunkt der Berechnung einen festen Wert haben. Alle Symbole, die in dem Ausdruck benutzt werden, muessen vorher definiert worden sein.
Beispiel:

```
TSTCHR    EQU  '$'
TS2CHR    EQU  '@'
ZAP       EQU  $900
ZONK       =   ZAP*2
```

ERR

ERR ermoeglicht es, eine Fehlermeldung zu erzwingen. Das Variablenfeld wird ignoriert. Wenn der Assembler eine unmoegliche oder unerwuenschte Bedingung beim Assemblieren findet, kann ERR dies signalisieren.
Beispiele:

```
IF   *> 4000h
ERR                ;Programm zu lang
ENDIF
```

IF...ENDIF,IF...ELSE...ENDIF

```
iglab     IF   <exp>
          <Code fuer besondere Situationen>
iglab     ENDIF

iglab     IF   <exp>
          <Quellcode>
iglab     ELSE
          <Quellcode>
iglab     ENDIF
```

<exp> = Ausdruck: ungleich Null = wahr, gleich Null = unwahr

IF...ENDIF und IF...ELSE...ENDIF kontrollieren welcher Quellcode an den Assembler gegeben wird. Beim Assemblieren wird <exp> berechnet, und das Ergebnis entscheidet, wo die Assemblierung fortgesetzt wird.

Immer wenn ein Programm in zwei oder mehreren Versionen moeglich sein soll, koennen IF...ENDIF und IF...ELSE...ENDIF Werte, die bei der Assemblierung auftreten, pruefen und nur die geeignete Version assemblieren lassen. Die Werte des Ausdrucks <exp> muessen numerisch sein; Zeichenketten duerfen nicht laenger sein als zwei Zeichen.

IF...ENDIF und IF...ELSE...ENDIF-Konstruktionen koennen in beliebiger Tiefe verschachtelt werden. Begrenzt wird die Tiefe nur durch den verfuegbaren Speicherplatz. Jedes Label im Labelfeld wird ignoriert. Zur besseren Lesbarkeit koennen dort aber Namen untergebracht werden.

Beispiele:

```
        IF   1                          ;1 ist nicht 0, also wahr
        JSR  OUTM
        JMP  BOOT                       ;diese beiden Zeilen werden
                                        ;assembliert
        ENDIF
LABEL:  IF   DEF X                      ;Bedingung
        JSR  PATH 1                     ;LABEL wird ignoriert,ver-
LABEL:  ELSE                            ;bessert aber die Lesbarkeit
        JMP  PATH2
        ENDIF
```

INCLUDE
Einschliessen anderer Quellprogramme
Label: INCLUDE <Dateispez>
<Dateispez> = <Dn:Dateiname.Ext>
n=1,...,4

Das Argument von INCLUDE gibt eine andere Datei an, die in die Assemblierung der Datei eingeschlossen werden soll, in der dieser Befehl steht. Die INCLUDE-Datei kann ihrerseits wieder INCLUDE-Befehle enthalten. Das Listing der INCLUDE-Dateien wird von der I-Option der LIST-Pseudooperation gesteuert.

INCLUDE erlaubt es, groessere Programme in handliche Abschnitte zu unterteilen, um das Editieren zu erleichtern, Programmbibliotheken anzulegen, usw.

Beispiel:
Die Zeile D:INCLDEX.ASM
i.: Zusammenhang mit folgenden Dateien
<Inhalt von INCLDEX.ASM>

```
            TITLE     'INCLUDE-Beispiel'
            ORG       $100
            INCLUDE   D:L1
            INCLUDE   D:L2
            INCLUDE   D2:L3.ACD
;*** Ende INCLDEX.ASM
```

<Inhalt von D:L1>

```
            LDA       L1VAL
;*** Ende L1.ASM
```

<Inhalt von D:L2>

```
            LDA       L2VAL
;*** Ende L2.ASM
```

<Inhalt von D2:L3.ACD>

```
L1VAL       DB
L2VAL       DB        0
            END       ;Assemblierung beendet
;*** Ende L3.ACD
```

wuerde vom Assembler wie folgt gelesen werden:

```
            TITLE     'INCLUDE Beispiel'
            ORG       $100
            LDA       L1VAL
;*** Ende L1.ASM
            LDA       L2VAL
;*** Ende L2.ASM
L1VAL       DB        '*'
L2VAL       DB        0
            END       ;Assemblierung beendet
;*** Ende L3.ACD
;*** Ende INCLDEX.ASM
```

LINK
Verkettung von Programmen
  iglab LINK <Dateispez>
  <Dateispez> = <Dn:Dateiname.Ext>
            n = 1,...,4

Die LINK-Pseudooperation wirkt aehnlich wie INCLUDE. Eine Datei, die einen LINK-Befehl enthaelt, wird jedoch bei der Assemblierung nicht unterbrochen, sondern vollstaendig abgearbeitet, bevor der LINK-Befehl ausgefuehrt wird.
Dateien die durch LINK aufgerufen werden, koennen ihrerseits LINK-Befehle enthalten; die Reihenfolge der Abarbeitung ist dann dieselbe wie bei INCLUDE. Die Tiefe der Verschachtelung ist beliebig, solange die Gesamtzahl der Dateien 40 nicht ueberschreitet.
Seien A,B,C,D,E,F Quellprogramme, und A enthalte der Reihenfolge nach LINK-Befehle zu den Dateien B,C,D, und C enthalte einen LINK-Befehl zu E, und E zu F, dann ist die Reihenfolge der Abarbeitung A,B,C,E,F,D.
Die Extension der Dateispezifikation darf fehlen. Es wird dann dieselbe Extension angenommen, die das Programm hat, in der der LINK-Befehl steht.

Beispiele:

```
        LINK    D2:PART1        ;assembliere Datei 'D2:PART1'
                                ;benutze dieselbe Extension wie
                                ;im Hauptprogramm
        LINK    D:UTIL.ACD
BLORP:  LINK    D2:PART2.ASM    ;'BLORP' wird nicht beachtet
```

Der LINK-Befehl erlaubt, wie der INCLUDE-Befehl, groessere Programme in handliche Abschnitte zu unterteilen, um das Editieren zu erleichtern und Programmbibliotheken anzulegen. Der LINK-Befehl erlaubt aber auch, Programme ueber verschiedene Disketten zu verketten. Das gesamte Quellprogramm muss also nicht auf einer Diskette stehen.

Beispiel:
Die Zeile AMAC D:LINKE6.ASM
wird in Zusammenhang mit den folgenden Dateien:

<Inhalt von LINKE6.ASM>

```
            TITLE           'LINK Beispiel'
            ORG             $100
            LINK            D:L1
            LINK            D:L2
            LINK            D2:L3.ACD
;***        Ende x          LINKE6.asm
```

<Inhalt von D:L1>

```
            LDA             L1VAL
;***        Endex           L1.asm
```

```
<Inhalt von D:L2>
                LDA             L2VAL
;***            Endex           L2.asm

<Inhalt von D2:L3.ACD>
L1VAL           DB              '*'
L2VAL           DB              0
                END             ;Assemblierung beendet
```

vom Assembler wie folgt gelesen:

```
                TITLE           'LINK Beispiel'
                ORG             $100
;***            Endex           LINKEG.asm
                LDA             L1VAL
;***            Endex           L1.asm
                LDA             L2VAL
;***            Endex           L2.asm
L1VAL           DB              '*'
L2VAL           DB              0
                END             ;Assemblierung beendet
;***            Endex           L3.acd
```

```
LIST
Steuerung des Listings
iglab LIST  *
iglab LIST  <opt>...,<opt>
<opt>= einer der folgenden Buchstaben, gegebenenfalls mit
       vorangestelltem Minuszeichen
```

C Listing mit EJECT, PAGE, SPACE, SUBTTL und TITLE-Zeilen. Keine Angabe = AUS

D detaillierten Code ausdrucken, d.h. jedes Byte, das von DB,DW, VFD, mehrzeiligen Statements usw. erzeugt wurde, mitausgeben.

F Codes, die durch IF...ENDIF oder IF...ELSE...ENDIF uebersprungen werden, mit ausgeben. Keine Angabe = EIN

G Jeden erzeugten Code ausgeben, d.h. jedes Byte ausgeben, das in die Objekt-Datei kommt, ohne Ruecksicht auf die Herkunft. G ueberschreibt -L. Keine Angabe = AUS

I Codes aus INCLUDE-Dateien mit ausgeben. Keine Angabe = AUS

L Generell drucken. Ist -L gesetzt, so wird nichts ausgegeben, ausgenommen fehlerhafte Zeilen, oder wenn -L durch G ueberschrieben wurde. Keine Angabe = EIN

M Alle Zeilen die durch Macros erzeugt wurden mit ausgeben.
   Keine Angabe = EIN

R Querverweise sammeln. Keine Angabe = EIN

S Gib die gesammelten Verweise in eine 'systext'-Datei
   Keine Angabe = AUS

LIST steuert die Art des Listings, das bei einer Assemblierung gemacht wird. Dabei werden die Parameter des LIST-Befehls aus je einem Stapel gelegt; das oberste Element eines jeden Stapels ist das Aktive. Jeder LIST-Befehl gibt ein weiteres Element auf den zugehoerigen Stapel, oder nimmt das letzte weg.
LIST * heisst: nimm die zuletzt aufgestapelten Elemente weg.
LIST M z.B. heisst: lege '+' auf den "M"-Stapel und lasse die Stapel "C", "D"... wie sie sind.
Mit LIST hat man also die Moeglichkeit, neu geschriebene Programme und Macros ausfuehrlich ausgeben zu lassen, und gleichzeitig das Ausgeben von erprobten Programmen aus der Bibliothek zu verhindern.

Beispiel:
In einer Programmbibliothek koennte am Anfang einer jeder Routine stehen:

```
IF      ILIST = 0
LIST    -L, R
ENDIF
```

Angenommen das globale Symbol ILIST ist gleich null, dann wird "-" auf die Stapel "L" und "R" gelegt und somit kein Text mehr ausgegeben und keine Querverweise mehr gesammelt.

Wenn am Ende jeder Bibliotheksroutine auch die Zeilen

```
IF      ILIST = 0
LIST    *
ENDIF
```

stehen, wird der Zustand wiederhergestellt, der vor dem Erreichen der Routine geherrscht hatte.

LOC
Stellen des Adressenzaehlers

```
LABEL:      LOC     <exp16>
            <exp16> = 16-Bit Ausdruck oder Wert
```

LOC veraendert den Adressenzaehler. Der Ausdruck wird als positive 16-Bit Zahl aufgefasst und dem internen Adressenzaehler (*L) des Macro Assemblers zugewiesen.
Codes die erzeugt werden, waehrend der interne LOC-Zaehler (*L) oder *) nicht mit dem internen ORG-Zaehler (*O) uebereinstimmt, werden mit # in Spalte 7 gekennzeichnet.
Label die vor dem LOC-Befehl stehen, erhalten den Wert von *L, den er vor der Aenderung hatte.

LOC ist nuetzlich fuer die Erzeugung sich selbst ueberlagernder Programme. Auf diese Weise erzeugte Codes, koennen irgendwo im Speicher abgelegt werde (entsprechend *O), werden aber so uebersetzt als staenden sie an der Stelle die *L angibt. Natuerlich kann das Programm dort nicht laufen. Bevor es ausgefuehrt werden kann, muss es tatsaechlich dahin geschoben werden, wo es das LOC-Statement angibt. Das ist aber besonders praktisch, wenn das Programm auf ROM kopiert werden soll, oder auf diese Weise erzeugte Objekt-Programme vor der Ausfuehrung von ROM auf RAM kopiert werden.

LOC ist aber auch zur Verbesserung der Lesbarkeit von Datentabellen zur Codeumwandlung nuetzlich. Das folgende Beispiel enthaelt eine Tabelle mit BCD-Codes. Der Adressenzaehler wird auf den ATASCII-Wert des ersten Buchstaben der Liste gesetzt. So erhaelt die Spalte mit den Speicherplatznummern jetzt den ATASCII-Wert, und die Spalte mit den erzeugten Objekt-Codes den zugeordneten BCD-Wert.

Beispiele

```
                                ;Beispiel fuer den Gebrauch von LOC
                                ;zur Verbesserung der Lesbarkeit. Der
                                ;Adressenzaehler wird auf den ATASCII-
                                ;Wert gesetzt, der dem ersten BCD-Wert
                                ;der Tabelle entspricht.
                                ;
0000  = 5000                    ORG  $5000
5000  = 0041#                   LOC  'A'
0041# 61          EBCTBL:       DB   $61
0042# 62                        DB   $62
0043# 63                        DB   $63
0044# 64                        DB   $64
0045# 65                        DB   $65
                                ...
                                END
No ERRORS, 1 labels, $2403 free.
  nEBCTBL 0041                  1#8
```

```
                                ;Beispiel eines Codes, der nach $2000
                                ;assembliert wird, um spaeter ins ROM
                                ;bei Adresse $F000 uebertragen zu
                                ;werden.
      = 0500    COUNT           EQU  $0500     ;Arbeitsspeicher
0000  = 2000        ORG        $2000
2000  = F000#       LOC        $0F000
F000# A907          LDA        #07
F002# 8D0005        STA        COUNT
F005# 4C0AF0        JMP        L1
F008# EA            NOP
F009# EA            NOP
F00A# CE 0005  L1   DEC        COUNT
F00D# EA            NOP
F00E#               END
No ERRORS, 2 labels, $23F7 free.

COUNT 0500     1#4  1/8        1/12
L1    F00A     1/9  1#12
```

```
MACRO...ENDM
Definition eines Macro's
MACNAM: MACRO  parm1,...,parmn
        <Rumpf>
        ENDM       ;Ende der Definition
mit:<Rumpf> = jeder gewuenschte Text der beinhalten kann:

        %1...%9 = Parameter 1...9
        %K      = Hexadezimal-Zahl dieses Macroaufrufs
        %L      = Markenfeld dieses Macroaufrufs
        %M      = Name dieses Macros
```

Die Symbole im Variablenfeld stellen austauschbare Parameter dar. Die Namen der Symbole dienen nur der Dokumentation und duerfen im Rumpf des Macros nicht erscheinen.
Die Parameter werden innerhalb des Macros mit %x bezeichnet, x ist dabei eine Dezimal-Ziffer. %K innerhalb des Macros, wird durch die Speicherplatznummer des jeweiligen Macroaufrufs ersetzt. %L innerhalb des Macros, wird durch das Markenfeld des Macroaufrufs, %M durch den Macroaufruf ersetzt.
Das Markenfeld vor MACRO bezeichnet den Namen des Macros waehrend der Assemblierung.

Beachte: Der Pseudooperation ENDM muss ein Tabulator-Zeichen vorangehen.

Macros koennen Zeilen erzeugen, die den Aufruf eines Macros bewirken. Auf diese Weise kann ein Macro, direkt oder indirekt, sich selbst aufrufen. Deshalb muss dafuer gesorgt werden, dass ein solches "rekursives Macro" sich nicht unbegrenzt oft aufruft.
Macros koennen benutzt werden, um viele Kopien derselben Prozedur, mit verschiedenen internen Konstanten, oder in Verbindung mit VFD, um jeden denkbaren Maschinen-Code zu erzeugen.

Beispiel:
Ein Weg, die Anzahl der Bits zu bestimmen, die noetig ist um eine Zahl darzustellen ist, den Logarithmus zur Basis 2 des Wertes zu berechnen. Um das waehrend der Assemblierung zu erledigen, kann man ein rekursives Macro verwenden, um eine Art Programmschleife zu erzeugen. Zu beachten ist, dass die Bedingung fuer VAL sicherstellt, dass das Macro sich nicht unendlich oft selbst aufruft.

```
        ;               SYM = Log2[VAL] berechnen
        LOG2:           MACRO SYM,VAL
                        IF    [%2]>1
                        LOG2  %1,[%2]/2
        %1:             SET   %1+1
                        ELSE
        %1:             SET   0
                        ENDIF
                        ENDM
```

Beispiel:

```
;Macro, dass das "High Nibble" eines Speicherplatzes mit dem
;"Low Nibble" des Akkumulators verbindet, und das Ergebnis im
;Akkumulator laesst.
NPACK:    MACRO     ADDR
          EOR       %1
          AND       #$OF
          EOR       %1
          ENDM
```

Beispiel:

Manchmal ist es noetig, ein Symbol zu verwenden, dass fuer jeden Aufruf des Macros verschieden ist. Der formale Parameter %K ermoeglicht dieses. Im folgenden Macro wird ein eindeutiges Sprungziel fuer jeden Aufruf erzeugt. Zu beachten ist, dass alle Marken mit '?' beginnen, so dass die Symbol-Tabelle nicht durcheinander gebracht wird.

```
;Setze den Akkumulator = 0 wenn das Vorzeichen-Bit gesetzt ist
PARVAL:   MACRO
          BMI  ?%k
          LDA  #0
?%K:
          ENDM
```

ORG

Ursprungszaehler setzen

```
LABEL:    ORG       <exp16>
```

<exp> = irgendein absoluter, vorher definierter 16 Bit-Wert oder -Ausdruck

ORG legt die Adresse des ersten Bytes eines Objektprogramms (oder von Daten) fest. Das Label vor ORG erhaelt den Wert der Adresse von *L vor der neuen Festlegung.

Der ORG-Befehl kann in einem Programm beliebig oft verwendet werden. Er aendert nicht den laufenden USE-Block (siehe USE-Pseudooperation), aber den bezogenen Wert des Ursprungs- und Adressenzaehlers des laufenden USE-Blocks.

ORG wird fast immer am Anfang eines Programms benutzt, um die Anfangsadresse des zu erzeugenden Objekt-Programms festzulegen. Werden Ursprungs- und Adressenzaehler nicht direkt durch den ORG-Befehl (oder durch den Parameter der 0 = Kommando-Zeile) gesetzt, so erhalten beide den Wert 0.

Beispiel:

```
PROG:     ORG       $100      ;Assemblierung bei $100 beginnen
SOCK:     ORG       *0        ;*0 und *L den Wert von *0 zuweisen
```

PROC...EPROC
Festlegen des Gueltigkeitsbereiches lokaler Symbole

```
LABEL:      PROC
            Rumpf
            EPROC
```

PROC teilt dem Assembler mit, dass der folgende Programmteil bis EPROC (oder bis zum naechsten PROC) eine Prozedur ist, die lokale Symbole enthalten kann. Einem lokalen Symbol wird ein Doppelpunkt vorangestellt; es erscheint nicht in der Querverweisliste, und kann nicht ausserhalb des Gueltigkeitsbereichs verwendet werden.

Das Labelfeld erhaelt die Adresse des Wertes des *O-Zaehlers beim Abarbeiten des PROC-Befehls.

PROC sollte die erste Anweisung jeder Prozedur sein, die lokale Parameter enthaelt.

Lokale Parameter sollten zum Einsparen von Speicherplatz immer dann verwendet werden, wenn beim Schreiben groesserer Programme, Platz fuer die Symbol-Tabelle knapp wird.

Beispiel:

```
INIT:       PROC                ;Prozedur
            LDA     #0          ;A=0
            LDY     #0          ;Y-Indizierung
:LOOP:      STA     (BEGMEM),Y  ;:LOOP ist ein lokales Symbol
            INY                 ;erscheint nicht in der Querver-
                                ;weisliste
            BNE     :LOOP       ;Schreibe 256 mal Null
```

REAL6
Festlegen des Wertes einer reellen Zahl

```
LABEL:      REAL6   <fpnum>
   <fpnum> = Fliesskommazahl
```

REAL6 sorgt fuer die Umwandlung in eine reelle 6-Byte Zahl, entsprechend dem ATARI-Operating-System.

Das Label bezeichnet die Anfangsadresse der umgewandelten 6-Byte Zahl.

Beispiel:

```
   PI:      REAL6       3.14159
```

SET
Festlegen des Wertes eines Symbols

```
LABEL:      SET     <exp>
     <exp> = numerischer Ausdruck
```

Die SET-Pseudooperation gibt dem LABEL den 16-Bit Wert, den der Ausdruck des Operanden angibt. SET entspricht genau dem Befehl EQU; Labels jedoch, denen ein Wert mit SET zugewiesen wurde, koennen spaeter einen anderen Wert erhalten.

Der Ausdruck im Variablen-Feld muss zum Zeitpunkt der Berechnung einen absoluten Wert haben. Verwendete Symbole muessen vorher definiert worden sein.

Beispiel:

```
TSTVAL      SET     027h
                    ...
                    DB      TSTVAL
                    ...
TSTVAL      SET     099h
            ...
TSTVAL      SET     063h
```

SPACE
Leerzeilen im Listing

```
iglab     SPACE     <exp1>
iglab     SPACE     <exp1>,<exp2>
```

<exp1>,<exp2> = positive, numerische Ausdruecke

SPACE erzeugt Leerzeilen im Listing. Wenn SPACE nur ein Argument hat, werden entsprechend viele Leerzeichen ausgeworfen, vorausgesetzt die laufende Seite wird nicht ueberschritten. Ist das aber der Fall, erzeugt SPACE ein EJECT. Wenn SPACE zwei Argumente hat, werden beide berechnet, und <exp1> Leerzeilen nur ausgeworfen, wenn danach noch <exp2> Zeilen auf derselben Seite vorhanden sind. Ist das nicht der Fall wird wiederum ein EJECT erzeugt. Das ist vor einer kurzen Prozedur von X Zeilen nuetzlich:

```
SPACE   4,X
<Prozedur>
```

wirft nur dann 4 Leerzeilen aus, wenn die Leerzeilen und die Procedur auf dieselbe Seite passen, wenn nicht wird die Prozedur auf die naechste Seite geschrieben.

SUBTTL
Festlegen einer zweiten Ueberschrift im Listing

```
iglab     SUBTTL          <string>
```

<string> = beliebiger Text mit maximal 32 Zeichen

SUBTTL erlaubt die Festlegung einer zweitrangigen Ueberschrift. SUBTTL ohne <string> wird ignoriert. Um den Untertitel zu loeschen muss ein leeres String benutzt werden.

Beispiel:

```
TITLE       'Teil 8 - Pseudooperationen'
SUBTTL      'SUBTTL - Aufbau und Beschreibung'
SUBTTL      ' ';Untertitel loeschen
```

TITLE

Festlegen einer Ueberschriftzeile im Listing

```
iglab     TITLE <string>
```

<string> = beliebiger Text mit maximal 32 Zeichen

Text, der dem Assembler mit dem TITLE-Befehl uebergeben wird, erscheint auf jeder neuen Seite des Listings als Ueberschrift. Die Ueberschrift kann geloescht werden, indem dem Assembler ein leerer String gegeben wird. Ein TITLE-Befehl ohne Argument hingegen, loescht die Ueberschrift nicht.

Der erste Aufruf von TITLE * bewirkt keinen Seitenvorschub, alle folgenden Aufrufe bewirken stets einen Seitenvorschub, wenn irgendwelche Befehle bearbeitet worden sind. TITLE wird normalerweise an den Anfang jeder Assembler-Datei gestellt. Jede angehaengte Datei beginnt dann die Ausgabe auf einer neuen Seite, und fuehrt eine entsprechende Ueberschrift, die ihren Inhalt erkennen laesst.

Beispiel:

```
    TITLE       'XONC.asm-Uebertragungs-Unterprogramme.'
```

USE

Festlegen eines Block-Bereiches

```
     iglab      USE  name
```

USE errichtet einen neuen "USE-Block", oder nimmt einen schon festgelegten wieder auf. Ein USE-Block ist ein Teil des Quell-Programms, der wiederum in mehrere Teile untergliedert sein kann, die an verschiedenen Stellen des Programms untergebracht sind. Jeder der Teile wird umrahmt durch USE name

```
     <Block-Teil>
     USE *
```

Bei der Assemblierung erzeugt der USE-Block, trotz der raeumlichen Trennung im Quell-Programm, einen fortlaufenden Objekt-Code. Die Teile des Blocks werden dabei in der Reihenfolge aneinander gefuegt, in der der Assembler sie auffindet. In einem Programm koennen 60 verschiedene USE-Bloecke verwendet werden.

Am Ende eines jeden Teils merkt sich der Assembler den Stand des Ursprungs- und des Bit-Zaehlers (*O und *P, siehe ORG und VFD). Der Adressenzaehler (*L, siehe LOC) wird nicht gespeichert, sondern bei der Wiederaufnahme eines Blocks auf den Stand des Ursprungszaehlers gesetzt (Vorsicht also bei LOC innerhalb eines USE-Blockes). Wird den Zaehlern *O und *P am Anfang eines neuen Blockes kein Wert zugewiesen, so erhalten sie den Wert 0.

Beispiel:

```
            USE       BTABL      ;am Anfang des Programms
BTABL       ...                  ;Anfangsadresse festlegen
            USE       *          ;Abschluss des ersten Teils
            ...
NXLAB       LDX       irgendwas
            USE       BTABL      ;Wiederaufnahme des Blocks
            DW        NXLAB
            USE       *
            STX       Adr.
            ...
            USE       BTABL
            DW        0
            USE       *
            END
```

VFD
Festlegung eines Variablenfeldes

```
LABEL:    VFD      <Fexp>\<exp>,...,<Fexp>\<exp>
    <Fexp> = 1...16
    <exp>  = irgendein numerischer Ausdruck
```

VFD legt Variablenfelder fest. Jedes <Fexp> benennt die Zahl der Bits, die die Variable beinhalten darf. Jedes <exp> benennt die Variable selbst. Werte von <exp> die keinen Platz in dem von <Fexp> genannten Feld finden, werden entsprechend abgeschnitten.

Negative Werte werden als Zweierkomplement dargestellt.Z.B. wird -32768 zu 32768 und -1 zu 65535. Das Resultat wird abgeschnitten, so dass es in das von <Fexp> beschriebene Feld passt.

Ein Bit-Zaehler (*P) haelt die Zahl, der am Ende des VFD-Befehls verbleibender Bits des letzten Bytes fest. Wenn die naechste Pseudooperation wieder ein VFD-Befehl ist, beginnen die naechsten Variablenfelder mit diesen letzten unbenutzten Bits. Wenn die naechste Code erzeugende Pseudooperation kein VFD ist, werden die uebrigen Bits mit Nullen aufgefuellt.

Mit VFD ist es moeglich, beliebig komplexe Datenfelder, ohne Beruecksichtigung der Grenzen von Bytes oder Worten, zu erzeugen.

Beispiel:

```
MVINST:     VFD  2\01,3\DDD,3\SSS
```

VFD kann auf diese Weise innerhalb von Macros Codes fuer wenig gebraeuchliche Prozessoren, spezielle Peripheriebausteine usw. erzeugen.

Beispiel:

```
SPEC:       VFD  7\@43,9\I'&&'
            VFD  13\$429
```

SPEC ist ein Label das auf ein Feld von 29 Bit weist. Die ersten 7 Bits beinhalten die Oktalzahl 43, die naechsten 9 Bits beinhalten die abgeschnittene Zeichenkette && und die letzten 13 Bits die Hexadezimalzahl 429. Der *P-Zaehler zeigt in das 4. Byte nach SPEC, mit 3 unbenutzten Bits.

7 Uebersicht ueber Pseudooperationen

```
 iglab     Assert   <exp>              ;Assemblierungs-Bedingung
                                       ;pruefen
 LABEL:    DB       <exp>,<exp>       ;Bytes erzeugen
 LABEL:    DB       'ABCDE','f',$0D   ;lange Zeichenketten erzeugen
 LABEL:    DC       'ABCDE'           ;wie DB, letztes Byte um 80h
                                       ;erhoeht
 LABEL:    DS       <exp>             ;Leerzeichen erzeugen
 LABEL:    DW       <exp>,<exp>       ;Worte erzeugen
 LABEL:    DW       'Xu',1234,'y'     ;Zeichenketten mit 1 oder
                                       ;2 Zeichen erzeugen
 LABEL:    ECHO     <exp>             ;Code <exp> mal wiederholen
 iglab     EJECT                      ;Seitenvorschub
 iglab     ELSE                       ;Teil der bedingten Assem-
                                       ;blierung
 LABEL:    END      [exp]             ;Ende der Assemblierung
 iglab     ENDIF                      ;Ende von IF
 iglab     ENDM                       ;Ende eines MACRO oder ECHO
 iglab     EPROC                      ;Ende des Gueltigkeisbereiches
                                       ;lokaler Symbole
 LABEL:    EQU      <exp>             ;LABEL auf einen Wert setzen
 iglab     ERR                        ;Fehler-Flag erzwingen
 iglab     IF       <exp>             ;Anfang der bedingten Assem-
                                       ;blierung
 LABEL:    INCLUDE  <filespez>        ;anderes Quellprogramm ein-
                                       ;schliessen
 iglab     LINK     <filespez>        ;anderes Quellprogramm anhaengen
 iglab     LIST     <opt>             ;LIST-Optionen aendern
 iglab     LIST     *                 ;alte LIST-Optionen wieder-
                                       ;herstellen
 LABEL:    LOC      <exp>             ;Adressenzaehler veraendern
 NAME:     MACRO    <params>          ;Anfang einer Macro-Definition
 LABEL:    ORG      <exp>             ;Ursprungszaehler veraendern
 LABEL:    PROC                       ;Anfang des Gueltigkeitsbereichs
                                       ;lokaler Symbole
 LABEL:    REAL6    <exp>             ;Umsetzung einer reellen Zahl
                                       ;in 6 Bytes
 LABEL:    SET      <exp>             ;Label auf einen neuen Wert
                                       ;setzen
 iglab     SPACE    <exp1>,<exp2>     ;<exp1> Leerzeilen erzeugen,
                                       ;wenn <exp2> Zeilen auf der
                                       ;Seite verbleiben
 iglab     SUBTTL   'text'            ;Untertitel waehlen
 iglab     TITLE    'text'            ;Ueberschrift waehlen
 iglab     USE      <name>            ;USE-Block festlegen
 LABEL:    VFD      <exp><exp>,...    ;erzeugen von Variablen-Feldern
 LABEL:    =        <exp>             ;dasselbe wie EQU
```

```
<exp>      = notwendiger Ausdruck
[exp]      = moeglicher Ausdruck
'text'     = Zeichenketten
<filspez>  = <Geraet>:<Dateiname>.<Extension>
iglab      = Label, das ignoriert wird
```

8 Mnemoniks der Anweisungen

Die Mnemoniks der Anweisungen, die fuer den Macro Assembler vorgesehen sind, sind dieselben, die MOS Technology festgelegt hat, mit folgenden Ausnahmen:

* Anfuehrungszeichen muessen paarweise auftreten. (Manche 6502-Assembler akzeptieren es, wenn bei einer ein Zeichen langen Kette, das zweite Anfuehrungszeichen fehlt)

* Bei diesem Assembler sind die Symbole < und > Relationen ("kleiner als" und "groesser als"). Bei manchen 6502-Assembler bezeichnen sie High- und Low-Byte(siehe hierzu Kapitel 4).

Beispiele:

AMAC	MOS
CMP #'?'	CMP#'?
LDX high EXP	LDX >EXP
LDY low EXP	LDY <EXP

Bezeichnungen

dd	Ersatz fuer Vorzeichenbehaftete 8-Bit Zahl -128<=dd<=+127
mmmm	16-Bit Adress-Ausdruck
nn	8-Bit Konstante 0<= nn <= 255
zz	Platz in der Zero-Page 0<= zz <= 255

HEX	OPCODE	ADRESSE	Bemerkungen
Transport von Daten			
	Registertransport		
AA	TAX		;Transportiere A nach X
A8	TAY		;Transportiere A nach Y
BA	TSX		;Transportiere S nach X
8A	TXA		;Transportiere X nach A
9A	TXS		;Transportiere X nach S
98	TYA		;Transportiere Y nach A
	Lade Konstante ins Register		
A9	LDA	#nn	
A2	LDX	#nn	
A0	LDY	#nn	

```
                Lade Register aus Speicher
A5              LDA             zz
B5              LDA             zz,X
A1              LDA             (zz,X)
B1              LDA             (zz),Y
AD              LDA             mmmm
BD              LDA             mmmm,X
B9              LDA             mmmm,Y
A6              LDX             zz
B6              LDX             zz,Y
AE              LDX             mmmm
BE              LDX             mmmm,Y
A4              LDY             zz
B4              LDY             zz,X
AC              LDY             mmmm
BC              LDY             mmmm,X

                Lade Register in Speicher
85              STA             zz
95              STA             zz,X
81              STA             (zz,X)
91              STA             (zz),Y
8D              STA             mmmm
9D              STA             mmmm,X
99              STA             mmmm,Y
86              STX             zz
96              STX             zz,Y
8E              STX             mmmm
84              STY             zz
94              STY             zz,X
8C              STY             mmmm

                Transporte vom/zum Stapel
48              PHA                             ;Accu->Stapel
08              PHP                             ;P   ->Stapel
68              PLA                             ;Accu<-Stapel
28              PLP                             ;P   <-Stapel
```

```
                Subtrahiere Operand mit Borgen
E9              SBC             #nn
E5              SBC             zz
F5              SBC             zz,X
E1              SBC             (zz,X)
F1              SBC             (zz),Y
ED              SBC             aaaa
FD              SBC             aaaa,X
F9              SBC             aaaa,Y

                Vergleiche 8-Bit-Operand mit Accu.Setze Flags wie
                beim subtrahieren, aber aendere nicht den Accu
C9              CMP             #nn
C5              CMP             zz
D5              CMP             zz,X
C1              CMP             (zz,X)
D1              CMP             (zz),Y
CD              CMP             aaaa
DD              CMP             aaaa,X
D9              CMP             aaaa,Y

                Vergleiche 8-Bit-Operand mit dem Indexregister
E0              CPX             #nn
E4              CPX             zz
EC              CPX             aaaa
C0              CPY             #nn
C4              CPY             zz
CC              CPY             aaaa

Monadische Arithmetik
                verringere um 1
C6              DEC             zz
D6              DEC             zz,X
CE              DEC             aaaa
DE              DEC             aaaa,X
CA              DEX
88              DEY

                vergroessere um 1
E6              INC             zz
F6              INC             zz,X
EE              INC             aaaa
FE              INC             aaaa,X
E8              INX
C8              INY
```

```
                Arithmetik Kontrollflags
18              CLC                     ;Loesche das Carryflag
D8              CLD                     ;Loesche den Dezimal-Modus
B8              CLV                     ;Setze das Ueberlaufflag
38              SEC                     ;Setze das Carryflag
F8              SED                     ;Setze den Dezimal-Modus

Dyadisch logische/Boolsche Operationen
                8-Bit logisches Produkt, Konjunktion
29              AND             #nn
25              AND             zz
35              AND             zz,X
21              AND             (zz,X)
31              AND             (zz),Y
2D              AND             mmmm
3D              AND             mmmm,X
39              AND             mmmm,Y

                Logische Summe, Disjunktion, inclusive Oder
09              ORA             #nn
05              ORA             zz
15              ORA             zz,X
01              ORA             (zz,X)
11              ORA             (zz),Y
0D              ORA             mmmm
1D              ORA             mmmm,X
19              ORA             mmmm,Y

                Logische Differenz, Antivalenz, exklusive Oder
49              EOR             #nn
45              EOR             zz
55              EOR             zz,X
41              EOR             (zz,X)
51              EOR             (zz),Y
4D              EOR             mmmm
5D              EOR             mmmm,X
59              EOR             mmmm,Y

                Logische Vergleiche
                Setze Flags wie folgt:
                Z=1 wenn A und mem=0
                Z=0 wenn A und mem=1
                S=Bit 7 von mem
                V=Bit 6 von mem
                (mem=mmmm oder zz)

24              BIT             zz
2C              BIT             mmmm
```

```
Rotieren und schieben
          Arithmetische Linksverschiebung
0A        ASL           A
06        ASL           zz
16        ASL           zz,X
0E        ASL           aaaa
1E        ASL           aaaa,X

          Logische Rechtsverschiebung
4A        LSR           A
46        LSR           zz
56        LSR           zz,X
4E        LSR           aaaa
5E        LSR           aaaa,X

          Rotieren nach links
2A        ROL           A
26        ROL           zz
36        ROL           zz,X
2E        ROL           aaaa
3E        ROL           aaaa,X

          Rotieren nach rechts
6A        ROR           A
66        ROR           zz
76        ROR           zz,X
6E        ROR           aaaa
7E        ROR           aaaa,X

Spruenge
90        BCC                         ;Wenn C=0
B0        BCS                         ;Wenn C=1
F0        BEQ                         ;Wenn Z=1
30        BMY                         ;Wenn N=1
D0        BNE                         ;Wenn Z=0
10        BPL                         ;Wenn N=0
50        BVC                         ;Wenn V=0
70        BVS                         ;Wenn V=1
4C        JMP           mmmm
6C        JMP           (mmmm)

Unterprogramm-Aufruf
00        BRK                         ;Software interupt
20        JSR           mmmm          ;Sprung zum Unterprogramm

Ruecksprung aus dem Unterprogramm
40        RTI                         ;Ruecksprung aus dem Interrupt
60        RTS                         ;Ruecksprung aus dem Unter-
                                      ;programm

Verschiedenes
58        CLI                         ;Loesche Interupt Maske (EI)
EA        NOP
78        SEI                         ;Setze die Interupt Maske (DI)
```

9 Benutzung des ATARI Macro Assemblers mit den ATARI Assembler Editor Quellprogrammen

Wenn man ein Quellprogramm hat, dass mit der ATARI Assembler Editor ROM entwickelt wurde, und man will es vom Macro Assembler assemblieren lassen, muss man folgende Unterschiede beruecksichtigen:

* Der Macro Assembler nimmt keine Zeilennummern an
* Das = fuer EQU muss von wenigstens je einem Leerzeichen umrahmt werden.
* Kommentaren muss ein Semicolon vorangestellt werden.
* Folgende Pseudo-Operationen entsprechen einander:

Assembler Editor	Macro Assembler
.BYTE	DB
.END	END
.PAGE	TITLE
.WORD	DW

* Statt des Kommandos *=... muss ORG zum Setzen des Ursprungszaehlers verwendet werden.

10 Fehlermeldungen

Fehler werden durch einzelne Buchstaben in der ersten Spalte des Listings signalisiert. Fehlerhafte Zeilen werden immer auf den Bildschirm geschrieben, egal welches Ausgabegeraet gewaehlt worden ist.

A = Adresse fehlerhaft. Die gegebene Anweisung vertraegt sich nicht mit der gegebenen Adressierungsart.

D = Doppelt vorhandenes Label. Das zuletzt benutzte Label wird genommen.

E = Falscher Ausdruck. Ausdruck der Quellzeile kann nicht erkannt werden.

F = Falsche Anordnung von IF...ELSE...ENDIF-Befehlen. Taucht diese Meldung vor der END-Zeile auf, heisst das, dass ein IF nicht gefunden wurde.

I = Anweisungsfeld wurde nicht gefunden. 3 NOP wurden erzeugt.

L = Labelfeld wurde nicht gefunden. 3 NOP wurden erzeugt.

M = Macro falsch definiert.

N = Zahlenfehler; Ziffer ueberschreitet die Basis, Wert ueberschreitet 16-Bit usw.

O = Ueberlauf des Stapels bei der Berechnung von Ausdruecken. Ausdruecke sollten vereinfacht werden. Zu viele LINK-Dateien, zu viele PROC,zu viele USE-Bloecke.

P = vom Programmierer erzwungener Fehler. Siehe ASSERT und ERR Pseudo-Operation.

R = Ausdruck im Variablenfeld kann nicht berechnet werden

S = Aufbaufehler im Befehl. Zu viele oder zu wenige Adressen-Unterfelder.

U = Verweis zu einem nicht definierten Symbol.

V = Ausdruck Ueberlauf: Ergebnis wurde abgeschnitten.

ATARI®

A Warner Communications Company

ATARI-Elektronik Vertriebsgesellschaft mbH
Postfach 60 01 69 · Bebelallee 10 · 2000 Hamburg 60