Commodore
Sachbuch

Mit Schlagwortregister der ROM-Routinen

Christian Quirin Spitzner

C16, C116 und Plus/4 ROM-Listing

Ausführlich dokumentiertes ROM-Listing des BASIC-Interpreters, Betriebssystems und Monitors. Mit Beschreibung der wichtigsten Kernel-Routinen und Zero-Page-Adressen.

Commodore
Sachbuch

C 16, C 116 und Plus/4 ROM-Listing

Ausführlich dokumentiertes ROM-Listing des BASIC-Interpreters, Betriebssystems und Monitors.
Mit Beschreibung der wichtigsten Kernel-Routinen und Zero-Page-Adressen.

Christian Quirin Spitzner

Markt & Technik Verlag AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Spitzner, Christian Quirin:
C 16, C 116, Plus 4 ROM-Listing : ausführl. dokumentiertes ROM-Listing d. BASIC-Interpreters, Betriebssystems u. Monitors ; mit Beschreibung d. wichtigsten Kernel-Routinen u. Zero-Page-Adressen / Christian Quirin Spitzner. – Haar bei München : Markt-und-Technik-Verlag, 1987.
(Commodore-Sachbuch)
ISBN 3-89090-425-4

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
90 89 88 87

ISBN 3-89090-425-4

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Kösel Kempten
Printed in Germany

Inhaltsverzeichnis

Vorwort

Dieses Buch findet seine Leser, im Gegensatz zu anderen Sachbüchern dieser Art, unter den Besitzern von drei verschiedenen Computern. Das soll nicht heißen, daß dieses Buch allgemeingültigen Charakter besitzt. Nein, dieses Buch enthält ein ausführlich dokumentiertes ROM-Listing (ROM = Read Only Memory) für die Commodore-Computer C 16, C 116 und Plus/4.

Alle drei Geräte besitzen dieselben ROMs. Es sei denn, Sie besitzen ein älteres Modell des Computers. Diese wurden mit kleinen Fehlern im Betriebssystem ausgeliefert. Dokumentiert werden beide Betriebssysteme, wobei die ältere Version grau unterlegt ist.

Die Computer C 16, C 116 und Plus/4 unterscheiden sich im wesentlichen nur in drei Dingen:

Zum einen steckt jeder der Computer in einem anderen Gehäuse mit anderer Tastatur. Der C 16 und C 116 verfügen im Gegensatz zum Plus/4 in der Grundversion nur über 16 Kbyte Speicherplatz. Bis auf diesen Unterschied sind C 16 und C 116 völlig identisch. Der Plus/4 besitzt eine Speicherkapazität von 64 Kbyte und verfügt zusätzlich über drei fest eingebaute Anwenderprogramme.

Das dokumentierte ROM-Listing umfaßt den BASIC-Interpreter, den Monitor (TED-MON) und das Betriebssystem. Die Anwender-Programme im ROM des Plus/4 werden nicht beschrieben, da sie für den Programmierer so gut wie nicht verwertbar sind!

Besonders interessant ist dieses Buch für Programmierer, die einmal etwas tiefer ins Innere ihres Computers blicken wollen. Maschinensprache-Programmierer finden in diesem Buch viele nützliche Routinen, die sie in ihren eigenen Programmen verwenden können. Im Anhang sind die Adressen aller wichtigen Routinen des Betriebssystems, sowohl alphabetisch als auch nach Einsprungadressen sortiert, zusammengetragen.

Für experimentierfreudige Programmierer bieten das Betriebssystem sowie der BASIC-Interpreter ein weites Betätigungsfeld. So könnte das gemeinsame BASIC 3.5 der drei Computer ohne weiteres noch Unterstützung durch eigene BASIC-Befehle vertragen. Entsprechende Routinen, die diesem Wunsch nach eigenen BASIC-Befehlen nachkommen, sind im Interpreter bereits vorhanden.

Ein sehr wichtiger Bereich des Computers ist die Zeropage bzw. die erweiterte Zeropage. Hier befinden sich alle wichtigen Systemadressen. Der Bereich liegt im RAM-Speicher der Computer und wird jedesmal nach dem Einschalten oder einem Reset neu initialisiert. Auch dieser Bereich wird in einem Kapitel ausführlich aufgelistet und beschrieben. Dadurch bieten sich dem Programmierer ungeahnte Möglichkeiten.

Um das Verständnis der ROM-Routinen zu erleichtern, werden in den ersten Kapiteln kurze Einführungen in wichtige Grundlagen der Computer gegeben. Die wichtigsten Kernel-Routinen werden kurz erklärt. Diese Einführung umfaßt außerdem einen groben Umriß der Speicherverwaltung des C 16, C 116 und Plus/4.

Es bleibt zu hoffen, daß Sie mit Hilfe dieses Buches in der Lage sind, auch schwierige Probleme in Assembler zu bewältigen.

Danken möchte ich allen, die mir beim Erstellen dieses Buches behilflich waren.

Poing, Oktober 1986

Christian Quirin Spitzner

1 Wichtige Kernel-Routinen

Viele Maschinenprogramme lassen sich mit Hilfe der im Betriebssystem vorhandenen ROM-Routinen erleichtern und verkürzen. Durch das Stichwortverzeichnis im Anhang des Buches können diese Betriebssystem-Routinen schnell und leicht ausfindig gemacht werden. Viele dieser Routinen können ohne weitere Vorarbeiten mit dem JMP-, JSR- oder SYS-Befehl aufgerufen werden. In der Regel stürzt jedoch der Computer bei unsachgemäßem Umgang ab. Das eventuell im Speicher vorhandene Programm ist dann unter Umständen zerstört. Also Vorsicht beim Umgang mit den Kernel-Routinen!

Die wichtigsten 39 Betriebssystem-Routinen des C 16, C 116 und Plus/4 können über die Kernel-Sprungtabelle am Ende des Betriebssystems erreicht werden. Hier finden sich im wesentlichen Standard-Routinen für die Ein-/Ausgabe und Speicherverwaltung des Betriebssystems. Da fast jeder Commodore-Computer ein anderes Betriebssystem besitzt, ist es in der Regel mühsam und zeitaufwendig, Maschinenprogramme von einem auf den anderen Computer umzuschreiben. Vor allem, wenn das Programm die spezifischen Routinen des Computers direkt anspringt. Verwendet das Programm ausschließlich Routinen, die über die Kernel-Sprungtabelle erreicht werden, so stehen die Chancen gut, Programme an andere Commodore-Computer anzupassen, da die Kernel-Sprungtabellen aller Commodore-Computer identisch sind. Bei allen Commodore-Computern liegt die Kernel-Sprungliste innerhalb der letzten Page des Betriebssystems (65409-65526 bzw.$FF81-$FFF6). Die Computer C 16, C 116 und Plus/4 besitzen zusätzliche Sprungtabellen für Optionen, die bei den älteren Commodore-Modellen nicht vorhanden waren. Auch von diesen Routinen werden einige behandelt.

Beim Erstellen von Maschinenprogrammen empfiehlt es sich, um Zeit und Platz zu sparen, die vorhandenen Routinen des Betriebssystems zu nutzen. Diese Routinen können auch direkt angesprungen werden. Vorausgesetzt, man weiß, wo sich diese befinden. Dazu ist ein ausführlich dokumentiertes ROM-Listing erforderlich, um zu ersehen, welche Vorarbeiten geleistet werden müssen, bevor eine Routine aufgerufen werden kann. Die meisten ROM-Routinen verlangen folgende Arbeitsschritte:

1) Übergabe der Parameter oder Adressen
2) Aufruf der Routine
3) Gegebenenfalls Fehlerbehandlung

Einsprung-adresse:	Sprung nach Adresse:	Label	Funktion
$ff81	JMP $d84e	CINT	Initialisiert TED-Chip und Bildschirmeditor
$ff84	JMP $f30b	IOINIT	Initialisiert I/O-Bausteine und Routinen
$ff87	JMP $f352	RAMTAS	Initialisiert RAM-Bereich und Funktionstasten
$ff8a	JMP $f2ce	RESTOR	Betriebssystem-Vektoren einrichten
$ff8d	JMP $f2d3	VECTOR	Vektor-Adressen kopieren und verändern
$ff90	JMP $f41a	SETMSG	Ausgabe von System- und Fehlermeldungen
$ff93	JMP $ee4d	SECOND	Übergabe der Sekundäradresse nach LISTEN
$ff96	JMP $ee1a	TKSA	Übergabe der Sekundäradresse nach TALK
$ff99	JMP $f427	MEMTOP	Zeiger für RAM-Ende setzen oder holen
$ff9c	JMP $f436	MEMBOT	Zeiger für RAM-Anfang setzen oder holen
$ff9f	JMP $db11	SCNKEY	Tastatur abfragen
$ffa2	JMP $f423	SETTMO	TIMEOUT-Flag setzen oder löschen (IEEE-Bus)
$ffa5	JMP $ec8b	ACPTR	Liest Daten vom seriellen Bus
$ffa8	JMP $ecdf	CIOUT	Gibt Bytes über den seriellen Bus aus
$ffab	JMP $ef3b	UNTLK	Beendet Datenübertragung (UNTALK)
$ffae	JMP $ef23	UNLSN	Beendet Datenempfang (UNLISTEN)
$ffb1	JMP $ee2c	LISTEN	Gerät für Datenempfang vorbereiten
$ffb4	JMP $edfa	TALK	Gerät für Datentransport vorbereiten
$ffb7	JMP $f41c	READST	Ein-/Ausgabestatus lesen
$ffba	JMP $f413	SETLFS	Logische Datei einrichten
$ffbd	JMP $f40c	SETNAM	Übergibt Dateiname für OPEN, SAVE oder LOAD
$ffc0	JMP ($0318)	OPEN	öffnet eine logische Datei
$ffc3	JMP ($031a)	CLOSE	Schließt eine logische Datei
$ffc6	JMP ($031c)	CHKIN	Setzt Eingabekanal
$ffc9	JMP ($031e)	CKOUT	Setzt Ausgabekanal
$ffcc	JMP ($0320)	CLRCH	Schließt alle Ein-/Ausgabekanäle
$ffcf	JMP ($0322)	BASIN	Eingabe eines Zeichens
$ffd2	JMP ($0324)	BSOUT	Ausgabe eines Zeichens
$ffd5	JMP $f043	LOAD	File laden (LOAD) oder überprüfen (VERIFY)
$ffd8	JMP $f194	SAVE	File speichern (SAVE)
$ffdb	JMP $cf2d	SETTIM	Setzt Timer (TI)
$ffde	JMP $cf26	RDTIM	Liest Timer (TI)
$ffe1	JMP ($0326)	STOP	Prüft STOP-Taste
$ffe4	JMP ($0328)	GETIN	Holt Zeichen
$ffe7	JMP ($032a)	CLALL	Schließt alle offenen Dateien
$ffea	JMP $cef0	UDTIM	Aktualisiert Systemzeit
$ffed	JMP $d834	SCREEN	Holt Bildschirmformat
$fff0	JMP $d839	PLOT	Cursorposition setzen/holen
$fff3	JMP $fc19	IOBASE	Adresse der Ein-/Ausgaberegister ermitteln

Tabelle 1.1: Alle 39 Kernel-Routinen mit Label und Bedeutung

CINT ($FF81 / 65409):
Diese Routine initialisiert den TED-Chip und den Bildschirmeditor. Sie kann ohne jegliche Vorarbeiten mit JSR $FF81 oder SYS 65409 aufgerufen werden.

IOINIT ($FF84 / 65412):
Mit IOINIT werden alle Ein-/Ausgabe-Bausteine und Routinen initialisiert. Da auch hier keine Parameter verlangt werden, kann die Routine mit JSR $FF84 oder SYS 65412 aufgerufen werden.

RAMTAS ($FF87 / 65415):
Die RAMTAS-Routine initialisiert das RAM des C 16, C 116 und Plus/4 und setzt entsprechend die Zeiger für den zulässigen RAM-Bereich des Computers. Zusätzlich werden die Funktionstasten programmiert, die RAM-Laderoutine sowie die Farbcodes ins RAM kopiert. Diese Routine kann nicht ohne weiteres aufgerufen werden, da die gesamte Zeropage gelöscht wird und somit der BASIC-Interpreter nicht mehr lauffähig ist. Diese Routine wird vom Betriebssystem bei einem Reset aufgerufen.

RESTOR ($FF8a / 65418):
Diese Routine korrigiert die Adressen für die wichtigsten Betriebssystem-Vektoren in den Bereich $0312-$0331. Auch diese Routine läßt sich ohne weitere Vorbereitungen aufrufen (JSR $FF8a oder SYS 65418).

VECTOR ($FF8d / 65421):
Mit dieser Routine lassen sich die wichtigsten Betriebssystem-Vektoren (Bereich $0312-$0331) beliebig ändern. Ist beim Aufruf der Routine Carry gleich 1, so werden zunächst alle Vektor-Adressen in den Speicherbereich kopiert, dessen Adresse in (X-Reg., Y-Reg.) gespeichert ist. Ist das Carry-Bit gelöscht, wird der Vorgang umgekehrt. Also Vorsicht mit diesem Befehl! Zeigen (X-Reg., Y-Reg.) nicht auf eine Tabelle mit sinnvollen Vektor-Adressen, so führt dies unweigerlich zum Absturz des Computers. Unser kleines Beispielprogramm (Listing 1.1) vertauscht die OPEN- und CLOSE-Vektoren indem die Vektor-Adressen nach $2050 kopiert, dort geändert und anschließend wieder zurückkopiert werden. Nach dem Starten der Routine mit SYS 8192 können Sie Ihren Drucker folgendermaßen aktivieren:

```
CLOSE 1,4 : CMD 1 : LIST        etc.
```

Es empfiehlt sich, nach diesem Beispielprogramm den Computer auszuschalten oder einen Reset durchzuführen, da durch Veränderung der beiden Vektoren der BASIC-Interpreter nicht mehr einwandfrei arbeitet!

```
. 2000  a2 50     ldx #$50    Tabelle nach Adresse
. 2002  a0 20     ldy #$20    ... $2050 kopieren
. 2004  38        sec         Carry = 1
. 2005  20 8d ff  jsr $ff8d   Vektoren kopieren
. 2008  a9 5d     lda #$5d    OPEN-
. 200a  8d 56 20  sta $2056   ... und
. 200d  a9 ee     lda #$ee    ... CLOSE-
. 200f  8d 57 20  sta $2057   ... Vektoren
```

```
. 2012  a9 53    lda #$53     ... vertauschen
. 2014  8d 58 20 sta $2058
. 2017  a9 ef    lda #$ef
. 2019  8d 59 20 sta $2059
. 201c  a2 50    ldx #$50
. 201e  a0 20    ldy #$20
. 2020  18       clc          Carry = 0
. 2021  20 8d ff jsr $ff8d    (VECTOR) Vektor einrichten
. 2024  60       rts          ENDE
```

Listing 1.1: Beispielprogramm (VECTOR)

SETMSG ($FF90 / 65424):
Die SETMSG-Routine erlaubt es, System- und Fehlermeldungen in gewissen Grenzen zu variieren oder gar zu unterdrücken. Unter Fehlermeldungen versteht man Meldungen wie I/O ERROR #. Systemmeldungen sind Kommentare wie LOADING etc. Vor Aufruf der Routine muß der Akku mit dem entsprechenden Code belegt werden. Der Aufruf erfolgt also folgendermaßen:

```
LDA #$..     (HEX-Code aus Tabelle 1.2)
JSR $FF90    (SETMSG)
```

Folgende Kombinationen lassen sich durch Aufruf der Routine mit entsprechender Belegung des Akkumulators einstellen:

Akku	Fehlermeldung	Systemmeldung
$00	Ein	Ein
$40	Aus	Ein
$80	Ein	Aus
$C0	Aus	Aus

Tabelle 1.2: Auswahl der Meldungen

SECOND ($FF93 / 65427):
Über diese Routine wird eine Sekundäradresse zu einem Ein-/Ausgabegerät übertragen. Voraussetzung ist der Aufruf der LISTEN-Routine. Das Ein-/Ausgabegerät ist anschließend empfangsbereit. Die Routine kann nicht zur Übertragung einer Sekundäradresse nach dem Aufruf der TALK-Routine benutzt werden! Als Vorarbeit muß zunächst die Gerätenummer mit 96 ($60 = %01100000) ODER-verknüpft und die LISTEN-Routine ($FFB1) aufgerufen werden. Erst dann kann die Sekundäradresse in den Akku geladen und die SECOND-Routine aufgerufen werden (Listing 1.2).

```
. 2000  a9 08    lda #$08     Geräteadresse 8
. 2002  09 60    ora #$60     ODER-Verknüpfung mit 96
. 2004  20 b1 ff jsr $ffb1    LISTEN-Routine
. 2007  a9 0f    lda #$0f     Sekundäradresse 15
```

```
. 2009  20 93 ff jsr $ff93     SECOND-Routine
. 200c  60       rts           ENDE
```

Listing 1.2: Beispielprogramm für SECOND

TKSA ($FF96 / 65430):
Diese Routine hat ebenfalls die Aufgabe, eine Sekundäradresse an ein Ein-/Ausgabegerät zu übertragen. Die Routine ist aber im Gegensatz zur SECOND-Routine dafür ausgelegt, eine Sekundäradresse über den seriellen Bus zu einem TALK-Gerät zu übertragen. Vor Aufruf der Routine ist unbedingt die TALK-Routine zu benutzen und der Akku mit dem Sekundäradreßbefehl zu belegen. Auch hier muß der Befehl mit 96 ODER-verknüpft werden. Die TKSA-Routine ist nach LISTEN unwirksam! Im folgenden Beispielprogramm wird an das Gerät 4 über den seriellen Bus die Sekundäradresse 7 gesandt (Listing 1.3).

```
. 2000  a9 04    lda #$04      Gerätenummer 4
. 2002  20 b4 ff jsr $ffb4     TALK-Routine
. 2005  a9 07    lda #$07      Sekundäradresse 7
. 2007  09 60    ora #$60      ODER-Verknüpfung mit 96
. 2009  20 96 ff jsr $ff96     TKSA-Routine
. 200c  60       rts           ENDE
```

Listing 1.3: Beispielprogramm für TKSA

MEMTOP ($FF99 / 65433):
Diese Routine hat zweierlei Funktionsmöglichkeiten:

- Ist beim Aufruf das Carry-Bit gesetzt, dann wird das RAM-Ende ausgelesen und an das X- und Y-Register übergeben.
- Ist Carry hingegen 0, so wird das RAM-Ende auf den Wert des X- und Y-Registers gesetzt und somit der RAM-Speicherbereich verkleinert oder vergrößert.

Das Beispielprogramm (Listing 1.4) liest das RAM-Ende und gibt dieses als Integerwert am Bildschirm aus. Wird das Programm zum Beispiel im Plus/4 mit SYS 8192 gestartet, erscheint die Zahl »64768« am Bildschirm. Das heißt, der RAM-Bereich des Plus/4 reicht bis $FD00.

Beispielprogramm 1.5 setzt zuerst das RAM-Ende auf $2000 und gibt dann ebenfalls wie Listing 1.4 das RAM-Ende als Integerzahl (8192) aus. Auch dieses Programm wird mit SYS 8192 gestartet.

```
. 2000  38       sec           Carry = 1 (für lesen)
. 2001  20 99 ff jsr $ff99     MEMTOP (RAM-Ende lesen)
. 2004  98       tya           Ergebnis nach Akku, X-Register
. 2005  20 5f a4 jsr $a45f     Integerwert ausgeben
. 2008  60       rts           ENDE
```

Listing 1.4: RAM-Ende auslesen und ausgeben

```
. 2000  a2 00     ldx #$00      RAM-Ende
. 2002  a0 20     ldy #$20      ... auf $2000 setzen
. 2004  18        clc           Carry = 0 (für setzen)
. 2005  20 99 ff  jsr $ff99     (MEMTOP) RAM-Ende := $2000
. 2008  38        sec           ...wie Listing 1.4...
. 2009  20 99 ff  jsr $ff99     MEMTOP (RAM-Ende setzen)
. 200c  98        tya           Ergebnis nach Akku, X-Register
. 200d  20 5f a4  jsr $a45f     Integerwert ausgeben
. 2010  60        rts           ENDE
```

Listing 1.5: RAM-Ende auf $2000 setzen und auslesen

MEMBOT ($FF9C / 65436):
Diese Routine ist äquivalent zur MEMTOP-Routine. Hier wird lediglich der RAM-Anfang gelesen oder neu festgelegt. Listing 1.4 und 1.5 lassen sich also auch hier anwenden. Es müssen nur die Sprungadressen von $FF99 auf $FF9C umgeändert werden.

SCNKEY ($FF9F / 65439):
Diese Routine stellt fest, welche Taste gerade gedrückt ist, und schreibt diese als ASCII-Wert in den Tastaturpuffer. Bevor diese Routine angesprungen wird, sollten mit IOINIT die Ein-/Ausgabe-Bausteine initialisiert werden.

Unser kleines Beispielprogramm (Listing 1.6) holt zunächst die aktuelle Taste in den Tastaturpuffer. Mit GETIN wird die Taste als ASCII-Code aus dem Tastaturpuffer in den Akku übertragen. Wurde keine Taste gedrückt, so wird erneut die Tastatur abgefragt. Ansonsten wird das entsprechende Zeichen mit BSOUT am Bildschirm ausgegeben. Das Programm wird mit SYS 8192 gestartet und mit der Stop-Taste unterbrochen.

```
. 2000  20 9c ff  jsr $ff9c   (SCNKEY) holt aktuelle Taste
. 2003  20 e4 ff  jsr $ffe4   (GETIN) ASCII-Code nach Akku
. 2006  f0 f8     beq $2000   Keine Taste?, dann nochmal
. 2008  20 d2 ff  jsr $ffd2   (BSOUT) Zeichen ausgeben
. 200b  c9 03     cmp #$03    STOP-Taste gedrückt ?
. 200d  d0 f1     bne $2000   Nein, dann weiter
. 200f  60        rts         ENDE
```

Listing 1.6: Beispielprogramm zur Tastaturabfrage mit der SCNKEY-Routine

SETTMO ($FFA2 / 65442):
SETTMO setzt das TIMEOUT-Flag für den IEEE-Bus. Diese Routine funktioniert natürlich nur bei angeschlossenen Peripheriegeräten mit IEEE-Bus und einem entsprechenden Interface für den C 16, C 116 und Plus/4. Diese Routine erinnert sozusagen an die alten CBM-Computer (CBM 20xx, 30xx, 40xx oder 80xx). Für diese Geräte existierten ausschließlich Peripheriegeräte mit IEEE-Bus. Bei gesetztem TIMEOUT-Flag wartet der Computer 64 Millisekunden auf eine Meldung eines Gerätes am IEEE-Port. Antwortet kein Peripheriegerät, so erkennt der Computer einen Fehler und verläßt die Handshake-Sequenz. Das TIMEOUT-Flag wird gesetzt, wenn vor

Aufruf der Routine Bit 7 des Akkumulators gelöscht ist. Ist Bit 7 gesetzt, so sind keine TIMEOUTs wirksam.

ACPTR ($FFA5 / 65445):
Mit der ACPTR-Routine lassen sich Daten vom seriellen Bus, wie zum Beispiel Diskettenstation, lesen. Es kann direkt ein Datenbyte vom seriellen Bus gelesen werden. Zur Vorbereitung muß die TALK-Routine und eventuell die TKSA-Routine aufgerufen werden. Mit TALK wird der serielle Bus für die Datenübertragung aktiviert. Die TKSA-Routine wurde bereits oben erklärt.

Nach Aufruf der ACPTR-Routine steht das aktuelle, gerade am seriellen Bus anliegende Byte im Akku und kann dort bearbeitet werden.

CIOUT ($FFA8 / 65448):
Mit dieser Routine kann ein Byte direkt an den seriellen Bus übergeben werden. Durch Aufruf dieser Routine wird der Akku auf den seriellen Bus mit »Handshake« übertragen. Als Vorbereitung muß die LISTEN-Routine aufgerufen werden. Bei Angabe einer Sekundäradresse muß zusätzlich die SECOND-Routine (oben besprochen) angesprungen werden. Ist das angesprochene serielle Gerät nicht für den Datenempfang vorbereitet, meldet das Statuswort ein TIMEOUT. Wie diese Fehlermeldung behandelt werden kann, wird später beschrieben. Zur Beendigung der Übertragung muß die UNLSN-Routine angesprungen werden.

UNTLK ($FFAB / 65451):
Mittels dieser Routine wird ein UNTALK-Befehl über den seriellen Bus übertragen. Alle Geräte, die zuvor einen TALK-Befehl erhalten haben, beenden daraufhin die Datenübertragung. Zum Aufruf der Routine sind keine Vorarbeiten nötig.

UNLSN ($FFAE / 65454):
Diese Routine sendet einen UNLISTEN-Befehl an den seriellen Bus. Dadurch erhalten alle Geräte am seriellen Bus den Befehl, den Datenempfang einzustellen. Hierbei werden allerdings nur die Geräte beeinflußt, die zuvor auch den LISTEN-Befehl erhalten haben. Nach Aufruf der Routine werden die betreffenden Geräte vom seriellen Bus getrennt und warten auf neue Anweisungen. Auch diese Routine kann ohne Vorarbeiten aufgerufen werden.

LISTEN ($FFB1 / 65457):
Die LISTEN-Routine sendet den LISTEN-Befehl an ein Gerät am seriellen Bus. Zur Vorbereitung wird die gewünschte Gerätenummer in den Akku geladen und die Routine aufgerufen. Über die LISTEN-Anweisung wird die Zahl im Akku, Bit für Bit ODER-verknüpft und in eine LISTEN-Adresse umgewandelt. Dieses Byte wird dann als Befehl über den seriellen Bus übertragen (Beispiel siehe Listing 1.2).

TALK ($FFB4 / 65460):
TALK sendet über den seriellen Bus den TALK-Befehl zu einem Peripheriegerät. Das Gerät wird dadurch vorbereitet, Daten an den Computer zu senden. Vor Aufruf der Routine ist zunächst die gewünschte Gerätenummer in den Akku zu laden. Dieser

Wert wird anschließend bitweise ODER-verknüpft und als TALK-Adresse über den seriellen Bus gesendet (siehe Listing 1.3).

READST ($FFB7 / 65463):
Die READST-Routine liest den Status der Ein-/Ausgabegeräte und übergibt diesen an den Akkumulator. Diese Routine sollte nach jedem Transfer mit einem Ein-/Ausgabegerät aufgerufen werden, um zu überprüfen, ob Fehler während der Ein-/Ausgabe aufgetreten sind. Nach Aufruf der Routine muß der Akku ausgewertet werden.

- Ist Bit 0 gesetzt, wird ein TIMEOUT beim Schreiben über den seriellen Bus gemeldet.
- Gesetztes Bit 1 zeigt einen TIMEOUT beim Lesen an.
- Bit 2 signalisiert bei Kassettenoperationen einen »kurzen Satz«.
- Ist Bit 3 gesetzt, handelt es sich entsprechend um einen »langen Satz«.
- Bei gesetztem Bit 4 wird ein »Nicht korrigierbarer Lesefehler« von Kassette gemeldet.
- Bit 5 zeigt bei Kassettenoperationen einen Prüfsummenfehler an.
- Bit 6 zeigt beim Lesen von Diskette das Dateiende an. Am seriellen Bus wird eine EOI-Leitung signalisiert.
- Ist das angesprochene serielle Gerät nicht vorhanden, wird Bit 7 gesetzt (DEVICE NOT PRESENT ERROR). Bei Kassettenoperationen zeigt Bit 7 das Bandende an.

Das Beispielprogramm (Listing 1.7) liest den Ein-/Ausgabestatus und gibt diesen bitweise am Bildschirm aus. Versuchen Sie nun, das Directory Ihrer Diskette zu lesen und lassen Sie dabei den Floppyschacht offen! Nun müßte die LED des Diskettenlaufwerkes einen Fehler anzeigen. Wenn nun mit SYS 8192 das Beispielprogramm gestartet wird, müßte Bit 2 (für TIMEOUT beim Lesen) gesetzt sein. Versucht man auf die Floppy zuzugreifen, wenn diese ausgeschaltet ist, ist Bit 7 gesetzt (Floppy nicht vorhanden). Diese Versuche lassen sich natürlich auch mit der Datasette durchführen.

```
. 2000  20 b7 ff jsr $ffb7  (READST) Status lesen
. 2003  8d 20 20 sta $2020  Status zwischenspeichern
. 2006  a9 08    lda #$08   Zähler für 8 Bit
. 2008  8d 21 20 sta $2021  ... (Bit 0-7) einrichten
. 200b  a9 00    lda #$00   High-Byte gleich 0
. 200d  a2 01    ldx #$01   Low-Byte gleich 1
. 200f  0e 20 20 asl $2020  Status Bit für Bit holen
. 2012  b0 02    bcs $2016  Bit gesetzt ?, dann 1 ausgeben
. 2014  a2 00    ldx #$00   Low-Byte gleich 0
. 2016  20 5f a4 jsr $a45f  Integerzahl ausgeben
. 2019  ce 21 20 dec $2021  Zähler um 1 verringern
```

```
. 201c  d0 ed    bne $200b  Zähler noch nicht 0 ?, dann weiter
. 201e  60       rts        ENDE
```

Listing 1.7: Ein-/Ausgabestatus bitweise darstellen

SETLFS ($FFBA / 65466):
Diese Kernel-Routine richtet eine logische Datei ein. Sie übergibt Dateinummer, Geräteadresse und Sekundäradresse für andere Kernel-Routinen. Zur Vorbereitung muß die Geräteadresse (0-31) in das X-Register geladen werden. Der Akku muß die logische Dateinummer enthalten. Die Sekundäradresse wird ins Y-Register gelegt. Soll keine Sekundäradresse übertragen werden, muß dieses Register auf $FF (255) gesetzt werden. Das Beispielprogramm setzt logische Dateinummer 1, Gerätenummer 8 und Sekundäradresse 15.

```
. 2000  a9 01    lda #$01   Logische Dateinummer 1
. 2002  a2 08    ldx #$08   Geräteadresse 8
. 2004  a0 0f    ldy #$0f   Sekundäradresse 15
. 2006  4c ba ff jmp $ffba  (SETLFS) Datei einrichten
```

Listing 1.8: Beispielprogramm für SETLFS

SETNAM ($FFBD / 65469):
SETNAM legt die Dateinamen für die Kernel-Routinen OPEN, SAVE und LOAD fest. Dabei wird der Akkumulator mit der Länge des Namens belegt. X- und Y-Register enthalten die Adresse des Namens, der als String irgendwo im Speicher liegen kann.

Das X-Register enthält dabei das Low-Byte der Adresse. Das Y-Register entsprechend das High-Byte.

Das Beispielprogramm (Listing 1.9) übergibt den 5stelligen Namen "c 116":

```
. 2000  a9 05    lda #$05   Länge des Namens gleich 5
. 2002  a0 0a    ldy #$20   Adresse des
. 2004  a2 20    ldx #$0a   ... Dateinamens ($200a)
. 2006  20 bd ff jsr $ffbd  (SETNAM) Name übergeben
. 2009  60       rts        ENDE
. 200a 43 20 31 31 36       "c 116"
```

Listing 1.9: Name "c 116" für OPEN, SAVE oder LOAD übergeben

OPEN ($FFC0 / 65472):
Die OPEN-Routine öffnet eine logische Datei. Voraussetzung ist hierbei, daß die Kernel-Routinen SETNAM und SETLFS vorher aufgerufen wurden. Für die Routine selbst sind keine Parameter erforderlich. Mit dem nun folgenden Beispielprogramm (Listing 1.10) wird der BASIC-Befehl

```
OPEN 1,8,15,"n:c 116"
```

simuliert. Vorsicht: Das Programm formatiert Ihre Diskette (ohne ID). Legen Sie also eine Diskette mit nicht (mehr) benötigtem Inhalt ein, bevor Sie das Programm starten!

```
. 2000  a9 07     lda #$07   Länge des Namens gleich 7
. 2002  a0 20     ldy #$20   Adresse des
. 2004  a2 16     ldx #$16   ... Dateinamens ($2016)
. 2006  20 bd ff  jsr $ffbd  (SETNAM) Name übergeben
. 2009  a9 01     lda #$01   Logische Dateinummer gleich 1
. 200b  a2 08     ldx #$08   Geräteadresse gleich 8
. 200d  a0 0f     ldy #$0f   Sekundäradresse gleich 15
. 200f  20 ba ff  jsr $ffba  (SETLFS) Parameter übergeben
. 2012  20 c0 ff  jsr $ffc0  (OPEN)
. 2015  60        rts        ENDE

. 2016 4e 3a 43 20 31 31 36 "n:c 116" Dateiname
```

Listing 1.10: Disketten formatieren mit 3 Kernel-Routinen

CLOSE ($FFC3 / 65475):
Wird eine Datei mittels der OPEN-Routine geöffnet, muß diese nach Beendigung der Ein-/Ausgabe wieder geschlossen werden. Die Voraussetzungen für die Funktion dieser Routine sind natürlich dieselben wie bei der OPEN-Routine. CLOSE schließt nur die logische Datei, deren Nummer sich im Akku befindet. Unser Beispielprogramm 10 ist also nicht ganz korrekt. Die Diskette wird zwar formatiert, aber die Datei nicht mehr geschlossen. Versucht man, die Routine erneut aufzurufen, tut sich nichts. Listing 1.11 zeigt nun das vollständige Programm zur Formatierung einer Diskette. Hier wird zusätzlich noch die ID angegeben, wodurch die Diskette vollständig formatiert wird. Das Beispielprogramm simuliert also folgende BASIC-Befehle:

```
OPEN 1,8,15,"n:c 116,00"
CLOSE 1
```

```
. 2000  a9 0a     lda #$0a   Länge des Namens gleich 10
. 2002  a0 20     ldy #$20   Adresse des
. 2004  a2 1b     ldx #$1b   ... Dateinamens ($201b)
. 2006  20 bd ff  jsr $ffbd  (SETNAM) Name übergeben
. 2009  a9 01     lda #$01   Logische Dateinummer gleich 1
. 200b  a2 08     ldx #$08   Geräteadresse gleich 8
. 200d  a0 0f     ldy #$0f   Sekundäradresse gleich 15
. 200f  20 ba ff  jsr $ffba  (SETLFS) Parameter übergeben
. 2012  20 c0 ff  jsr $ffc0  (OPEN)
. 2015  a9 01     lda #$01   Logische Dateinummer gleich 1
. 2017  20 c3 ff  jsr $ffc3  (CLOSE)
. 201a  60        rts        ENDE

. 201b 4e 3a 43 20 31 31 36 2c "n:c 116,
. 2023 30 30                  00"
```

Listing 1.11: Korrekte Formatierungs-Routine mit den Kernel-Routinen SETNAM, SETLFS, OPEN und CLOSE

CHKIN ($FFC6 / 65478):
Jede über die OPEN-Routine geöffnete, logische Datei kann über diese Routine als Eingabekanal definiert werden. Voraussetzung ist natürlich, daß es sich dabei um ein Eingabegerät handelt, da es sonst zu einer Fehlermeldung kommt und die Routine abgebrochen wird.

Diese Routine muß dann aufgerufen werden, wenn keine Daten von der Tastatur übergeben werden, und anschließend mit den Kernel-Routinen CHRIN oder GETIN gearbeitet werden soll. Wird diese Routine mit einem Gerät am seriellen Bus benutzt, dann wird automatisch die TALK-Adresse und die eventuell festgelegte Sekundär-Adresse an den Bus übergeben.

Vor Aufruf dieser Routine muß die Nummer der logischen Datei ins X-Register geladen werden. Bei Verwendung dieser Routine kann es zu folgenden drei Fehlermeldungen kommen:

```
I/O ERROR #3 : logische Datei nicht geöffnet
I/O ERROR #5 : Gerät nicht vorhanden
I/O ERROR #6 : logische Datei ist keine Eingabedatei
```

CKOUT ($FFC9 / 65481):
Mit der CKOUT-Routine wird die Ausgabe (z.B BSOUT) auf ein beliebiges Ausgabegerät geleitet. Bedingung ist hierbei, daß die logische Datei als Ausgabedatei definiert ist, und daß es sich bei dem angesprochenen Gerät um ein Ausgabegerät handelt. Ist dies nicht der Fall, erfolgt eine Fehlerausgabe und die Routine wird abgebrochen. Bei Ausgabe auf dem Bildschirm ist diese Routine nicht erforderlich. Bei Ausgabe über den seriellen Bus überträgt diese Routine automatisch die durch die OPEN-Routine festgelegte LISTEN- und Sekundär-Adresse.

Bevor diese Routine aufgerufen werden kann, muß über den OPEN-Befehl die LISTEN-Adresse und eventuell die Sekundär-Adresse übergeben werden. Anschließend muß dem X-Register die logische Dateinummer (dieselbe wie bei OPEN) übergeben werden. Erst jetzt kann die CKOUT-Routine aufgerufen werden. Auch hier können drei Fehlermeldungen auftreten:

```
I/O ERROR #3 : logische Datei nicht geöffnet
I/O ERROR #5 : Gerät nicht vorhanden
I/O ERROR #7 : logische Datei ist keine Ausgabedatei
```

CLRCH ($FFCC / 65484):
Die CLRCH-Routine wird zum Schließen aller offenen Kanäle und zur Initialisierung der Ein-/Ausgabekanäle benutzt. Die Standard-Ein-/Ausgabegeräte des C 16, C 116 oder Plus/4 sind Tastatur und Bildschirm. Wurden für den Datenaustausch andere Geräte als Ein-/Ausgabegeräte gesetzt, empfiehlt es sich, nach Beendigung der Arbeit wieder mit CLRCH auf die Standard-Geräte umzuschalten.

Ist einer der zu schließenden Kanäle der serielle Bus, so wird zunächst zum Schließen des Eingabekanals der UNTALK-Befehl gesendet. Die seriellen Ausgabekanäle wer-

den über die UNLISTEN-Routine geschlossen. Diese Routine kann ohne Vorbereitungen aufgerufen werden, da alle Kanäle geschlossen werden. Bei Verwendung der CLALL-Routine wird diese Routine automatisch mit aufgerufen.

BASIN ($FFCF / 65487):
Diese Kernel-Routine dient zur Dateneingabe über die Tastatur oder über ein Gerät, das zuvor mit CHKIN als Eingabegerät definiert wurde. Bei Eingabe über die Tastatur blinkt der Cursor. Es können ebenso wie bei der BASIC-Anweisung INPUT alle Zeichen eingegeben werden. Es ist also auch möglich, mit dem Cursor aus der Eingabezeile zu fahren oder den Bildschirm zu löschen.

Bei Aufruf der BASIN-Routine wird das eingegebene Zeichen an den Akku übergeben und gleichzeitig am Bildschirm dargestellt. Die kleine Beispiel-Routine (Listing 1.12) führt in einer Schleife so lange die BASIN-Routine aus, bis die Return-Taste gedrückt wird. Betätigt man nach dem Start des Programms mit SYS 8192 irgendeine Taste, so wird diese am Bildschirm dargestellt. Für den Editor haben diese Zeichen aber keinerlei Bedeutung.

```
. 2000  20 cf ff jsr $ffcf  (BASIN) Zeichen holen
. 2003  c9 0d    cmp #$0d   Zeichen gleich RETURN ?
. 2005  d0 f9    bne $2000  Nein, dann weiter
. 2007  60       rts        ENDE
```

Listing 1.12: Beispielprogramm BASIN

BSOUT ($FFD2 / 65490):
Dies ist die wohl am häufigsten verwendete Kernel-Routine aller Commodore-Computer. Die Routine gibt ein Zeichen, das im ASCII-Format im Akku steht, am Bildschirm aus. Wurde zuvor mit CKOUT ein anderes Gerät als Ausgabegerät definiert, so erfolgt die Ausgabe natürlich auf diesem Gerät. Die BSOUT-Routine überträgt jedes ASCII-Zeichen. So ist es auch möglich, den Bildschirm zu löschen, die Farbe zu ändern und alle anderen Funktionen, die im BASIC über PRINT CHR$(x) zu erreichen sind, durchzuführen. Bei der Bildschirmausgabe ist zu beachten, daß der Code an der momentanen Cursorposition ausgegeben wird. Handelt es sich dabei um ein Zeichen, wird die Cursorposition nach der Ausgabe um 1 Zeichen vorgerückt. Unser Beispielprogramm (Listing 1.13) gibt einen kurzen, farbigen und blinkenden Text am gelöschten Bildschirm aus:

```
. 2000  a0 ff     ldy #$ff       Zeiger in Texttabelle
. 2002  c8        iny            Zeiger um 1 erhöhen
. 2003  b9 0c 20  lda $200c,y    Holt Zeichen aus Tabelle
. 2006  20 d2 ff  jsr $ffd2      (BSOUT) Zeichen ausgeben
. 2009  d0 f7     bne $2002      Zeichen nicht 0 ?, dann weiter
. 200b  60        rts            ENDE

. 200c 93 11 11 11 11 11 20 20  Beispieltext:
. 2014 20 20 20 82 50 52 4f 47  ... ($00 = ENDE)
. 201c 52 41 4d 4d 20 1e 56 4f
```

```
. 2024 4e 20 1c 43 2e 51 2e 53
. 202c 90 11 11 11 11 00 00 00
```

Listing 1.13: Textausgabe mit der BSOUT-Routine

LOAD ($FFD5 / 65493):
Mit Hilfe dieser Kernel-Routine werden Daten direkt von einem beliebigen Eingabegerät in den Speicher des C 16, C 116 oder Plus/4 geladen. Sie kann auch für Verify-Zwecke verwendet werden, das heißt, die Daten im Speicher werden mit den Daten, die über die LOAD-Routine geholt werden, verglichen. Die Daten im Speicher bleiben dabei unverändert. Damit die Routine zwischen LOAD und VERIFY unterscheiden kann, muß der Akku entsprechend mit 0 oder 1 belegt werden:

```
LOAD :   Akku = 0
VERIFY : Akku = 1
```

Wird vor dem Aufruf der LOAD-Routine die OPEN-Routine mit Sekundäradresse 0 angesprungen, so wird die Ladeadresse des Daten-Files ignoriert. In diesem Fall muß die Ladeadresse im X- und Y-Register enthalten sein (X-Register = Low, Y-Register = High). Wählt man als Sekundäradresse 1 oder 2, so wird das Programm an die im File als Ladeadresse angegebene Position in den Speicher geladen. Nach Beendigung der LOAD-Routine enthalten X- und Y-Register die Adresse des letzten geschriebenen Bytes. Diese Adresse kann dann dem Variablenanfangs-Zeiger übergeben werden, wenn BASIC-Programme nachgeladen werden – die Routine zum LOAD-Befehl des BASIC 3.5 sorgt dafür bei $a834/$a836. Das Beispiel-Listing (Listing 1.14) lädt ein Programm mit dem Namen PROGRAMMNAME von Diskette an die absolute Ladeadresse (wie LOAD PROGRAMMNAME,8,1). Anschließend wird die Adresse des obersten benutzten RAM-Speichers in den Variablenanfangs-Zeiger geschrieben. Vor Aufruf der LOAD-Routine muß zunächst mit SETLFS eine logische Datei eröffnet, und mit SETNAM der Programmname übergeben werden. Im Akku ist das Flag für »Laden« zu setzen.

```
. 2000  a9 01     lda #$01   Logische Datei 1
. 2002  a2 08     ldx #$08   Gerätenummer 8
. 2004  a0 00     ldy #$01   Absolut Laden
. 2006  20 ba ff  jsr $ffba  (SETLFS)
. 2009  a9 0c     lda #$0c   Länge des Namens =: 12 Zeichen
. 200b  a2 22     ldx #$22   Zeiger auf
. 200d  a0 20     ldy #$20   ... Programmname ($2022)
. 200f  20 bd ff  jsr $ffbd  (SETNAM)
. 2012  a9 00     lda #$00   Flag für Laden
. 2014  a0 ff     ldy #$ff
. 2016  a2 ff     ldx #$ff
. 2018  20 d5 ff  jsr $ffd5  (LOAD)
. 201b  86 2d     stx $2d    Schreibt
. 201d  84 2e     sty $2e    ... Variablenanfang
. 201f  60        rts        ENDE
. 2020  ea        nop
. 2021  ea        nop
```

```
. 2022 50 52 4f 47 52 41 4d 4d   Programmname
. 202a 4e 41 4d 45 00 00 00 00
```

Listing 1.14: Beispielprogramm für absolutes Laden von Diskette

SAVE ($FFD8 / 65496):
Mit Hilfe der SAVE-Routine läßt sich ein beliebiger Speicherbereich auf Diskette bzw. Kassette speichern. Zuvor müssen wiederum die SETLFS- und SETNAM-Routine ausgerufen werden. Die SAVE-Routine benötigt mehr Parameter als zum Beispiel die LOAD-Routine. Die Startadresse des zu speichernden Bereichs muß in einer Zeropage-Adresse (2 Byte) abgelegt werden. Dem Akku wird anschließend diese Adresse übergeben. Die X- und Y-Register müssen die Endadresse +1 (!!) im Format (Low-Byte, High-Byte) enthalten. Folgendes Beispielprogramm (Listing 1.15) speichert den Grafik-Bereich ($2000-$3FFF) unter dem Namen GRAFIK auf Diskette ab. Das Programm wird mit SYS 4352 gestartet. Nach dem BASIC-Befehl SCNCLR kann das Grafikbild mit SYS 4394 wieder geladen werden.

```
        SAVE-Routine (SYS 4352)

. 1100  a9 01    lda #$01  Logische Dateinummer 1
. 1102  a2 08    ldx #$08  Geräteadresse 8 (Floppy)
. 1104  a0 00    ldy #$00  Sekundäradresse 0
. 1106  20 ba ff jsr $ffba (SETLFS)
. 1109  a9 06    lda #$06  Länge des Namens = 6
. 110b  a2 22    ldx #$22  Adresse des
. 110d  a0 11    ldy #$11  ... Filenamens ($1122)
. 110f  20 bd ff jsr $ffbd (SETNAM)
. 1112  a2 00    ldx #$00  Startadresse des Grafik-
. 1114  a0 20    ldy #$20  ... Bereichs ($2000)
. 1116  86 d8    stx $d8   ... nach $d8-$d9
. 1118  84 d9    sty $d9   ... kopieren
. 111a  a9 d8    lda #$d8  ... Zeropage-Adresse in Akku
. 111c  a0 40    ldy #$40  ... Endadresse ($4000)
. 111e  20 d8 ff jsr $ffd8 (SAVE)
. 1121  60       rts       ENDE
. 1122  47 52 41 46 49 4b  Filename "GRAFIK"

        LOAD-Routine (SYS 4394)

. 112a  a9 01    lda #$01  Logische Dateinummer 1
. 112c  a2 08    ldx #$08  Geräteadresse 8
. 112e  a0 01    ldy #$01  Absolut Laden
. 1130  20 ba ff jsr $ffba (SETLFS)
. 1133  a9 06    lda #$06  Länge des Namens = 6
. 1135  a2 22    ldx #$22  Adresse des
. 1137  a0 11    ldy #$11  ... Filenamens ($1122)
. 1139  20 bd ff jsr $ffbd (SETNAM)
. 113c  a9 00    lda #$00  Flag für LOAD setzen
```

```
. 113e  20 d5 ff jsr $ffd5 (LOAD)
. 1141  60       rts       ENDE
```

Listing 1.15: Programm zum Speichern und Laden des Grafikbildschirmes auf Diskette

Listing 1.16 erledigt dieselbe Aufgabe wie das Beispielprogramm 1.15. Nur kann in diesem Fall der Grafikschirm auf Kassette gespeichert werden:

```
        SAVE-Routine (SYS 4352)

. 1100  a9 01    lda #$01  Logische Dateinummer 1
. 1102  a2 08    ldx #$01  Geräteadresse 1 (Datasette)
. 1104  a0 00    ldy #$00  Sekundäradresse 0
. 1106  20 ba ff jsr $ffba (SETLFS)
. 1109  a9 06    lda #$06  Länge des Namens = 6
. 110b  a2 22    ldx #$22  Adresse des
. 110d  a0 11    ldy #$11  ... Filenamens ($1122)
. 110f  20 bd ff jsr $ffbd (SETNAM)
. 1112  a2 00    ldx #$00  Startadresse des Grafik-
. 1114  a0 20    ldy #$20  ... Bereichs ($2000)
. 1116  86 d8    stx $d8   ... nach $d8-$d9
. 1118  84 d9    sty $d9   ... kopieren
. 111a  a9 d8    lda #$d8  ... Zeropage-Adresse in Akku
. 111c  a0 40    ldy #$40  ... Endadresse ($4000)
. 111e  20 d8 ff jsr $ffd8 (SAVE)
. 1121  60       rts       ENDE

. 1122  47 52 41 46 49 4b  Filename "GRAFIK"

        LOAD-Routine (SYS 4394)

. 112a  a9 01    lda #$01  Logische Dateinummer 1
. 112c  a2 08    ldx #$01  Geräteadresse 1 (Datasette)
. 112e  a0 01    ldy #$01  Absolut Laden
. 1130  20 ba ff jsr $ffba (SETLFS)
. 1133  a9 06    lda #$06  Länge des Namens = 6
. 1135  a2 22    ldx #$22  Adresse des
. 1137  a0 11    ldy #$11  ... Filenamens ($1122)
. 1139  20 bd ff jsr $ffbd (SETNAM)
. 113c  a9 00    lda #$00  Flag für LOAD setzen
. 113e  20 d5 ff jsr $ffd5 (LOAD)
. 1141  60       rts       ENDE
```

Listing 1.16: Programm zum Speichern und Laden des Grafikbildschirmes auf Kassette

SETTIM ($FFDB / 65499):
C 16, C 116 und Plus/4 besitzen eine interne Uhr (Systemtaktgeber), die von einer Interrupt-Routine alle $^1/_{60}$ s aktualisiert wird. Um aber 24 Stunden darstellen zu können, benötigt man eine Zahl der Größe 24 * 3600 * 60 = 5184000. Diese Zahl kann nicht mehr mit 2 Byte dargestellt werden. Man benötigt also ein 3-Byte-Wort. Dieses wird beim Einschalten des Computers automatisch auf Null gesetzt, und jede $^1/_{60}$ s nachgestellt. Die Variable TI enthält ständig den aktuellen Wert der Systemuhr. Die Zeitvariable TI$ wird übrigens aus TI berechnet. Doch nun zur SETTIM-Routine. Sie dient zum Setzen der 3 Byte des Systemtaktgebers. Vor dem Aufruf dieser Routine muß die gewünschte Uhrzeit umgerechnet in $^1/_{60}$ s, in die 3 Register (Akku, X- und Y-Register) übergeben werden. Hierbei belegt das höchstwertige Byte das Y-Register. Das zweithöchste Byte belegt das X-Register, während der Akku mit dem niederwertigen Byte belegt wird. Das Beispielprogramm soll die Systemuhr auf 7 Uhr stellen. Dazu wird erst die Uhrzeit in $^1/_{60}$ s zerlegt 7 * 3600 * 60 = 1512000. Diese Zahl wird nun in 3 Byte zerlegt:

Höchstes Byte :	23 * 65535
Mittleres Byte :	18 * 256
Niederwertiges Byte :	64

Nach Übergabe in die 3 Register und Aufruf der SETTIM-Routine ist die neue Uhrzeit auf 7 Uhr eingestellt. Starten Sie nun das Beispielprogramm (Listing 1.17) mit SYS 8192 und überprüfen Sie die Uhrzeit

```
SYS 8192 : PRINT TI$
```

```
. 2000  a0 17    ldy #$17   Uhrzeit
. 2002  a2 12    ldx #$12   ... auf 7 Uhr
. 2004  a9 40    lda #$40   ... setzen
. 2006  20 db ff jsr $ffdb  (SETTIM)
. 2009  60       rts        ENDE
```

Listing 1.17: Uhrzeit auf 7 Uhr stellen

RDTIM ($FFDE / 65502):
Diese Routine dient zum Lesen des Systemtaktgebers. Die aktuelle Uhrzeit wird dabei in $^1/_{60}$ s umgerechnet und an Akku, X- und Y-Register übergeben. Auch hierbei wird das höchstwertige Byte im Y-Register abgelegt. Das zweithöchste Byte wird dem X-Register übergeben, während im Akku das niederwertige Byte abgelegt wird. Berechnet wird die aktuelle Uhrzeit mit folgender Formel:

Zeit in Sekunden:

```
(Y-Register * 65536 + X-Register * 256 + Akku ) / 60
```

Das Beispielprogramm (Listing 1.18) holt die aktuelle Systemzeit in die drei Prozessorregister und speichert diese nach $200D-$200F / 8205-8207. Starten Sie nun das Programm wie folgt:

```
SYS 8192 : PRINT TI
```

Merken Sie sich die ausgegebene Zahl und tippen Sie bitte folgende BASIC-Zeile im Direktmodus ein:

```
PRINT PEEK(8205) * 65535 + PEEK(8206) * 256 +PEEK(8207)
```

Werden nun beide Zahlen verglichen, dürften sich diese um höchstens 3/60 Sekunden unterscheiden.

```
. 2000  20 de ff jsr $ffde
. 2003  8c 0d 20 sty $200d
. 2006  8e 0e 20 stx $200e
. 2009  8d 0f 20 sta $200f
. 200c  60       rts
```

Listing 1.18: Beispielprogramm zur Kernel-Routine RDTIM

STOP ($FFE1 / 65505):
Diese Routine ist nur in Verbindung mit der UDTIM-Routine wirksam. Wurde während der Abarbeitung der UDTIM-Routine (Systemtaktgeber aktualisieren) die Stop-Taste gedrückt, setzt die STOP-Routine das Zero-Flag. Die Vorgehensweise ist sinnvoll, da die UDTIM-Routine von der Kernel-Interrupt-Routine aufgerufen wird, und somit jederzeit eine Abfrage der Stop-Taste gewährleistet ist. Vor Benutzung der STOP-Routine muß also gesichert sein, daß die UDTIM-Routine aufgerufen wurde. Anschließend ist lediglich zu prüfen, ob das Zero-Bit gesetzt ist oder nicht. Die STOP-Routine hat aber noch eine zweite Aufgabe. Sie setzt alle Kanäle auf ihre Standardwerte zurück.

GETIN ($FFE4 / 65508):
GETIN holt ein Byte aus dem Tastaturpuffer und übergibt dieses als ASCII-Wert an den Akku. Ist der Akku gleich Null, war der Tastaturpuffer leer. Im Tastaturpuffer finden maximal 10 Zeichen Platz. Diese werden mittels der SCNKEY-Routine in den Puffer übertragen. Ist der Tastaturpuffer voll, werden die folgenden Tasten so lange überlesen, bis mittels der GETIN-Routine Zeichen auf dem Puffer ausgelesen werden.

Die GETIN-Routine kann aber auch Zeichen von anderen Eingabegeräten lesen. Dazu muß natürlich erst mit der OPEN-Routine eine logische Datei eröffnet und diese mittels CHKIN als Eingabedatei deklariert werden. Während diese Routine besonders zum Einlesen der Tastatur und des RS-232-Kanals geeignet ist, empfiehlt es sich, bei Dateneingabe vom seriellen Bus oder von Kassette die BASIN-Routine zu verwenden. Wird trotzdem mit GETIN gearbeitet, sollte auf keinen Fall vergessen werden, mittels der READST-Routine mögliche Fehler zu erkennen und auszuwerten. Das Beispielprogramm (Listing 1.19) holt zunächst die aktuelle Taste in den Tastaturpuffer. Mit GETIN wird die Taste als ASCII-Code aus dem Tastaturpuffer in den Akku übertragen. Ist der Akku gleich Null (Tastaturpuffer leer), so wird erneut die Tastatur abgefragt. Ansonsten wird das entsprechende Zeichen mit Hilfe der

BSOUT-Routine am Bildschirm ausgegeben. Das Programm wird mit SYS 8192 gestartet und kann mit der Stop-Taste unterbrochen werden.

```
. 2000  20 9c ff jsr $ff9c   (SCNKEY) Tastatur abfragen
. 2003  20 e4 ff jsr $ffe4   (GETIN) ASCII-Code in Akku
. 2006  f0 f8    beq $2000   Keine Taste gedrückt?, dann nochmal
. 2008  20 d2 ff jsr $ffd2   (BSOUT) Zeichen ausgeben
. 200b  c9 03    cmp #$03    STOP-Taste gedrückt ?
. 200d  d0 f1    bne $2000   Nein, dann nächstes Zeichen
. 200f  60       rts         Abbruch durch STOP
```

Listing 1.19: Beispielprogramm zur Übergabe des Tastaturcodes als ASCII-Wert mittels GETIN.

CLALL ($FFE7 / 65511):
Die Kernel-Routine CLALL schließt alle offenen Dateien. Außerdem werden alle Zeiger in der Tabelle der offenen Dateien zurückgestellt. Beim Aufruf der CLALL-Routine wird automatisch die CLRCHN-Routine angesprungen, wodurch auch alle Ein-/Ausgabekanäle initialisiert werden. Diese Routine sollte am Anfang eines jeden Maschinenprogrammes stehen, in dem mit logischen Dateien gearbeitet wird, um sicherzustellen, daß keine offenen Dateien oder falsche Ein-/Ausgabekanäle aktiv sind. Die Routine kann ohne jegliche Vorarbeiten aufgerufen werden.

Achtung: Die I/O-Files werden zwar computerintern geschlossen, um Fehler wie »FILE OPEN ERROR« abzufangen; ein ordnungsgemäßes Schließen von Diskettendateien erfolgt jedoch nicht! CLALL ist also kein vollwertiger Ersatz für CLOSE.

UDTIM ($FFEA / 65514):
Diese Routine aktualisiert die Systemuhr des C 16, C 116 und Plus/4. Normalerweise wird die UDTIM-Routine alle $^1/_{60}$ s automatisch von der Kernel-Interrupt-Routine aufgerufen. Sollte Ihr eigenes Programm auf diesen Kernel-Interrupt verzichten und mit eigenen Interrupts arbeiten, so muß diese Routine eingebunden werden, um sicherzustellen, daß die Systemuhr weiterhin aktualisiert wird. Diese Routine ist außerdem die Voraussetzung für die Abfrage der Stop-Taste. Wird nach Aufruf dieser Routine die STOP-Routine angesprungen, setzt diese bei betätigter Stop-Taste das Zero-Bit. Auch diese Routine kann ohne Vorbereitungen aufgerufen werden. Soll jedoch die Stop-Taste wirksam bleiben, so empfiehlt es sich, anschließend die STOP-Routine aufzurufen. Das Beispielprogramm (Listing 1.20) sperrt zunächst die Kernel-Interrupt-Routine und stellt anschließend in einer Endlosschleife die Zeit jeweils um $^1/_{60}$ s weiter. Um das Programm unterbrechen zu können, wird noch die STOP-Routine aufgerufen. Starten Sie das Programm (Listing 1.20) wie folgt:

```
LET TI$= "000000" : SYS 8192
```

Wenn Sie das Programm nach etwa einer Minute mit STOP unterbrechen und mit

```
PRINT TI$
```

die Uhrzeit abfragen, werden Sie feststellen, daß auf unserer Systemuhr in einer Minute zirka 2 Stunden 20 Minuten vergangen sind.

```
. 2000  78       sei         Interrupt sperren
. 2001  20 ea ff jsr $ffea   (UDTIM) Zeit stellen
. 2004  20 e1 ff jsr $ffe1   (STOP) Stop-Taste prüfen
. 2007  d0 f8    bne $2001   Kein STOP, dann weiter
. 2009  58       cli         Interrupt ermöglichen
. 200a  60       rts         ENDE
```

Listing 1.20: Beispielprogramm »Turbozeit« zur Demonstration der UDTIM- und STOP-Routine

SCREEN ($FFED / 65517):
Zur Ermittlung des Bildschirmformates wird die SCREEN-Routine verwendet. Sie übergibt die Bildschirmbreite (Standard: 40 Spalten) dem X-Register und die Bildschirmhöhe (Standard: 25 Zeilen) dem Y-Register. Vor Aufruf der Routine sind keine Vorarbeiten zu treffen. Anschließend müssen lediglich X- und Y-Register ausgewertet werden. Listing 1.21 gibt das aktuelle Bildschirmformat am Bildschirm aus. Das Programm wird mit SYS 8192 gestartet.

```
. 2000  a0 ff    ldy #$ff    Zeiger in Texttabelle
. 2002  c8       iny         Zeiger um 1 erhöhen
. 2003  b9 50 20 lda $2050,y Holt Zeichen aus Tabelle
. 2006  20 d2 ff jsr $ffd2   (BSOUT) Zeichen ausgeben
. 2009  d0 f7    bne $2002   Zeichen nicht 0 ?, dann weiter
. 200b  20 ed ff jsr $ffed   (SCREEN) Bildschirmformat
. 200e  98       tya         Bildschirmhöhe
. 200f  48       pha         ... auf Stack retten
. 2010  a9 00    lda #$00    Breite nach Akku, X-Reg.
. 2012  20 5f a4 jsr $a45f   Integerzahl ausgeben
. 2015  a9 2a    lda #$2a    Holt '*'-Zeichen
. 2017  20 d2 ff jsr $ffd2   (BSOUT) Zeichen ausgeben
. 201a  68       pla         Holt Bildschirmhöhe
. 201b  aa       tax         Bildschirmhöhe nach
. 201c  a9 00    lda #$00    ... Akku, X-Register
. 201e  20 5f a4 jsr $a45f   Integerzahl ausgeben
. 2021  60       rts         ENDE

. 2050 93 11 11 11 11 11 20 20 TEXT
. 2058 42 49 4c 44 53 43 48 49
. 2060 52 4d 46 4f 52 4d 41 54
. 2068 20 00 00 00 00 00 00 00
```

Listing 1.21: Bildschirmformat ausgeben

PLOT ($FFF0 / 65520):
Diese Kernel-Routine hat zwei Aufgaben. Sie dient sowohl zum Lesen als auch zum Setzen der aktuellen Cursorposition. Ist das Carry-Bit gesetzt, so wird die Cursorposition ausgelesen und dem X- und Y-Register übergeben. Hierbei enthält das Y-Register die Spaltennummer (0-39) und das X-Register die Zeilennummer (0-24). Beispielprogramm 1.22 gibt je nach Position des SYS-Befehls die aktuelle Cursorposition (unter der SYS-Zeile) aus. Auch dieses Programm wird mit SYS 8192 gestartet.

```
. 2000  38       sec         Holt
. 2001  20 f0 ff jsr $fff0   ... Cursorposition
. 2004  8a       txa         Cursorposition
. 2005  48       pha         ... auf
. 2006  98       tya         ... Stack
. 2007  48       pha         ... retten
. 2008  a0 ff    ldy #$ff    Zeiger in Texttabelle
. 200a  c8       iny         Zeiger um 1 erhöhen
. 200b  b9 50 20 lda $2050,y Zeichen aus Tabelle holen
. 200e  20 d2 ff jsr $ffd2   Zeichen ausgeben
. 2011  d0 f7    bne $200a   Zeichen nicht 0 ?, dann weiter
. 2013  68       pla         Holt Spalte
. 2014  aa       tax         ... vom Stack
. 2015  a9 00    lda #$00    Spalte nach (Akku, X-Reg.)
. 2017  20 5f a4 jsr $a45f   Integerzahl ausgeben
. 201a  a9 3a    lda #$3a    Holt ':'- Zeichen
. 201c  20 d2 ff jsr $ffd2   Zeichen ausgeben
. 201f  68       pla         Holt Zeile
. 2020  aa       tax         ... vom Stack
. 2021  a9 00    lda #$00    Zeile nach (Akku, X-Reg.)
. 2023  20 5f a4 jsr $a45f   Integerzahl ausgeben
. 2026  60       rts         ENDE

. 2050 43 55 52 53 4f 52 50 4f Text
. 2058 53 49 54 49 4f 4e 3a 20
. 2060 00 00 00 00 00 00 00 00
```

Listing 1.22: Aktuelle Cursorposition ausgeben

Ist beim Aufruf der PLOT-Routine das Carry-Bit gleich Null, so wird der Cursor an die durch X- und Y-Register gegebene Position gesetzt. Auch hier enthält das Y-Register die Spaltennummer (0-39) und das X-Register die Zeilennummer (0-24). Vor Aufruf der PLOT-Routine bei gelöschtem Carry-Bit muß den beiden Registern die gewünschte Cursorposition übergeben werden. Das Beispielprogramm (Listing 1.23) schreibt, nach dem Starten mit SYS 8192, ein Sternchen in die 10. Zeile und 20. Spalte.

```
. 2000  a2 09    ldx #$09    Zeile 10
. 2002  a0 13    ldy #$13    Spalte 20
. 2004  18       clc         Carry = 0
. 2005  20 f0 ff jsr $fff0   (PLOT) Cursor setzen
. 2008  a9 2a    lda #$2a    Holt '*'-Zeichen
. 200a  20 d2 ff jsr $ffd2   (BSOUT) Zeichen ausgeben
. 200d  60       rts         ENDE
```

Listing 1.23: Beispielprogramm für PLOT: Sternchen in 10. Zeile und 20. Spalte setzen

IOBASE ($FFF3 / 65523):
Diese letzte Kernel-Routine übergibt dem X- und Y-Register die Adresse des Speicherabschnittes, ab welchem sich die Ein-/Ausgaberegister befinden. Dabei enthält das X-Register das Low-Byte und das Y-Register das High-Byte der Adresse. Das

Beispielprogramm 1.24 ermittelt die Basisadresse der Ein-/Ausgaberegister und gibt diese als Dezimalzahl am Bildschirm aus.

```
. 2000  20 f3 ff jsr $fff3   (IOBASE)
. 2003  98       tya         Adresse nach Akku, Y-Reg.
. 2004  20 5f a4 jsr $a45f   Integerzahl ausgeben
. 2007  60       rts         ENDE
```

Listing 1.24: Adresse der Ein-/Ausgaberegister ermitteln

Soweit zu den 39 Kernel-Routinen des C 16, C 116 oder Plus/4. Tabelle 1.3 gibt noch eine Übersicht über Stack-Bedarf und über die beeinflußten Register der einzelnen Kernel-Routinen.

Da nicht alle wichtigen Routinen in der Kernel-Sprungtabelle untergebracht sind, werden im folgenden noch zusätzliche ROM-Routinen erklärt. Alle folgenden Routinen treffen nur für den C 16, C 116 und Plus/4 zu.

INTOUT ($A45F / 42079):
Eine sehr häufig benutzte Routine ist die Ausgabe einer Integerzahl. Die Einsprungadresse liegt bei $A45F. Zuvor ist dem Akku und dem X-Register der entsprechende Wert zu übergeben. Wird nun die Routine aufgerufen, erscheint die gewünschte Integerzahl am Bildschirm.

PUTHEX ($FB10 / 64272):
Diese Routine gibt den Wert im Akku als zweistellige Hex-Zahl aus.

PUTWRD ($FAFF / 64255):
Diese Routine gibt eine vierstellige Hex-Zahl aus. Dabei muß der Akku mit dem Low-Byte und das X-Register mit dem High-Byte der entsprechenden Zahl belegt werden.

RESET ($FFF9 / 65529):
Diese Routine führt ein System-Reset durch. Sie hat also die gleiche Wirkung wie die Betätigung der Reset-Taste am Computer.

INFOUT ($FF4F / 65359):
Diese Routine gibt eine unmittelbar folgende Meldung aus. Die Meldung besteht aus ASCII-Zeichen und folgt direkt dem JSR $FF4F-Befehl. Die Meldung darf nicht länger als 255 Zeichen sein und muß mit $00 enden. Nach Ausgabe der Meldung wird das Programm automatisch nach dem $00-Code fortgesetzt.

Soviel zu den ROM-Routinen des C 16, C 116 und Plus/4. Mit Hilfe dieser ausführlichen Beschreibung können viele Maschinenprogramme vereinfacht werden. Es dürfte nun ebenso möglich sein, Programme von anderen Commodore-Computern auf diese drei Geräte umzuschreiben. Für schwierigere Aufgaben hilft immer noch ein Blick ins ROM-Listing.

2 Speicherverwaltung

2.1 Das Bankswitching

Im Gegensatz zu den C 16- und Plus/4-Vorgängern (C 64, VC 20), können diese Computer auf mehr als nur 64 Kbyte Speicherplatz zugreifen. Normalerweise stößt man bei Computern mit einem 16-Bit-Adreßbus sehr schnell an die Grenzen des Möglichen. Theoretisch können maximal 2 hoch 16 (65536) Speicherstellen (64 Kbyte) adressiert werden. Diese Tatsache bleibt beim C 16, C 116 und Plus/4 auch nach wie vor erhalten. Für den Prozessor existieren zu jeder Zeit genau 64 Kbyte Speicherplatz.

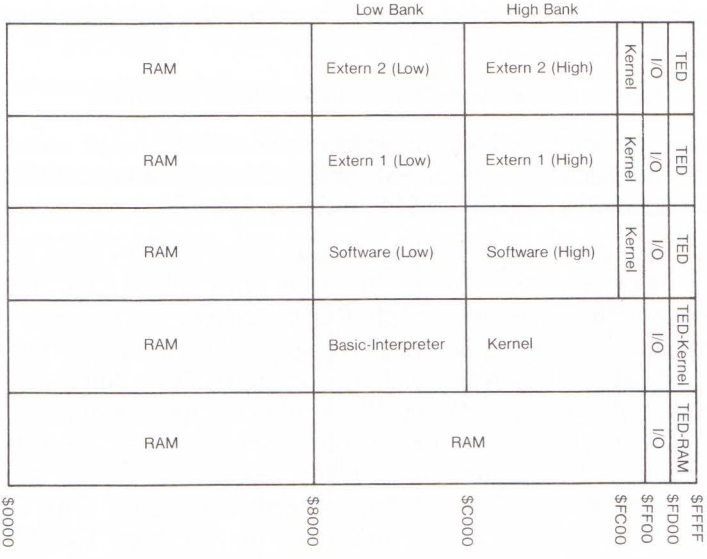

Bild 2.1: Speicheraufbau des Plus/4 (C16/C116)

Wenn Sie sich nun fragen, wie es möglich ist, doch auf mehr als 64 Kbyte zuzugreifen, so heißt das Zauberwort »Bankswitching«. Aus Bild 2.1 läßt sich ersehen, daß für den Prozessor immer nur der Speicherbereich $0000-$FFFF existiert. Die einzelnen Speicherbauteile können beliebig kombiniert werden. Das Umschalten zwischen den einzelnen Bausteinen erfolgt per Software. Dieser Vorgang nennt sich »Bankswitching« (zu deutsch: Umschalten zwischen Speicherblöcken).

Wie aus Bild 2.1 zu ersehen ist, erstreckt sich der RAM-Speicher des Plus/4 (C 16 mit Erweiterung) von $0000-$FCFF. Gleichzeitig liegt aber das Betriebssystem »darüber«, im Bereich $8000-$FFFF. Da kommt die Frage auf, wie es möglich ist, daß der Plus/4 trotz eingeschalteten BASIC-Interpreters, auf fast 64 Kbyte BASIC-Speicher zugreifen kann. Dies ist durch elegantes Umschalten zwischen RAM und ROM im laufenden Programm möglich. Der TED-Chip besitzt 2 Register, die diese Aufgabe erledigen:

```
Register $FF3E: ROM einschalten
Register $FF3F: RAM einschalten
```

Das Umschalten erfolgt durch Schreibzugriff auf die entsprechenden Register. Versucht man allerdings von BASIC aus, mit POKE 65343,X das RAM einzuschalten, wird man feststellen, daß der Computer abgestürzt ist, weil sich der Prozessor in den Untiefen des RAM-Speichers verirrt hat. Das Umschalten zwischen RAM und ROM bleibt also ausschließlich dem Maschinenprogramm vorenthalten. Dabei ist aber ebenfalls darauf zu achten, daß während des RAM-Zugriffes kein Sprung ins Betriebssystem erfolgt, weil sich an der betreffenden Stelle im Moment RAM befindet. Damit nicht versucht wird, auf das Betriebssystem zuzugreifen, muß vor Umschaltung auf RAM unbedingt der Interrupt gesperrt werden. Dies geschieht durch Setzen des I-Flags mittels des SEI-Befehls.

Drei Speicherbereiche werden vom Bankswitching nicht betroffen:

- Der RAM-Bereich $0000-$7FFF bleibt immer erhalten.
- Die Ein-/Ausgaberegister $FD00-$FDFF sind in jeder Bank enthalten.
- Auch die TED-Register $FF00-$FF3F sind in jeder Bank zugänglich.

Der ROM-Bereich des C 16, C 116 und Plus/4 ($8000-$FFFF) kann sich aus 16 unterschiedlichen Kombinationen zusammensetzen. Wird mittels STA $FF3E ins ROM umgeschaltet, so wird das gerade aktuelle ROM aktiviert. Die Nummer der aktuellen ROM-Konfiguration findet sich in der Zeropage-Adresse $FB. Jede der 16 Speicherkonfigurationen hat eine I/O-Adresse als »Einschalter«. Diese Adressen liegen im Bereich $FDD0-$FDDF. Auch in diesem Fall werden die ROM-Konfigurationen durch einen Schreibbefehl in die entsprechende Adresse eingeschaltet. Zusätzlich muß noch die Nummer der ROM-Konfiguration nach $FB geschrieben werden, da der Interrupt alle $^1/_{60}$ s die aktuelle Speicherkonfiguration erneut herstellt. Wird $FB nicht geändert, schaltet der Interrupt sofort wieder auf die vorherige ROM-Konfiguration zurück. Die möglichen ROM-Konfigurationen können aus Tabelle 2.1 ersehen werden.

MODUL	ADR.	Low-Bank	High-Bank	
0	$FDD0	BASIC-Interpreter	Betriebssystem	Standard
1	$FDD1	Software (Low)	Betriebssystem	
2	$FDD2	Extern 1 (Low)	Betriebssystem	
3	$FDD3	Extern 2 (Low)	Betriebssystem	
4	$FDD4	BASIC-Interpreter	Software (High)	
5	$FDD5	Software (Low)	Software (High)	Plus/4-Software
6	$FDD6	Extern 1 (Low)	Software (High)	
7	$FDD7	Extern 2 (Low)	Software (High)	
8	$FDD8	BASIC-Interpreter	Extern 1 (High)	
9	$FDD9	Software (Low)	Extern 1 (High)	
10	$FDDA	Extern 1 (Low)	Extern 1 (High)	
11	$FDDB	Extern 2 (Low)	Extern 1 (High)	
12	$FDDC	BASIC-Interpreter	Extern 2 (High)	
13	$FDDD	Software (Low)	Extern 2 (High)	
14	$FDDE	Extern 1 (Low)	Extern 2 (High)	
15	$FDDF	Extern 2 (Low)	Extern 2 (High)	

Tabelle 2.1: Mögliche Speicherkonfigurationen

Mit folgendem Beispielprogramm (Listing 2.1) wird beim Plus/4 die Systemsoftware gestartet. Dazu wird die Modulnummer 5 (siehe Tabelle 2.1) nach $FB geschrieben und mittels eines Schreibbefehls (STA $FDD5) die ROM-Konfiguration 5 eingeschaltet. Anschließend kann die Software mit JMP $8003 gestartet werden.

```
. 2000  a9 05    lda #$05      Holt Modulnummer 5
. 2002  85 fb    sta $fb       Schreibt Modulnummer nach $FB
. 2004  8d d5 fd sta $fdd5     Schaltet Modul ein
. 2007  4c 03 80 jmp $8003     Startet Plus/4-Software
```

Listing 2.1: Einschalten der Plus/4-Software

Beim Umschalten zwischen den Modulen ist zu beachten, daß der Speicherbereich $FC00-$FCFF immer wichtige Betriebssytem-Routinen enthält. Dieser Bereich kann also ebensowenig wie die Ein-/Ausgabe-Adressen oder TED-Register, für eigene Programme verwendet werden. Das hat auch einen guten Grund: In diesem Bereich des Kernels befinden sich wichtige Routinen, die in jedem selbstgeschriebenen Betriebssystem verwendet werden müssen. Der Speicherbereich beherbergt die Routinen Modul-Reset , Modul-Initialisierung, Modul-Zugriff, Modul-Aufruf und Teile der Interrupt-Routine (Puls). Die Puls-Routine wird bei jedem Interrupt abgearbeitet. Sie schaltet das Betriebssystem-ROM ein und startet die Interrupt-Routine. Anschließend wird wieder auf die aktuelle ROM-Konfiguration zurückgeschaltet. Die Puls-Routine ($FCB3) kann also sehr gut für eigene Software-Module (Extern 1/2) benutzt werden. Die Routine Modul-Zugriff ($FC7F) erlaubt es, aus einem beliebigen Modul Werte zu übernehmen. Die Routine Modul-Aufruf ($FC89) bearbeitet ein Unterprogramm in einem beliebigen Modul. Zum Starten eines Moduls kann die Routine ab $FCC9 verwendet werden. Für diesen Kernel-Teil besteht eine eigene

Sprungtabelle am Ende des Speicherbereichs ($FCF1-$FCFF). Um die Funktionsweise dieser Routinen zu ergründen, ist sicher ein Blick ins ROM-Listing behilflich.

2.2 C 16/C 116 contra Plus/4 - Die Unterschiede

Im Gegensatz zum Plus/4 wurden der C 16 und C 116 nicht mit integrierter Software (Tabellenkalkulation, Textverarbeitung und Dateiverwaltung) ausgestattet. Dies ist kein gravierender Nachteil, da sich über die Qualität dieser Programme streiten läßt. Von größerer Bedeutung ist der Speicherplatz-Unterschied. Die 16 Kbyte des C 16 machen sich gegen die 64 Kbyte RAM-Speicher des Plus/4 kläglich bemerkbar. Vor allem bei eingeschaltetem Grafik-Modus fällt dieser Unterschied schwer ins Gewicht. Tabelle 2.2 enthält eine Übersicht über die RAM-Verteilung beider Geräte.

	Plus/4	C 16/C 116
Ohne Grafik:	$1000-$FCFF: BASIC-RAM	$1000-$3FFF: BASIC-RAM
Mit Grafik:	$1800-$1BFF: Helligkeit $1C00-$1FFF: Farbe $2000-$3FFF: Grafik $4000-$FCFF: BASIC-RAM	$1000-$17FF: BASIC-RAM $1800-$1BFF: Helligkeit $1C00-$1FFF: Farbe $2000-$3FFF: Grafik

Tabelle 2.2: RAM-Verteilung bei C 16/C 116 und Plus/4

Vergleicht man alle 3 Computer, stellt man fest, daß beim C 16 und C 116 offensichtlich der User-Port vergessen wurde. Dies ist insofern erstaunlich, als im Betriebssystem die RS-232-Routinen voll enthalten sind. Da alle seriellen Schnittstellen dieser Art in der Regel am User-Port angeschlossen werden, sind diese Routinen beim C 16 und C 116 sinnlos.

C 16, C 116, Plus/4 - ALT oder NEU ?

Beim Erstellen dieses Buches mußte ich leider feststellen, daß es zwei verschiedene Betriebssysteme für diese Computer gibt. Im ersten Betriebssystem, das allem Anschein nach nur ganz am Anfang ausgeliefert wurde, befinden sich einige gravierende Fehler. In der alten Version ist vor allem die Routine zur Bearbeitung von DS$ fehlerhaft. Beim Anlegen von DS$ wird der Deskriptor-Zeiger nicht korrekt angelegt. Ist nun beim C 16, C 116 oder Plus/4 eine Garbage-Collection notwendig, um die ungültigen Strings zu beseitigen, verirrt sich der Prozessor durch die falsche R-Adresse in den Untiefen des Betriebssystems. Dieser Fehler und andere Unkorrektheiten wurden in der neueren Version des C 16, C 116 und Plus/4 korrigiert.

Um festzustellen, ob Ihr Computer diesen Fehler besitzt, müssen Sie lediglich feststellen, ob DS$ als gültige Stringvariable angelegt werden kann. Dies geht folgendermaßen:

- Zuerst NEW eingeben.
- Anschließend mit PRINT FRE(0) den Speicherplatz prüfen. (Ausgegebene Zahl bitte merken !)
- Dann mit PRINT DS$ die DS$-Stringvariable anlegen. (Floppy muß dabei eingeschaltet sein!!)
- Nun erneut mittels PRINT FRE(0) den freien Speicherplatz abrufen.

Wenn diese Zahl um den Wert 42 kleiner geworden ist (Stringlänge 40 + 2 Byte für R-Zeiger), so ist Ihr Computer in Ordnung. Bei einem anderen Ergebnis, oder beim Absturz des Computers, besitzen Sie das ältere Modell.

Von diesem Fehler sind außerdem noch die Befehle HEADER und SCRATCH betroffen, da auch sie die Variable DS$ anlegen.

Im nun folgenden ROM-Listing werden beide Betriebssysteme abgedruckt. Die wenigen Unterschiede zur alten Betriebssystem-Version sind an gegebener Stelle grau unterlegt.

3 System-Adressen

Ein sehr wichtiger Teil der Computer ist die Zeropage (Seite 0: $0000-$00FF). Für die ersten CBM-Rechner (PET, CBM 30xx und CBM 40xx) reichten die 256 Byte der Zeropage vollkommen aus, um alle wichtigen Adressen dort unterzubringen. Mit zunehmenden Fähigkeiten der Commodore-Nachfolger wuchs auch der Bedarf an System-Adressen. Die System-Adressen des C 16, C 116 und Plus/4 umfassen den Speicherbereich 0-2047 ($0000-$07FF). Der Bereich $0100-$07FF wird auch als »erweiterte Zeropage« bezeichnet. In der Zeropage selbst befinden sich die wichtigsten, am häufigsten benötigten Flags und Parameter. Der Prozessor verfügt über spezielle Zeropage-Befehle, die ein schnelleres Abarbeiten, bedingt durch kürzere Befehle und Zugriffszeiten, ermöglichen.

Die System-Adressen beinhalten alle wichtigen Parameter, Flags und Betriebssystem-Vektoren. Da die System-Adressen im RAM-Bereich liegen, kann durch Veränderung von Vektoren oder Parametern fast beliebig ins »Computergeschehen« eingegriffen werden. Die System-Adressen können nicht nur in Maschinensprache verändert werden. Auch von BASIC aus lassen sich diese Adressen mittels POKE-Befehl manipulieren.

Beim Experimentieren mit den System-Adressen ist jedoch Vorsicht geboten: Unkontrollierte Änderungen führen fast immer zu einem Totalabsturz des Computers. Da hilft dann nur noch die Reset-Taste.

Damit Sie sich besser in der Zeropage und der »erweiterten« Zeropage zurechtfinden, werden im folgenden die System-Adressen mit ausführlicher Beschreibung aufgezeigt.

3.1 Zeropage

Dezimal	Hexadezimal	Beschreibung
Ø	$ØØØØ	Datenrichtungsregister des 75Ø1
1	$ØØØ1	Ein-/Ausgabe-Port des 75Ø1
2	$ØØØ2	Flag für Schleifen
3- 4	$ØØØ3-$ØØØ4	Neue Startadresse (Renumber)
5- 6	$ØØØ5-$ØØØ6	Schrittweite (Renumber)
7	$ØØØ7	Gesuchtes Zeichen
8	$ØØØ8	Flag für Anführungszeichen-Modus, Zeilennr.
9	$ØØØ9	TAB-Spaltenzähler
1Ø	$ØØØA	LOAD/VERIFY-Flag: Ø = Load, 1 = Verify
11	$ØØØB	Zeiger für Eingabepuffer, Anzahl der Elemente
12	$ØØØC	Flag für Standard-DIM
13	$ØØØD	Datentyp: $FF=String, $ØØ=Numerisch
14	$ØØØE	Datentyp: $8Ø=Integer, $ØØ=Fließkomma
15	$ØØØF	Flag für DATA/LIST
16	$ØØ1Ø	Flag: Element/FNx-Flag
17	$ØØ11	Flag: $ØØ=INPUT, $4Ø=GET, $98=READ
18	$ØØ12	Flag: Vorzeichen des ATN
19	$ØØ13	Flag: Input-Prompt
2Ø- 21	$ØØ14-$ØØ15	2-Byte-Adresse, temporärer Speicher für RENUMBER
22	$ØØ16	Zeiger auf temporären Stringstapel
23- 24	$ØØ17-$ØØ18	Letzter temporärer Stringvektor
25- 33	$ØØ19-$ØØ21	Stapel für temporäre Strings
34- 35	$ØØ22-$ØØ23	Bereich für Hilfszeiger 1
36- 37	$ØØ24-$ØØ25	Bereich für Hilfszeiger 2
38	$ØØ26	Bereich für Produkt bei Multiplikation
39	$ØØ27	Bereich für Produkt bei Multiplikation
4Ø	$ØØ28	Bereich für Produkt bei Multiplikation
41	$ØØ29	Bereich für Produkt bei Multiplikation
42	$ØØ2A	Bereich für Produkt bei Multiplikation
43- 44	$ØØ2B-$ØØ2C	Zeiger auf BASIC-Anfang
45- 46	$ØØ2D-$ØØ2E	Zeiger auf Variablen-Anfang = BASIC-Programmende
47- 48	$ØØ2F-$ØØ3Ø	Zeiger auf Beginn der Arrays
49- 5Ø	$ØØ31-$ØØ32	Zeiger auf Ende der Arrays (+1)
51- 52	$ØØ33-$ØØ34	Zeiger auf Stringspeicher
53- 54	$ØØ35-$ØØ36	Hilfszeiger für Strings
55- 56	$ØØ37-$ØØ38	Zeiger auf oberste Speichergrenze
57- 58	$ØØ39-$ØØ3A	Laufende BASIC-Zeilennummer
59- 6Ø	$ØØ3B-$ØØ3C	Textpointer
61- 62	$ØØ3D-$ØØ3E	Zeiger auf BASIC-Statement für CONT
63- 64	$ØØ3F-$ØØ4Ø	Nummer der aktuellen DATA-Zeile
65- 66	$ØØ41-$ØØ42	Adresse des aktuellen DATA-Elementes
67- 68	$ØØ43-$ØØ44	Sprungvektor für INPUT
69- 7Ø	$ØØ45-$ØØ46	Aktueller Variablenname

71- 72	$0047-$0048	Adresse der aktuellen Variablen
73- 74	$0049-$004A	Variablenzeiger für FOR...NEXT
75- 76	$004B-$004C	Zwischenspeicher für Basic-Zeiger
77	$004D	Akkumulator für Vergleichssymbole
78- 79	$004E-$004F	Arbeitsbereich (Zeiger etc.)
80- 81	$0050-$0051	Arbeitsbereich (Zeiger etc.)
82	$0052	Arbeitsbereich (Zeiger etc.)
83	$0053	Grafik-Modus, HELP-Flag
84- 86	$0054-$0056	Sprungvektor für Funktionen
87- 96	$0057-$0060	Bereich für numerische Operationen
97	$0061	Fließkomma-Akkumulator 1 (FAC): Exponent
98- 101	$0062-$0065	Fließkomma-Akkumulator 1 (FAC): Mantisse
102	$0066	Fließkomma-Akkumulator 1 (FAC): Vorzeichen
103	$0067	Zeiger für Polynom-Auswertung
104	$0068	Fließkomma-Akkumulator 1 Überlauf
105	$0069	Fließkomma-Akkumulator 2 Exponent
106- 109	$006A-$006D	Fließkomma-Akkumulator 2 Mantisse
110	$006E	Fließkomma-Akkumulator 2 Vorzeichen
111	$006F	Vorzeichenvergleich Akku 1 mit Akku 2
112	$0070	Fließkomma-Akkumulator 1 niederwertige Stelle
113- 114	$0071-$0072	Kassettenpuffer Länge/Zeiger
115- 116	$0073-$0074	Automatisches Zeileninkrement (0=AUS)
117	$0075	Grafik-Flag (0=Text, 255=Grafik)
118- 120	$0076-$0078	Arbeitsbereich (MID$ etc.)
121- 123	$0079-$007B	Darstellung von DS$
124- 125	$007C-$007D	Basic-Pseudo-Stack-Pointer
126- 127	$007E-$007F	Arbeitsbereich (Sound)
128	$0080	Arbeitsbereich (Sound)
129	$0081	Flag für RUN-Modus ($80:=Ja; $00:=Nein)
130	$0082	Flag für DOS-Befehl, Zeiger bei PRINT USING
131	$0083	Grafikmodus: $00:=Text; $20:=HIRES $60:=HIRES und Text; $A0:=Multicolor $E0:=Multicolor und Text
132	$0084	Derzeitige ausgewählte Farbe
133	$0085	Multicolor-Farbe 1
134	$0086	Vordergrundfarbe
135	$0087	Maximale Anzahl der Spalten
136	$0088	Maximale Anzahl der Zeilen
137	$0089	Flag: PAINT links
138	$008A	Flag: PAINT rechts
139	$008B	Flag: PAINT Farbe der Umrandung
140- 141	$008C-$008D	Zeiger auf BIT-MAP Farbinformation
142- 143	$008E-$008F	Temporäre Speicher
144	$0090	Statuswort ST
145	$0091	Flag: Stop-Taste/RVS-Taste
146	$0092	Temporärer Speicher
147	$0093	LOAD/VERIFY-Flag: $00:=LOAD; $01:=VERIFY

148	$ØØ94	Flag: Serieller Bus - Ausgabe Zeichenpuffer
149	$ØØ95	Zeichenspeicher - Serieller Bus
150	$ØØ96	Temporärer Speicher für BASIN
151	$ØØ97	Anzahl der geöffneten Dateien
152	$ØØ98	Eingabegerät (Normalwert: Ø)
153	$ØØ99	Ausgabegerät (Normalwert: 3)
154	$ØØ9A	Flag: $8Ø=Direkt-, $ØØ=Programm-Modus
155- 156	$ØØ9B-$ØØ9C	Zeiger: Kassettenpuffer/Scrolling
157- 158	$ØØ9D-$ØØ9E	Zeiger auf Ende des Programms
159- 160	$ØØ9F-$ØØAØ	Temporärer Speicher
161- 162	$ØØA1-$ØØA2	Temporärer Speicher
163- 165	$ØØA3-$ØØA5	Echtzeit-Uhr (1/6Ø Sekunden)
166	$ØØA6	Register für seriellen Bus
167	$ØØA7	Register für Kassettenroutine
168	$ØØA8	Register für seriellen Bus
169	$ØØA9	Temporärer Farb-Vektor
170	$ØØAA	Register für Kassettenroutine
171	$ØØAB	Anzahl der Zeichen im Filenamen
172	$ØØAC	Aktuelle logische Dateinummer
173	$ØØAD	Aktuelle Sekundäradresse
174	$ØØAE	Aktuelle Geräteadresse
175- 176	$ØØAF-$ØØBØ	Zeiger auf Filenamen
177	$ØØB1	Von Fehlerroutine benutzt
178- 179	$ØØB2-$ØØB3	I/Ø Startadresse
180- 181	$ØØB4-$ØØB5	Basis-Ladeadresse
182- 183	$ØØB6-$ØØB7	Zeiger: Anfang Kassettenpuffer
184- 185	$ØØB8-$ØØB9	Adresse für Vektor
186- 187	$ØØBA-$ØØBB	Zeiger auf Zeichen im Kassettenpuffer
188- 189	$ØØBC-$ØØBD	Zeiger auf String für Primms
190- 191	$ØØBE-$ØØBF	Register für Long-Fetch-Routine
192- 193	$ØØCØ-$ØØC1	Register für Scrolling
194	$ØØC2	RVS-Flag (Bildschirm)
195	$ØØC3	Zeiger: Ende der Zeile (Input)
196- 197	$ØØC4-$ØØC5	Cursorposition (Input), Zeile/Spalte
198	$ØØC6	Tastaturabfrage ($4Ø=keine Taste)
199	$ØØC7	Flag: Eingabe Bildschirm/Tastatur
200- 201	$ØØC8-$ØØC9	Zeiger auf Bildschirmzeile
202	$ØØCA	Cursorposition in aktueller Bildschirmzeile
203	$ØØCB	Flag: Anführungszeichen-Modus ($ØØ:=Nein)
204	$ØØCC	Länge der aktuellen Bildschirmzeile
205	$ØØCD	Zeile, in der sich der Cursor befindet
206	$ØØCE	Letztes Zeichen (I/Ø)
207	$ØØCF	Anzahl der Zeichen (Insert-Modus)
208- 215	$ØØDØ-$ØØD7	Reserviert für Software
216- 232	$ØØD8-$ØØE8	Reserviert für Anwendungssoftware
233	$ØØE9	Arbeitsbereich
234- 235	$ØØEA-$ØØEB	Bildschirm-Editor (Farb-RAM)
236- 238	$ØØEC-$ØØEE	Arbeitsbereich (Bildschirm)
239	$ØØEF	Anzahl der Zeichen im Tastaturpuffer

240	$00F0	Flag: Bildschirm-Pause (CTRL-S/T)
241- 242	$00F1-$00F2	Register für Monitor
243	$00F3	Zeropage-Adresse für Monitor
244	$00F4	Zeropage-Adresse für Monitor
245	$00F5	Register für Prüfsumme (Kassettenoperationen)
246	$00F6	Blocklänge (Kassettenoperationen)
247	$00F7	Pass (Kassettenoperationen)
248	$00F8	Block-Typ (Kassettenoperationen)
249	$00F9	Flag für Write, Read und IEC-Bus
250	$00FA	Register für X bei STOP-Tasten-Test
251	$00FB	Aktuelle Bank-Konfiguration
252	$00FC	Zu sendendes Zeichen für X-On
253	$00FD	Zu sendendes Zeichen für X-Off
254	$00FE	Arbeitsregister (Editor)
255	$00FF	LOFBUF

3.2 Erweiterte Zeropage

Dezimal	Hexadezimal	Beschreibung
256- 271	$0100-$010F	FBUFFER
272	$0110	Register für Akku bei SAVE und RESTORE
273	$0111	Register für Y-Reg. bei SAVE und RESTORE
274	$0112	Register für X-Reg. bei SAVE und RESTORE
275- 289	$0113-$0122	Farb- und Helligkeits-Tabellen im RAM
291- 511	$0124-$01FF	Prozessorstack
512- 600	$0200-$0258	Eingabepuffer
601- 602	$0259-$025A	Vorherige Basic-Zeilennummer
603- 604	$0259-$025C	Zeiger für CONT
605- 684	$025D-$02AC	Basic-/DOS-Arbeitsbereich:
605	$025D	DOS-Schleifenzähler
606- 621	$025E-$026D	Bereich für Filenamen
622	$026E	1. Filename (Länge)
623	$026F	DOS (Laufwerk 1)
624- 625	$0270-$0271	1. Filename (Adresse)
626	$0272	2. Filename (Länge)
627	$0273	DOS (Laufwerk 2)
628- 629	$0274-$0275	2. Filename (Adresse)
630	$0276	DOS logische Adresse
631	$0277	DOS Geräteadresse
632	$0278	DOS Sekundäradresse
633- 634	$0279-$027A	DOS Disketten-ID
635	$027B	ID-Flag
636	$027C	DOS Ausgabepuffer
637- 684	$027D-$02AC	DOS Arbeitsbereich
685- 686	$02AD-$02AE	Grafik-Cursor (X-Position)
687- 688	$02AF-$02B0	Grafik-Cursor (Y-Position)

689- 690	$02B1-$02B2	Grafik-Cursor (X-Zielkoordinate)
691- 692	$02B3-$02B4	Grafik-Cursor (Y-Zielkoordinate)
693- 694	$02B5-$02B6	Absolutwert von X-Position
695- 696	$02B7-$02B8	Absolutwert von Y-Position
697- 698	$02B9-$02BA	Signumwert von X-Position
699- 700	$02BB-$02BC	Signumwert von Y-Position
701- 708	$02BD-$02C4	Grafik-Register
709	$02C5	Vorzeichen des Winkels
710- 711	$02C6-$02C7	Sinus des Winkels
712- 713	$02C8-$02C9	Cosinus des Winkels
714- 715	$02CA-$02CB	Register für Winkelabstand
717- 736	$02CD-$02E0	Adressen für PRINT-USING:
717	$02CD	Zeiger auf Beginn des Zahlenstrings
718	$02CE	Zeiger auf ENDE des Zahlenstrings
719	$02CF	Dollar-Flag
720	$02D0	Komma-Flag
721	$02D1	Zähler
722	$02D2	Vorzeichen Exponent
723	$02D3	Zeiger auf Exponent
724	$02D4	Anzahl der Vorkommastellen im Zahlenstring
725	$02D5	Justierungs-Flag
726	$02D6	Anzahl der Vorkommastellen im Formatstring
727	$02D7	Anzahl der Nachkommastellen im Formatstring
728	$02D8	Vorzeichen-Flag im Formatstring
729	$02D9	Exponent-Flag
730	$02DA	Schalter
731	$02DB	Zeichenzähler
732	$02DC	Vorzeichen-Flag im Zahlenstring
733	$02DD	Flag für */SPACE
734	$02DE	Zeiger auf Feldanfang
735	$02DF	Länge des Formatstrings
736	$02E0	Zeiger auf Feldende
740	$02E4	High-Byte-Adresse des Charakter-ROM (normalerweise: $d0)
741	$02E5	Register für GSHAPE
742	$02E6	SCALE-Flag ($00:=Aus)
743	$02E7	Flag: Doppelte Pixel-Größe
744	$02E8	Flag: Box/Frame
745	$02E9	Temporärer Speicher für Bit-Maske
746	$02EA	Stringlänge
747	$02EB	Trace-Flag ($00:=Aus)
748- 750	$02EC-$02EE	Zwischenspeicher für Directory
751	$02EF	Temporärer Speicher für Grafik
752	$02F0	Anzahl der Grafik-Parameter
753	$02F1	Parameter ($01:=Relativ, $00:=Absolut)
754- 755	$02F2-$02F3	Fließkomma-Vektor
756- 757	$02F4-$02F5	Integer-Vektor
766- 767	$02FE-$02FF	Vektor: Funktions-Modul

768- 769	$0300-$0301	Sprungvektor: Fehlermeldung
770- 771	$0302-$0303	Sprungvektor: BASIC-Warmstart
772- 773	$0304-$0305	Sprungvektor: Token-Generierung
774- 775	$0306-$0307	Sprungvektor: Keywort erzeugen
776- 777	$0308-$0309	Sprungvektor: Hauptschleife
778- 779	$030A-$030B	Sprungvektor: Eval
780- 781	$030C-$030D	Sprungvektor: Token-Generierung (User)
782- 783	$030E-$030F	Sprungvektor: Keywort erzeugen
784- 785	$0310-$0311	Sprungvektor: User-Token bearbeiten
786- 787	$0312-$0313	Sprungvektor: Interrupt (Uhr)
788- 789	$0314-$0315	Sprungvektor: Hardware-Interrupt
790- 791	$0316-$0317	Sprungvektor: Break-Interrupt
792- 793	$0318-$0319	Sprungvektor: Open
794- 795	$031A-$031B	Sprungvektor: Close
796- 797	$031C-$031D	Sprungvektor: Kanal für Eingabe öffnen
798- 799	$031E-$031F	Sprungvektor: Kanal für Ausgabe öffnen
800- 801	$0320-$0321	Sprungvektor: I/Ø zurücksetzen
802- 803	$0322-$0323	Sprungvektor: Input
804- 805	$0324-$0325	Sprungvektor: Ausgabe
806- 807	$0326-$0327	Sprungvektor: Abfrage der Stoptaste
808- 809	$0328-$0329	Sprungvektor: Getin-Routine
810- 811	$032A-$032B	Sprungvektor: Schließen aller Files
812- 813	$032C-$032D	Sprungvektor: Monitor Break
814- 815	$032E-$032F	Sprungvektor: Load/Verify-Routine
816- 817	$0330-$0331	Sprungvektor: Save-Routine
818-1010	$0332-$03F2	Kassettenpuffer
818	$0332	File-Typ ($00:=BASIC; $01:=MPG)
819- 820	$0333-$0334	Startadresse
821- 822	$0335-$0336	Endadresse
823- 838	$0337-$0346	Filename
1011-1012	$03F3-$03F4	Datenzähler (Write)
1013-1014	$03F5-$03F6	Datenzähler (Read)
1015-1078	$03F7-$0436	RS232-Input-Puffer (64 Byte)
1079-1100	$0437-$0454	Kassetten-Fehler-Stack (Low)
1101-1138	$0455-$0472	Kassetten-Fehler-Stack (High)
1139-1144	$0473-$0478	CHRGET-Routine
1145-1156	$0479-$0484	CHRGOT-Routine
1172-1185	$0494-$04A1	Ind. RAM-Zugriff auf beliebige Bank
1186-1188	$04A2-$04A4	Numerische BASIC-Konstante
1189-1199	$04A5-$04AF	Textpointer
1200-1210	$04B0-$04BA	Index, Index 1
1211-1221	$04BB-$04C5	Index 2
1222-1232	$04C6-$04D0	String 1
1233-1243	$04D1-$04DB	Lowtr
1244-1254	$04DC-$04E6	Facmo
1255-1258	$04E7-$04EA	PRINT USING-Parameter:
1255	$04E7	PRINT USING-Füllzeichen
1256	$04E8	Komma-Symbol
1257	$04E9	Punkt-Symbol
1258	$04EA	Dollar-Symbol
1259-1262	$04EB-$04EE	Temporärer Speicher für INSTR
1263	$04EF	Fehlercode

1264-1265	$04F0-$04F1	ERROR-Zeilennummer
1266-1267	$04F2-$04F3	ON-ERROR-Zeilennummer
1268	$04F4	Temporäres Register für TRAP
1269-1270	$04F5-$04F6	Zwischenspeicher für TRAP
1271	$04F7	Zeiger auf letzten Fehler
1272-1273	$04F8-$04F9	DO-Adresse
1274-1275	$04FA-$04FB	DO-Zeilennummer
1276-1277	$04FC-$04FD	Tonhöhe/Länge (Low)
1278-1279	$04FE-$04FF	Tonhöhe/Länge (High)
1280-1282	$0500-$0502	USR-Sprungbefehl
1281-1282	$0501-$0502	USR-Adresse
1283-1287	$0503-$0507	Startwert für RND
1288	$0508	Flag für Kalt- oder Warmstart
1289-1298	$0509-$0512	Tabelle der logischen Filenummern
1299-1308	$0513-$051C	Tabelle der Geräteadressen
1309-1318	$051D-$0526	Tabelle der Sekundäradressen
1319-1328	$0527-$0530	Tastaturpuffer
1329-1330	$0531-$0532	Zeiger: RAM-Anfang
1331-1332	$0533-$0534	Zeiger: RAM-Ende
1333	$0535	TIMEOUT-Flag (IEC-Bus)
1334	$0536	Flag: File-Ende erreicht ($01:=Ja)
1335	$0537	Anzahl der Zeichen im Puffer
1336	$0538	Anzahl der gültigen Zeichen im Puffer
1337	$0539	Zeiger: Kassettenpuffer
1338	$053A	Typ des Kassetten-Files
1339	$053B	Flag: Farbe, Helligkeit und Blinken
1340	$053C	Blink-Flag
1342	$053E	Zeiger auf Anfangs-Page (Video-RAM)
1343	$053F	Größe des Tastaturpuffers
1344	$0540	Tasten-Wiederholfunktion ($80=alle, $40=keine, $00=nur DEL-, CURSOR-, oder Leertaste)
1345-1346	$0541-$0542	Zähler für Repeat
1347	$0543	Flag: Shift, CTRL, C=
1348	$0544	Letztes Shift-Zeichen
1349-1350	$0545-$0546	Sprungvektor: Keylog
1351	$0547	Text-/Grafik-Flag
1352	$0548	Scroll-Flag
1353	$0549	Temporärer Speicher
1354	$054A	Temporärer Speicher
1355	$054B	Arbeitsbereich von Monitor
1356-1358	$054C-$054E	Bereich für Assembler
1359-1361	$054F-$0551	Temporäre Speicher für Assembler
1362-1367	$0552-$0557	CPU-Register:
1362	$0552	Programm-Counter (High)
1363	$0553	Programm-Counter (Low)
1364	$0554	Prozessor-Flags
1365	$0555	Akkumulator
1366	$0556	X-Register
1367	$0557	Y-Register
1368	$0558	Stack-Pointer

1369-1372	$Ø559-$Ø55C	Temporäre Speicher für Monitor
1373	$Ø55D	Zähler für Funktionstasten
1374	$Ø55E	Zeiger auf Funktionstasten-String
1375-1382	$Ø55F-$Ø566	Länge der Funktionstasten-Strings
1383-1510	$Ø567-$Ø5E6	Speicher für Funktionstasten-Strings
1511-1515	$Ø5E7-$Ø5EB	Register für IEC-Bus
1516-1571	$Ø5EC-$Ø6EB	Page 1 der Banking-Routinen
1516-1519	$Ø5EC-$Ø5EF	Adreßtabelle
1520-1521	$Ø5FØ-$Ø5F1	Long-Jump (Adresse)
1522	$Ø5F2	Long-Jump (Akkumulator)
1523	$Ø5F3	Long-Jump (X-Register)
1524	$Ø5F4	Long-Jump (Status-Register)
1525-1629	$Ø5F5-$Ø65D	RAM-Bereich für Bank-Switching
1630-1771	$Ø65E-$Ø6EB	RAM-Bereich für Benutzersoftware
1792-1967	$Ø6EC-$Ø7AF	BASIC-Pseudo-Stack
1968-1996	$Ø7BØ-$Ø7CC	Kassetten-Arbeitsbereich:
1968	$Ø7BØ	Datenbyte, das geschrieben werden soll
1969	$Ø7B1	Temporärer Speicher für Prüfsumme
1970-1971	$Ø7B2-$Ø7B3	Temporäre Speicher für HEADER
1973	$Ø7B5	Index für READ-BYTE-Routine
1974	$Ø7B6	Zeiger: ERROR-Stack
1975	$Ø7B7	Fehlerzähler Pass 1
1976-1977	$Ø7B8-$Ø7B9	Zeitkonstante
1978-1979	$Ø7BA-$Ø7BB	Zeitkonstante
1980-1981	$Ø7BC-$Ø7BD	Zeitkonstante
1982	$Ø7BE	Stack-Markierung für STOP-Taste
1983	$Ø7BF	Stack-Markierung für DROP-Taste
1984-1987	$Ø7CØ-$Ø7C3	Parameter nach Lesen eines Blocks
1988	$Ø7C4	Temporärer Speicher für RDBLOCK
1989	$Ø7C5	Anzahl der Shorts im HEADER
1990	$Ø7C6	Anzahl der Fehler bei Countdown
1991	$Ø7C7	Temporärer Speicher für VERIFY
1992-1995	$Ø7C8-$Ø7CB	Temporärer Speicher für T1
1996	$Ø7CC	Temporärer Speicher für Read-Error
1997-2000	$Ø7CD-$Ø7DØ	RS232-Arbeitsbereich
1997	$Ø7CD	Zu sendendes User-Zeichen
1998	$Ø7CE	Flag: $ØØ:=Leer; $Ø1:=Voll
1999	$Ø7CF	Zu sendendes System-Zeichen
2000	$Ø7DØ	Flag: $ØØ:=Leer; $Ø1:=Voll
2001	$Ø7D1	RS232: Pointer auf Anf. Eingabepuffer
2002	$Ø7D2	RS232: Pointer auf Ende Eingabepuffer
2003	$Ø7D3	Anzahl der Zeichen im Eingabepuffer
2004	$Ø7D4	Temporärer Status der ACIA
2005	$Ø7D5	Temporärer Speicher für INPUT
2006	$Ø7D6	Flag: Lokale Pause
2007	$Ø7D7	Flag: Ferngesteuerte Pause
2008	$Ø7D8	Flag: ACIA
2009-2020	$Ø7D9-$Ø7E4	Routine für indirekten RAM-Zugriff
2021	$Ø7E5	Bildschirm: Unterer Rand
2022	$Ø7E6	Bildschirm: Oberer Rand
2023	$Ø7E7	Bildschirm: Linker Rand
2024	$Ø7E8	Bildschirm: Rechter Rand

2026	$Ø7EA	Flag: Automatisches Einfügen ($FF:=Ein)
2027	$Ø7EB	Letztes ausgegebenes Zeichen (für ESC)
2028	$Ø7EC	Speicher für Bildschirmverwaltung
2029	$Ø7ED	Temporäres Register für Farbe bei INST/DEL
2030-2033	$Ø7EE-$Ø7F1	Zeilen-Link-Tabelle
2034	$Ø7F2	A-Register retten bei Sys-Kommando
2035	$Ø7F3	X-Register retten bei Sys-Kommando
2036	$Ø7F4	Y-Register retten bei Sys-Kommando
2037	$Ø7F5	Status-Register retten bei Sys-Kommando
2038	$Ø7F6	Register für Tastaturabfrage
2039	$Ø7F7	Flag: CTRL-S: ($ØØ:=offen, $Ø6:=gesperrt)
2040	$Ø7F8	RAM/ROM-Umschaltung für Monitor ($ØØ:=ROM; $8Ø:=RAM)
2041	$Ø7F9	RAM/ROM-Umschaltung für Farbe/Helligkeit ($ØØ:=ROM; $8Ø:=RAM)
2042	$Ø7FA	ROM-Maske für geteilten Bildschirm
2043	$Ø7FB	Video-RAM-Maske für geteilten Bildschirm
2044	$Ø7FC	Motorsteuerung für Kassettenrekorder
2045	$Ø7FD	Hilfszähler für Echtzeituhr bei PAL-System
2046-2047	$Ø7FE-$Ø7FF	Unbenutzt

3 System-Adressen

Ein sehr wichtiger Teil der Computer ist die Zeropage (Seite 0: $0000-$00FF). Für die ersten CBM-Rechner (PET, CBM 30xx und CBM 40xx) reichten die 256 Byte der Zeropage vollkommen aus, um alle wichtigen Adressen dort unterzubringen. Mit zunehmenden Fähigkeiten der Commodore-Nachfolger wuchs auch der Bedarf an System-Adressen. Die System-Adressen des C 16, C 116 und Plus/4 umfassen den Speicherbereich 0-2047 ($0000-$07FF). Der Bereich $0100-$07FF wird auch als »erweiterte Zeropage« bezeichnet. In der Zeropage selbst befinden sich die wichtigsten, am häufigsten benötigten Flags und Parameter. Der Prozessor verfügt über spezielle Zeropage-Befehle, die ein schnelleres Abarbeiten, bedingt durch kürzere Befehle und Zugriffszeiten, ermöglichen.

Die System-Adressen beinhalten alle wichtigen Parameter, Flags und Betriebssystem-Vektoren. Da die System-Adressen im RAM-Bereich liegen, kann durch Veränderung von Vektoren oder Parametern fast beliebig ins »Computergeschehen« eingegriffen werden. Die System-Adressen können nicht nur in Maschinensprache verändert werden. Auch von BASIC aus lassen sich diese Adressen mittels POKE-Befehl manipulieren.

Beim Experimentieren mit den System-Adressen ist jedoch Vorsicht geboten: Unkontrollierte Änderungen führen fast immer zu einem Totalabsturz des Computers. Da hilft dann nur noch die Reset-Taste.

Damit Sie sich besser in der Zeropage und der »erweiterten« Zeropage zurechtfinden, werden im folgenden die System-Adressen mit ausführlicher Beschreibung aufgezeigt.

3.1 Zeropage

Dezimal	Hexadezimal	Beschreibung
Ø	$ØØØØ	Datenrichtungsregister des 75Ø1
1	$ØØØ1	Ein-/Ausgabe-Port des 75Ø1
2	$ØØØ2	Flag für Schleifen
3- 4	$ØØØ3-$ØØØ4	Neue Startadresse (Renumber)
5- 6	$ØØØ5-$ØØØ6	Schrittweite (Renumber)
7	$ØØØ7	Gesuchtes Zeichen
8	$ØØØ8	Flag für Anführungszeichen-Modus, Zeilennr.
9	$ØØØ9	TAB-Spaltenzähler
1Ø	$ØØØA	LOAD/VERIFY-Flag: Ø = Load, 1 = Verify
11	$ØØØB	Zeiger für Eingabepuffer, Anzahl der Elemente
12	$ØØØC	Flag für Standard-DIM
13	$ØØØD	Datentyp: $FF=String, $ØØ=Numerisch
14	$ØØØE	Datentyp: $8Ø=Integer, $ØØ=Fließkomma
15	$ØØØF	Flag für DATA/LIST
16	$ØØ1Ø	Flag: Element/FNx-Flag
17	$ØØ11	Flag: $ØØ=INPUT, $4Ø=GET, $98=READ
18	$ØØ12	Flag: Vorzeichen des ATN
19	$ØØ13	Flag: Input-Prompt
2Ø- 21	$ØØ14-$ØØ15	2-Byte-Adresse, temporärer Speicher für RENUMBER
22	$ØØ16	Zeiger auf temporären Stringstapel
23- 24	$ØØ17-$ØØ18	Letzter temporärer Stringvektor
25- 33	$ØØ19-$ØØ21	Stapel für temporäre Strings
34- 35	$ØØ22-$ØØ23	Bereich für Hilfszeiger 1
36- 37	$ØØ24-$ØØ25	Bereich für Hilfszeiger 2
38	$ØØ26	Bereich für Produkt bei Multiplikation
39	$ØØ27	Bereich für Produkt bei Multiplikation
4Ø	$ØØ28	Bereich für Produkt bei Multiplikation
41	$ØØ29	Bereich für Produkt bei Multiplikation
42	$ØØ2A	Bereich für Produkt bei Multiplikation
43- 44	$ØØ2B-$ØØ2C	Zeiger auf BASIC-Anfang
45- 46	$ØØ2D-$ØØ2E	Zeiger auf Variablen-Anfang = BASIC-Programmende
47- 48	$ØØ2F-$ØØ3Ø	Zeiger auf Beginn der Arrays
49- 5Ø	$ØØ31-$ØØ32	Zeiger auf Ende der Arrays (+1)
51- 52	$ØØ33-$ØØ34	Zeiger auf Stringspeicher
53- 54	$ØØ35-$ØØ36	Hilfszeiger für Strings
55- 56	$ØØ37-$ØØ38	Zeiger auf oberste Speichergrenze
57- 58	$ØØ39-$ØØ3A	Laufende BASIC-Zeilennummer
59- 6Ø	$ØØ3B-$ØØ3C	Textpointer
61- 62	$ØØ3D-$ØØ3E	Zeiger auf BASIC-Statement für CONT
63- 64	$ØØ3F-$ØØ4Ø	Nummer der aktuellen DATA-Zeile
65- 66	$ØØ41-$ØØ42	Adresse des aktuellen DATA-Elementes
67- 68	$ØØ43-$ØØ44	Sprungvektor für INPUT
69- 7Ø	$ØØ45-$ØØ46	Aktueller Variablenname

71- 72	$0047-$0048	Adresse der aktuellen Variablen
73- 74	$0049-$004A	Variablenzeiger für FOR...NEXT
75- 76	$004B-$004C	Zwischenspeicher für Basic-Zeiger
77	$004D	Akkumulator für Vergleichssymbole
78- 79	$004E-$004F	Arbeitsbereich (Zeiger etc.)
80- 81	$0050-$0051	Arbeitsbereich (Zeiger etc.)
82	$0052	Arbeitsbereich (Zeiger etc.)
83	$0053	Grafik-Modus, HELP-Flag
84- 86	$0054-$0056	Sprungvektor für Funktionen
87- 96	$0057-$0060	Bereich für numerische Operationen
97	$0061	Fließkomma-Akkumulator 1 (FAC): Exponent
98- 101	$0062-$0065	Fließkomma-Akkumulator 1 (FAC): Mantisse
102	$0066	Fließkomma-Akkumulator 1 (FAC): Vorzeichen
103	$0067	Zeiger für Polynom-Auswertung
104	$0068	Fließkomma-Akkumulator 1 Überlauf
105	$0069	Fließkomma-Akkumulator 2 Exponent
106- 109	$006A-$006D	Fließkomma-Akkumulator 2 Mantisse
110	$006E	Fließkomma-Akkumulator 2 Vorzeichen
111	$006F	Vorzeichenvergleich Akku 1 mit Akku 2
112	$0070	Fließkomma-Akkumulator 1 niederwertige Stelle
113- 114	$0071-$0072	Kassettenpuffer Länge/Zeiger
115- 116	$0073-$0074	Automatisches Zeileninkrement (0=AUS)
117	$0075	Grafik-Flag (0=Text, 255=Grafik)
118- 120	$0076-$0078	Arbeitsbereich (MID$ etc.)
121- 123	$0079-$007B	Darstellung von DS$
124- 125	$007C-$007D	Basic-Pseudo-Stack-Pointer
126- 127	$007E-$007F	Arbeitsbereich (Sound)
128	$0080	Arbeitsbereich (Sound)
129	$0081	Flag für RUN-Modus ($80:=Ja; $00:=Nein)
130	$0082	Flag für DOS-Befehl, Zeiger bei PRINT USING
131	$0083	Grafikmodus: $00:=Text; $20:=HIRES $60:=HIRES und Text; $A0:=Multicolor $E0:=Multicolor und Text
132	$0084	Derzeitige ausgewählte Farbe
133	$0085	Multicolor-Farbe 1
134	$0086	Vordergrundfarbe
135	$0087	Maximale Anzahl der Spalten
136	$0088	Maximale Anzahl der Zeilen
137	$0089	Flag: PAINT links
138	$008A	Flag: PAINT rechts
139	$008B	Flag: PAINT Farbe der Umrandung
140- 141	$008C-$008D	Zeiger auf BIT-MAP Farbinformation
142- 143	$008E-$008F	Temporäre Speicher
144	$0090	Statuswort ST
145	$0091	Flag: Stop-Taste/RVS-Taste
146	$0092	Temporärer Speicher
147	$0093	LOAD/VERIFY-Flag: $00:=LOAD; $01:=VERIFY

148	$ØØ94	Flag: Serieller Bus - Ausgabe Zeichenpuffer
149	$ØØ95	Zeichenspeicher - Serieller Bus
150	$ØØ96	Temporärer Speicher für BASIN
151	$ØØ97	Anzahl der geöffneten Dateien
152	$ØØ98	Eingabegerät (Normalwert: Ø)
153	$ØØ99	Ausgabegerät (Normalwert: 3)
154	$ØØ9A	Flag: $8Ø=Direkt-, $ØØ=Programm-Modus
155- 156	$ØØ9B-$ØØ9C	Zeiger: Kassettenpuffer/Scrolling
157- 158	$ØØ9D-$ØØ9E	Zeiger auf Ende des Programms
159- 160	$ØØ9F-$ØØAØ	Temporärer Speicher
161- 162	$ØØA1-$ØØA2	Temporärer Speicher
163- 165	$ØØA3-$ØØA5	Echtzeit-Uhr (1/6Ø Sekunden)
166	$ØØA6	Register für seriellen Bus
167	$ØØA7	Register für Kassettenroutine
168	$ØØA8	Register für seriellen Bus
169	$ØØA9	Temporärer Farb-Vektor
170	$ØØAA	Register für Kassettenroutine
171	$ØØAB	Anzahl der Zeichen im Filenamen
172	$ØØAC	Aktuelle logische Dateinummer
173	$ØØAD	Aktuelle Sekundäradresse
174	$ØØAE	Aktuelle Geräteadresse
175- 176	$ØØAF-$ØØBØ	Zeiger auf Filenamen
177	$ØØB1	Von Fehlerroutine benutzt
178- 179	$ØØB2-$ØØB3	I/Ø Startadresse
180- 181	$ØØB4-$ØØB5	Basis-Ladeadresse
182- 183	$ØØB6-$ØØB7	Zeiger: Anfang Kassettenpuffer
184- 185	$ØØB8-$ØØB9	Adresse für Vektor
186- 187	$ØØBA-$ØØBB	Zeiger auf Zeichen im Kassettenpuffer
188- 189	$ØØBC-$ØØBD	Zeiger auf String für Primms
190- 191	$ØØBE-$ØØBF	Register für Long-Fetch-Routine
192- 193	$ØØCØ-$ØØC1	Register für Scrolling
194	$ØØC2	RVS-Flag (Bildschirm)
195	$ØØC3	Zeiger: Ende der Zeile (Input)
196- 197	$ØØC4-$ØØC5	Cursorposition (Input), Zeile/Spalte
198	$ØØC6	Tastaturabfrage ($4Ø=keine Taste)
199	$ØØC7	Flag: Eingabe Bildschirm/Tastatur
200- 201	$ØØC8-$ØØC9	Zeiger auf Bildschirmzeile
202	$ØØCA	Cursorposition in aktueller Bildschirmzeile
203	$ØØCB	Flag: Anführungszeichen-Modus ($ØØ:=Nein)
204	$ØØCC	Länge der aktuellen Bildschirmzeile
205	$ØØCD	Zeile, in der sich der Cursor befindet
206	$ØØCE	Letztes Zeichen (I/Ø)
207	$ØØCF	Anzahl der Zeichen (Insert-Modus)
208- 215	$ØØDØ-$ØØD7	Reserviert für Software
216- 232	$ØØD8-$ØØE8	Reserviert für Anwendungssoftware
233	$ØØE9	Arbeitsbereich
234- 235	$ØØEA-$ØØEB	Bildschirm-Editor (Farb-RAM)
236- 238	$ØØEC-$ØØEE	Arbeitsbereich (Bildschirm)
239	$ØØEF	Anzahl der Zeichen im Tastaturpuffer

240	$00F0	Flag: Bildschirm-Pause (CTRL-S/T)
241- 242	$00F1-$00F2	Register für Monitor
243	$00F3	Zeropage-Adresse für Monitor
244	$00F4	Zeropage-Adresse für Monitor
245	$00F5	Register für Prüfsumme (Kassettenoperationen)
246	$00F6	Blocklänge (Kassettenoperationen)
247	$00F7	Pass (Kassettenoperationen)
248	$00F8	Block-Typ (Kassettenoperationen)
249	$00F9	Flag für Write, Read und IEC-Bus
250	$00FA	Register für X bei STOP-Tasten-Test
251	$00FB	Aktuelle Bank-Konfiguration
252	$00FC	Zu sendendes Zeichen für X-On
253	$00FD	Zu sendendes Zeichen für X-Off
254	$00FE	Arbeitsregister (Editor)
255	$00FF	LOFBUF

3.2 Erweiterte Zeropage

Dezimal	Hexadezimal	Beschreibung
256- 271	$0100-$010F	FBUFFER
272	$0110	Register für Akku bei SAVE und RESTORE
273	$0111	Register für Y-Reg. bei SAVE und RESTORE
274	$0112	Register für X-Reg. bei SAVE und RESTORE
275- 289	$0113-$0122	Farb- und Helligkeits-Tabellen im RAM
291- 511	$0124-$01FF	Prozessorstack
512- 600	$0200-$0258	Eingabepuffer
601- 602	$0259-$025A	Vorherige Basic-Zeilennummer
603- 604	$0259-$025C	Zeiger für CONT
605- 684	$025D-$02AC	Basic-/DOS-Arbeitsbereich:
605	$025D	DOS-Schleifenzähler
606- 621	$025E-$026D	Bereich für Filenamen
622	$026E	1. Filename (Länge)
623	$026F	DOS (Laufwerk 1)
624- 625	$0270-$0271	1. Filename (Adresse)
626	$0272	2. Filename (Länge)
627	$0273	DOS (Laufwerk 2)
628- 629	$0274-$0275	2. Filename (Adresse)
630	$0276	DOS logische Adresse
631	$0277	DOS Geräteadresse
632	$0278	DOS Sekundäradresse
633- 634	$0279-$027A	DOS Disketten-ID
635	$027B	ID-Flag
636	$027C	DOS Ausgabepuffer
637- 684	$027D-$02AC	DOS Arbeitsbereich
685- 686	$02AD-$02AE	Grafik-Cursor (X-Position)
687- 688	$02AF-$02B0	Grafik-Cursor (Y-Position)

689- 690	$02B1-$02B2	Grafik-Cursor (X-Zielkoordinate)
691- 692	$02B3-$02B4	Grafik-Cursor (Y-Zielkoordinate)
693- 694	$02B5-$02B6	Absolutwert von X-Position
695- 696	$02B7-$02B8	Absolutwert von Y-Position
697- 698	$02B9-$02BA	Signumwert von X-Position
699- 700	$02BB-$02BC	Signumwert von Y-Position
701- 708	$02BD-$02C4	Grafik-Register
709	$02C5	Vorzeichen des Winkels
710- 711	$02C6-$02C7	Sinus des Winkels
712- 713	$02C8-$02C9	Cosinus des Winkels
714- 715	$02CA-$02CB	Register für Winkelabstand
717- 736	$02CD-$02E0	Adressen für PRINT-USING:
717	$02CD	Zeiger auf Beginn des Zahlenstrings
718	$02CE	Zeiger auf ENDE des Zahlenstrings
719	$02CF	Dollar-Flag
720	$02D0	Komma-Flag
721	$02D1	Zähler
722	$02D2	Vorzeichen Exponent
723	$02D3	Zeiger auf Exponent
724	$02D4	Anzahl der Vorkommastellen im Zahlenstring
725	$02D5	Justierungs-Flag
726	$02D6	Anzahl der Vorkommastellen im Formatstring
727	$02D7	Anzahl der Nachkommastellen im Formatstring
728	$02D8	Vorzeichen-Flag im Formatstring
729	$02D9	Exponent-Flag
730	$02DA	Schalter
731	$02DB	Zeichenzähler
732	$02DC	Vorzeichen-Flag im Zahlenstring
733	$02DD	Flag für */SPACE
734	$02DE	Zeiger auf Feldanfang
735	$02DF	Länge des Formatstrings
736	$02E0	Zeiger auf Feldende
740	$02E4	High-Byte-Adresse des Charakter-ROM (normalerweise: $d0)
741	$02E5	Register für GSHAPE
742	$02E6	SCALE-Flag ($00:=Aus)
743	$02E7	Flag: Doppelte Pixel-Größe
744	$02E8	Flag: Box/Frame
745	$02E9	Temporärer Speicher für Bit-Maske
746	$02EA	Stringlänge
747	$02EB	Trace-Flag ($00:=Aus)
748- 750	$02EC-$02EE	Zwischenspeicher für Directory
751	$02EF	Temporärer Speicher für Grafik
752	$02F0	Anzahl der Grafik-Parameter
753	$02F1	Parameter ($01:=Relativ, $00:=Absolut)
754- 755	$02F2-$02F3	Fließkomma-Vektor
756- 757	$02F4-$02F5	Integer-Vektor
766- 767	$02FE-$02FF	Vektor: Funktions-Modul

768- 769	$0300-$0301	Sprungvektor: Fehlermeldung
770- 771	$0302-$0303	Sprungvektor: BASIC-Warmstart
772- 773	$0304-$0305	Sprungvektor: Token-Generierung
774- 775	$0306-$0307	Sprungvektor: Keywort erzeugen
776- 777	$0308-$0309	Sprungvektor: Hauptschleife
778- 779	$030A-$030B	Sprungvektor: Eval
780- 781	$030C-$030D	Sprungvektor: Token-Generierung (User)
782- 783	$030E-$030F	Sprungvektor: Keywort erzeugen
784- 785	$0310-$0311	Sprungvektor: User-Token bearbeiten
786- 787	$0312-$0313	Sprungvektor: Interrupt (Uhr)
788- 789	$0314-$0315	Sprungvektor: Hardware-Interrupt
790- 791	$0316-$0317	Sprungvektor: Break-Interrupt
792- 793	$0318-$0319	Sprungvektor: Open
794- 795	$031A-$031B	Sprungvektor: Close
796- 797	$031C-$031D	Sprungvektor: Kanal für Eingabe öffnen
798- 799	$031E-$031F	Sprungvektor: Kanal für Ausgabe öffnen
800- 801	$0320-$0321	Sprungvektor: I/Ø zurücksetzen
802- 803	$0322-$0323	Sprungvektor: Input
804- 805	$0324-$0325	Sprungvektor: Ausgabe
806- 807	$0326-$0327	Sprungvektor: Abfrage der Stoptaste
808- 809	$0328-$0329	Sprungvektor: Getin-Routine
810- 811	$032A-$032B	Sprungvektor: Schließen aller Files
812- 813	$032C-$032D	Sprungvektor: Monitor Break
814- 815	$032E-$032F	Sprungvektor: Load/Verify-Routine
816- 817	$0330-$0331	Sprungvektor: Save-Routine
818-1010	$0332-$03F2	Kassettenpuffer
818	$0332	File-Typ ($00:=BASIC; $01:=MPG)
819- 820	$0333-$0334	Startadresse
821- 822	$0335-$0336	Endadresse
823- 838	$0337-$0346	Filename
1011-1012	$03F3-$03F4	Datenzähler (Write)
1013-1014	$03F5-$03F6	Datenzähler (Read)
1015-1078	$03F7-$0436	RS232-Input-Puffer (64 Byte)
1079-1100	$0437-$0454	Kassetten-Fehler-Stack (Low)
1101-1138	$0455-$0472	Kassetten-Fehler-Stack (High)
1139-1144	$0473-$0478	CHRGET-Routine
1145-1156	$0479-$0484	CHRGOT-Routine
1172-1185	$0494-$04A1	Ind. RAM-Zugriff auf beliebige Bank
1186-1188	$04A2-$04A4	Numerische BASIC-Konstante
1189-1199	$04A5-$04AF	Textpointer
1200-1210	$04B0-$04BA	Index, Index 1
1211-1221	$04BB-$04C5	Index 2
1222-1232	$04C6-$04D0	String 1
1233-1243	$04D1-$04DB	Lowtr
1244-1254	$04DC-$04E6	Facmo
1255-1258	$04E7-$04EA	PRINT USING-Parameter:
1255	$04E7	PRINT USING-Füllzeichen
1256	$04E8	Komma-Symbol
1257	$04E9	Punkt-Symbol
1258	$04EA	Dollar-Symbol
1259-1262	$04EB-$04EE	Temporärer Speicher für INSTR
1263	$04EF	Fehlercode

1264-1265	$04F0-$04F1	ERROR-Zeilennummer
1266-1267	$04F2-$04F3	ON-ERROR-Zeilennummer
1268	$04F4	Temporäres Register für TRAP
1269-1270	$04F5-$04F6	Zwischenspeicher für TRAP
1271	$04F7	Zeiger auf letzten Fehler
1272-1273	$04F8-$04F9	DO-Adresse
1274-1275	$04FA-$04FB	DO-Zeilennummer
1276-1277	$04FC-$04FD	Tonhöhe/Länge (Low)
1278-1279	$04FE-$04FF	Tonhöhe/Länge (High)
1280-1282	$0500-$0502	USR-Sprungbefehl
1281-1282	$0501-$0502	USR-Adresse
1283-1287	$0503-$0507	Startwert für RND
1288	$0508	Flag für Kalt- oder Warmstart
1289-1298	$0509-$0512	Tabelle der logischen Filenummern
1299-1308	$0513-$051C	Tabelle der Geräteadressen
1309-1318	$051D-$0526	Tabelle der Sekundäradressen
1319-1328	$0527-$0530	Tastaturpuffer
1329-1330	$0531-$0532	Zeiger: RAM-Anfang
1331-1332	$0533-$0534	Zeiger: RAM-Ende
1333	$0535	TIMEOUT-Flag (IEC-Bus)
1334	$0536	Flag: File-Ende erreicht ($01:=Ja)
1335	$0537	Anzahl der Zeichen im Puffer
1336	$0538	Anzahl der gültigen Zeichen im Puffer
1337	$0539	Zeiger: Kassettenpuffer
1338	$053A	Typ des Kassetten-Files
1339	$053B	Flag: Farbe, Helligkeit und Blinken
1340	$053C	Blink-Flag
1342	$053E	Zeiger auf Anfangs-Page (Video-RAM)
1343	$053F	Größe des Tastaturpuffers
1344	$0540	Tasten-Wiederholfunktion ($80=alle, $40=keine, $00=nur DEL-, CURSOR-, oder Leertaste)
1345-1346	$0541-$0542	Zähler für Repeat
1347	$0543	Flag: Shift, CTRL, C=
1348	$0544	Letztes Shift-Zeichen
1349-1350	$0545-$0546	Sprungvektor: Keylog
1351	$0547	Text-/Grafik-Flag
1352	$0548	Scroll-Flag
1353	$0549	Temporärer Speicher
1354	$054A	Temporärer Speicher
1355	$054B	Arbeitsbereich von Monitor
1356-1358	$054C-$054E	Bereich für Assembler
1359-1361	$054F-$0551	Temporäre Speicher für Assembler
1362-1367	$0552-$0557	CPU-Register:
1362	$0552	Programm-Counter (High)
1363	$0553	Programm-Counter (Low)
1364	$0554	Prozessor-Flags
1365	$0555	Akkumulator
1366	$0556	X-Register
1367	$0557	Y-Register
1368	$0558	Stack-Pointer

1369-1372	$Ø559-$Ø55C	Temporäre Speicher für Monitor
1373	$Ø55D	Zähler für Funktionstasten
1374	$Ø55E	Zeiger auf Funktionstasten-String
1375-1382	$Ø55F-$Ø566	Länge der Funktionstasten-Strings
1383-1510	$Ø567-$Ø5E6	Speicher für Funktionstasten-Strings
1511-1515	$Ø5E7-$Ø5EB	Register für IEC-Bus
1516-1571	$Ø5EC-$Ø6EB	Page 1 der Banking-Routinen
1516-1519	$Ø5EC-$Ø5EF	Adreßtabelle
1520-1521	$Ø5FØ-$Ø5F1	Long-Jump (Adresse)
1522	$Ø5F2	Long-Jump (Akkumulator)
1523	$Ø5F3	Long-Jump (X-Register)
1524	$Ø5F4	Long-Jump (Status-Register)
1525-1629	$Ø5F5-$Ø65D	RAM-Bereich für Bank-Switching
1630-1771	$Ø65E-$Ø6EB	RAM-Bereich für Benutzersoftware
1792-1967	$Ø6EC-$Ø7AF	BASIC-Pseudo-Stack
1968-1996	$Ø7BØ-$Ø7CC	Kassetten-Arbeitsbereich:
1968	$Ø7BØ	Datenbyte, das geschrieben werden soll
1969	$Ø7B1	Temporärer Speicher für Prüfsumme
1970-1971	$Ø7B2-$Ø7B3	Temporäre Speicher für HEADER
1973	$Ø7B5	Index für READ-BYTE-Routine
1974	$Ø7B6	Zeiger: ERROR-Stack
1975	$Ø7B7	Fehlerzähler Pass 1
1976-1977	$Ø7B8-$Ø7B9	Zeitkonstante
1978-1979	$Ø7BA-$Ø7BB	Zeitkonstante
1980-1981	$Ø7BC-$Ø7BD	Zeitkonstante
1982	$Ø7BE	Stack-Markierung für STOP-Taste
1983	$Ø7BF	Stack-Markierung für DROP-Taste
1984-1987	$Ø7CØ-$Ø7C3	Parameter nach Lesen eines Blocks
1988	$Ø7C4	Temporärer Speicher für RDBLOCK
1989	$Ø7C5	Anzahl der Shorts im HEADER
1990	$Ø7C6	Anzahl der Fehler bei Countdown
1991	$Ø7C7	Temporärer Speicher für VERIFY
1992-1995	$Ø7C8-$Ø7CB	Temporärer Speicher für T1
1996	$Ø7CC	Temporärer Speicher für Read-Error
1997-2000	$Ø7CD-$Ø7DØ	RS232-Arbeitsbereich
1997	$Ø7CD	Zu sendendes User-Zeichen
1998	$Ø7CE	Flag: $ØØ:=Leer; $Ø1:=Voll
1999	$Ø7CF	Zu sendendes System-Zeichen
2000	$Ø7DØ	Flag: $ØØ:=Leer; $Ø1:=Voll
2001	$Ø7D1	RS232: Pointer auf Anf. Eingabepuffer
2002	$Ø7D2	RS232: Pointer auf Ende Eingabepuffer
2003	$Ø7D3	Anzahl der Zeichen im Eingabepuffer
2004	$Ø7D4	Temporärer Status der ACIA
2005	$Ø7D5	Temporärer Speicher für INPUT
2006	$Ø7D6	Flag: Lokale Pause
2007	$Ø7D7	Flag: Ferngesteuerte Pause
2008	$Ø7D8	Flag: ACIA
2009-2020	$Ø7D9-$Ø7E4	Routine für indirekten RAM-Zugriff
2021	$Ø7E5	Bildschirm: Unterer Rand
2022	$Ø7E6	Bildschirm: Oberer Rand
2023	$Ø7E7	Bildschirm: Linker Rand
2024	$Ø7E8	Bildschirm: Rechter Rand

2026	$Ø7EA	Flag: Automatisches Einfügen ($FF:=Ein)
2027	$Ø7EB	Letztes ausgegebenes Zeichen (für ESC)
2028	$Ø7EC	Speicher für Bildschirmverwaltung
2029	$Ø7ED	Temporäres Register für Farbe bei INST/DEL
2030-2033	$Ø7EE-$Ø7F1	Zeilen-Link-Tabelle
2034	$Ø7F2	A-Register retten bei Sys-Kommando
2035	$Ø7F3	X-Register retten bei Sys-Kommando
2036	$Ø7F4	Y-Register retten bei Sys-Kommando
2037	$Ø7F5	Status-Register retten bei Sys-Kommando
2038	$Ø7F6	Register für Tastaturabfrage
2039	$Ø7F7	Flag: CTRL-S: ($ØØ:=offen, $Ø6:=gesperrt)
2040	$Ø7F8	RAM/ROM-Umschaltung für Monitor ($ØØ:=ROM; $8Ø:=RAM)
2041	$Ø7F9	RAM/ROM-Umschaltung für Farbe/Helligkeit ($ØØ:=ROM; $8Ø:=RAM)
2042	$Ø7FA	ROM-Maske für geteilten Bildschirm
2043	$Ø7FB	Video-RAM-Maske für geteilten Bildschirm
2044	$Ø7FC	Motorsteuerung für Kassettenrekorder
2045	$Ø7FD	Hilfszähler für Echtzeituhr bei PAL-System
2046-2047	$Ø7FE-$Ø7FF	Unbenutzt

4 ROM-Listing

```
. 8000  :
. 8000  *** BASIC START ***
. 8000  :
. 8000  4C 19 80 jmp $8019   Basic Kaltstart

. 8003  4C 0a 80 jmp $800a   Basic Warmstart

. 8006  00       .byte $00
. 8007  43 42 20 .byte 'cbm'
. 800a  :
. 800a  *** WARMSTART ***
. 800a  :
. 800a  20 cc ff jsr $ffcc   (CLRCH) Ein-/Ausgabe-Kanal schliessen
. 800d  20 d8 8a jsr $8ad8   Stack initialisieren
. 8010  85 13    sta $13     Eingabegerät gleich Tastatur
. 8012  20 c9 c7 jsr $c7c9   Grafik abschalten
. 8015  58       cli
. 8016  4c 7e 86 jmp $867e   'READY.' ausgeben

. 8019  :
. 8019  *** KALTSTART ***
. 8019  :
. 8019  20 17 81 jsr $8117   Basic-Vektoren setzen
. 801c  20 2e 80 jsr $802e   RAM initialisieren
. 801f  20 c2 80 jsr $80c2   Einschaltmeldung ausgeben
. 8022  20 f4 fc jsr $fcf4   Evtl. vorhandenes Modul initialisieren
. 8025  a2 fb    ldx #$fb    4 Byte am Ende des Stacks
. 8027  9a       txs         ... bleiben für Zeilenkopf frei
. 8028  d0 ec    bne $8016   'READY.' ausgeben
. 802a  :
. 802a  *** VEKTOREN ***
. 802a  :
. 802a  71 98    .adr $9871  Gleitkomma-Integer Wandlung
. 802c  71 94    .adr $9471  Integer-Gleitkomma Wandlung
. 802e  :
. 802e  *** RAM FÜR BASIC INITIALISIEREN ***
```

```
. 802e  :
. 802e  a9 4c     lda #$4c     JMP-Code
. 8030  85 54     sta $54      Für Funktionen
. 8032  8d 00 05  sta $0500    Für USR-Funktion
. 8035  a9 1c     lda #$1c     Zeiger auf 'illegal quantity'
. 8037  a0 99     ldy #$99
. 8039  8d 01 05  sta $0501    Als USR-Vektor speichern
. 803c  8c 02 05  sty $0502
. 803f  a2 03     ldx #$03     Vektoren für Gleitkomma-Integer
. 8041  bd 2a 80  lda $802a,x  ... und Integer-Gleitkomma
. 8044  9d f2 02  sta $02f2,x  ... Wandlung einrichten
. 8047  ca        dex
. 8048  10 f7     bpl $8041
. 804a  a2 32     ldx #$32     CHRGET-Routine und Routine
. 804c  bd 22 81  lda $8122,x  ... für LDA (...),Y ins RAM
. 804f  9d 72 04  sta $0472,x  ... kopieren
. 8052  ca        dex
. 8053  d0 f7     bne $804c
. 8055  86 68     stx $68      Vorzeichen Fließkomma Akkumulator 1
. 8057  86 13     stx $13      Eingabegerät gleich Tastatur
. 8059  86 18     stx $18      Zeiger auf letzten String im Stack
. 805b  8e eb 02  stx $02eb    TRACE ausschalten
. 805e  8e 00 10  stx $1000    $00 an Basic-Start
. 8061  8a        txa          Akku gleich 0
. 8062  a2 03     ldx #$03
. 8064  95 72     sta $72,x    AUTO-Flag und Grafik-Flag zurücksetzen
. 8066  9d e5 02  sta $02e5,x  Flags für Grafik zurücksetzen
. 8069  ca        dex
. 806a  d0 f8     bne $8064
. 806c  ea        nop
. 806d  8e 03 05  stx $0503    1. Stelle der Zufallszahl gleich Null
. 8070  e8        inx          X-Reg. gleich 1
. 8071  8e fd 01  stx $01fd    Link-Adresse für Basic-Zeilen
. 8074  8e fc 01  stx $01fc    ... in Stack schreiben
. 8077  ae 3b 05  ldx $053b    Farbe und Helligkeit
. 807a  86 86     stx $86      ... für Zeichen setzen
. 807c  a2 36     ldx #$36     Farbe und Helligkeit
. 807e  86 85     stx $85      ... für Multicolor-Grafik setzen
. 8080  a2 19     ldx #$19     Anfang der Strings
. 8082  86 16     stx $16      ... auf String-Stapelzeiger legen
. 8084  a2 01     ldx #$01     Adresse von Basic-Anfang ($1001)
. 8086  a0 10     ldy #$10
. 8088  86 2b     stx $2b      ... auf Basic-Anfangs-Zeiger legen
. 808a  84 2c     sty $2c
. 808c  a2 05     ldx #$05     Zähler für Kopierroutine
. 808e  86 22     stx $22
. 8090  a9 d0     lda #$d0     Adresse des Standard-Zeichensatzes
. 8092  8d e4 02  sta $02e4    ... auf $d000 legen.
. 8095  a2 02     ldx #$02
. 8097  bd 32 05  lda $0532,x  Holt Speicher-Obergrenze
. 809a  95 36     sta $36,x    Als Basic-Obergrenze speichern
. 809c  95 32     sta $32,x    Als String-Anfang speichern
```

```
. 809e  ca       dex
. 809f  d0 f6    bne $8097
. 80a1  a0 00    ldy #$00    Routine für indirekten RAM-Zugriff
. 80a3  b9 47 81 lda $8147,y ... wird von Adresse $8147
. 80a6  9d a5 04 sta $04a5,x ... ins RAM kopiert
. 80a9  e8       inx
. 80aa  c8       iny
. 80ab  c0 0b    cpy #$0b
. 80ad  90 f4    bcc $80a3
. 80af  a4 22    ldy $22
. 80b1  b9 bc 80 lda $80bc,y
. 80b4  9d 9f 04 sta $049f,x
. 80b7  c6 22    dec $22
. 80b9  10 e6    bpl $80a1
. 80bb  60       rts

. 80bc  64 5f 6f .by $64 $5f $6f  Adressen für
. 80bf  24 22 3b .by $24 $22 $3b  ... indirekten Speicherzugriff

. 80c2  :
. 80c2  *** EINSCHALTMELDUNG ***
. 80c2  :
. 80c2  a5 2b    lda $2b     Zeiger für
. 80c4  a4 2c    ldy $2c     ... Basic-Anfang
. 80c6  20 23 89 jsr $8923   Prüft auf Platz im Speicher
. 80c9  20 4f ff jsr $ff4f   Meldung ausgeben

. 80cc  93 0d 20 43 4f 4d 4d 4f   (CLR)(CR)' commo'  MELDUNG
. 80d4  44 4f 52 45 20 42 41 53   'dore bas'
. 80dc  49 43 20 56 33 2e 35 20   'ic v3.5 '
. 80e4  00       .byte $00

. 80e5  a5 37    lda $37     Basic-Obergrenze (Low)
. 80e7  38       sec
. 80e8  e5 2b    sbc $2b     ... -Programm-Anfangszeiger (Low)
. 80ea  aa       tax         ... gleich freier Speicher (Low)
. 80eb  a5 38    lda $38     Basic-Obergrenze (High-Byte)
. 80ed  e5 2c    sbc $2c     ... -Programm-Anfangszeiger (High)
. 80ef  20 5f a4 jsr $a45f   (X,A) als Dez-Zahl ausgeben
. 80f2  20 4f ff jsr $ff4f   Meldung ausgeben

. 80f5  20 42 59 54 45 53 20 46   ' bytes f'         MELDUNG
. 80fd  52 45 45 0d               'ree'(CR)
. 8101  00       .byte $00

. 8102  4c 7b 8a jmp $8a7b   NEW-Befehl

. 8105  :
. 8105  *** BASIC-VEKTOREN ***
. 8105  :
. 8105  86 86    .adr $8686  Fehlerausgabe
. 8107  12 87    .adr $8712  READY.-Schleife
```

```
. 8109  56 89     .adr $8956  Zeile tokenisieren
. 810b  6e 8b     .adr $8b6e  Token in Keywort wandeln
. 810d  d6 8b     .adr $8bd6  Basic-Routinenaufruf
. 810f  17 94     .adr $9417  Formelauswertung
. 8111  6a 89     .adr $896a  Keywort in Token 254 wandeln
. 8113  88 8b     .adr $8b88  Token 254 listen
. 8115  8b 8c     .adr $8c8b  Token 254 ausführen
. 8117  :
. 8117  *** BASIC-VEKTOREN ins RAM kopieren ***
. 8117  :
. 8117  a2 11     ldx #$11    Vektoren ins
. 8119  bd 05 81 lda $8105,x ... RAM kopieren
. 811c  9d 00 03 sta $0300,x
. 811f  ca        dex
. 8120  10 f7     bpl $8119
. 8122  60        rts

. 8123  :
. 8123  *** CHRGET-ROUTINE (KOPIE) ***
. 8123  :
. 8123  Die folgende Routine wird ab Adresse $0473 ins RAM kopiert
. 8123  :
. 8123  e6 3b     inc $3b
. 8125  d0 02     bne $8129
. 8127  e6 3c     inc $3c
. 8129  78        sei
. 812a  8d 3f ff sta $ff3f
. 812d  a0 00     ldy #$00
. 812f  b1 3b     lda ($3b),y
. 8131  8d 3e ff sta $ff3e
. 8134  58        cli
. 8135  c9 3a     cmp #$3a
. 8137  b0 0a     bcs $8143
. 8139  c9 20     cmp #$20
. 813b  f0 e6     beq $8123
. 813d  38        sec
. 813e  e9 30     sbc #$30
. 8140  38        sec
. 8141  e9 d0     sbc #$d0
. 8143  60        rts
. 8144  :
. 8144  *** ROUTINE FÜR INDIREKTEN RAM-ZUGRIFF ***
. 8144  :
. 8144  Die folgende Routine wird ab Adresse $0494 ins RAM kopiert
. 8144  :
. 8144  8d 9c 04 sta $049c
. 8147  78        sei
. 8148  8d 3f ff sta $ff3f
. 814b  b1 00     lda ($00),y
. 814d  8d 3e ff sta $ff3e
. 8150  58        cli
. 8151  60        rts
```

```
. 8152  ØØ ØØ ØØ .byte $ØØ $ØØ $ØØ  Konstante Ø für Systemvariable

. 8155  :
. 8155  *** ADRESSEN FÜR LDA (...),Y AUS RAM ***
. 8155  :
. 8155  a9 43     lda #$43      LDA (43),Y
. 8157  dØ 32     bne $818b

. 8159  a9 4e     lda #$4e      LDA (4e),Y
. 815b  dØ 2e     bne $818b

. 815d  a9 14     lda #$14      LDA (14),Y
. 815f  dØ 2a     bne $818b

. 8161  a9 47     lda #$47      LDA (47),Y
. 8163  dØ 26     bne $818b

. 8165  a9 4e     lda #$4e      LDA (4e),Y
. 8167  dØ 22     bne $818b

. 8169  a9 5c     lda #$5c      LDA (5c),Y
. 816b  dØ 1e     bne $818b

. 816d  a9 5f     lda #$5f      LDA (5f),Y
. 816f  dØ 1a     bne $818b

. 8171  a9 3d     lda #$3d      LDA (3d),Y
. 8173  dØ 16     bne $818b

. 8175  a9 57     lda #$57      LDA (57),Y
. 8177  dØ 12     bne $818b

. 8179  a9 59     lda #$59      LDA (59),Y
. 817b  dØ Øe     bne $818b

. 817d  a9 62     lda #$62      LDA (62),Y
. 817f  dØ Øa     bne $818b

. 8181  a9 5Ø     lda #$5Ø      LDA (5Ø),Y
. 8183  dØ Ø6     bne $818b

. 8185  a9 6c     lda #$6c      LDA (6c),Y
. 8187  dØ Ø2     bne $818b

. 8189  a9 5a     lda #$5a      LDA (5a),Y
. 818b  4c 94 Ø4  jmp $Ø494     Indirekter RAM-Zugriff

. 818e  :
. 818e  *** KEYWÖRTER ***
. 818e  :
. 818e  45 4e c4              .byte 'enD'     Token 128
```

```
. 8191  46 4f d2              .byte 'foR'     Token 129
. 8194  4e 45 58 d4           .byte 'nexT'    Token 130
. 8198  44 41 54 c1           .byte 'datA'    Token 131
. 819c  49 4e 50 55 54 a3     .byte 'input#'  Token 132
. 81a2  49 4e 50 55 d4        .byte 'inpuT'   Token 133
. 81a7  44 49 cd              .byte 'diM'     Token 134
. 81aa  52 45 41 c4           .byte 'reaD'    Token 135
. 81ae  4c 45 d4              .byte 'leT'     Token 136
. 81b1  47 4f 54 cf           .byte 'gotO'    Token 137
. 81b5  52 55 ce              .byte 'ruN'     Token 138
. 81b8  49 c6                 .byte 'iF'      Token 139
. 81ba  52 45 53 54 4f 52 c5  .byte 'restorE' Token 140
. 81c1  47 4f 53 55 c2        .byte 'gosuB'   Token 141
. 81c6  52 45 54 55 52 ce     .byte 'returN'  Token 142
. 81cc  52 45 cd              .byte 'reM'     Token 143
. 81cf  53 54 4f d0           .byte 'stoP'    Token 144
. 81d3  4f ce                 .byte 'oN'      Token 145
. 81d5  57 41 49 d4           .byte 'waiT'    Token 146
. 81d9  4c 4f 41 c4           .byte 'loaD'    Token 147
. 81dd  53 41 56 c5           .byte 'savE'    Token 148
. 81e1  56 45 52 49 46 d9     .byte 'verifY'  Token 149
. 81e7  44 45 c6              .byte 'deF'     Token 150
. 81ea  50 4f 4b c5           .byte 'pokE'    Token 151
. 81ee  50 52 49 4e 54 a3     .byte 'print#'  Token 152
. 81f4  50 52 49 4e d4        .byte 'prinT'   Token 153
. 81f9  43 4f 4e d4           .byte 'conT'    Token 154
. 81fd  4c 49 53 d4           .byte 'lisT'    Token 155
. 8201  43 4c d2              .byte 'clR'     Token 156
. 8204  43 4d c4              .byte 'cmD'     Token 157
. 8207  53 59 d3              .byte 'syS'     Token 158
. 820a  4f 50 45 ce           .byte 'opeN'    Token 159
. 820e  43 4c 4f 53 c5        .byte 'closE'   Token 160
. 8213  47 45 d4              .byte 'geT'     Token 161
. 8216  4e 45 d7              .byte 'neW'     Token 162
. 8219  54 41 42 a8           .byte 'tab('    Token 163
. 821d  54 cf                 .byte 'tO'      Token 164
. 821f  46 ce                 .byte 'fN'      Token 165
. 8221  53 50 43 a8           .byte 'spc('    Token 166
. 8225  54 48 45 ce           .byte 'theN'    Token 167
. 8229  4e 4f d4              .byte 'noT'     Token 168
. 822c  53 54 45 d0           .byte 'steP'    Token 169
. 8230  ab                    .byte '+'       Token 170
. 8231  ad                    .byte '-'       Token 171
. 8232  aa                    .byte '*'       Token 172
. 8233  af                    .byte '/'       Token 173
. 8234  de                    .byte ''        Token 174
. 8235  41 4e c4              .byte 'anD'     Token 175
. 8238  4f d2                 .byte 'oR'      Token 176
. 823a  be                    .byte '>'       Token 177
. 823b  bd                    .byte '='       Token 178
. 823c  bc                    .byte '<'       Token 179
. 823d  53 47 ce              .byte 'sgN'     Token 180
```

```
. 8240  49 4e d4              .byte 'inT'     Token 181
. 8243  41 42 d3              .byte 'abS'     Token 182
. 8246  55 53 d2              .byte 'usR'     Token 183
. 8249  46 52 c5              .byte 'frE'     Token 184
. 824c  50 4f d3              .byte 'poS'     Token 185
. 824f  53 51 d2              .byte 'sqR'     Token 186
. 8252  52 4e c4              .byte 'rnD'     Token 187
. 8255  4c 4f c7              .byte 'loG'     Token 188
. 8258  45 58 d0              .byte 'exP'     Token 189
. 825b  43 4f d3              .byte 'coS'     Token 190
. 825e  53 49 ce              .byte 'siN'     Token 191
. 8261  54 41 ce              .byte 'taN'     Token 192
. 8264  41 54 ce              .byte 'atN'     Token 193
. 8267  50 45 45 cb           .byte 'peeK'    Token 194
. 826b  4c 45 ce              .byte 'leN'     Token 195
. 826e  53 54 52 a4           .byte 'str$'    Token 196
. 8272  56 41 cc              .byte 'vaL'     Token 197
. 8275  41 53 c3              .byte 'asC'     Token 198
. 8278  43 48 52 a4           .byte 'chr$'    Token 199
. 827c  4c 45 46 54 a4        .byte 'left$'   Token 200
. 8281  52 49 47 48 54 a4     .byte 'right$'  Token 201
. 8287  4d 49 44 a4           .byte 'mid$'    Token 202
. 828b  47 cf                 .byte 'gO'      Token 203
. 828d  52 47 d2              .byte 'rgR'     Token 204
. 8290  52 43 4c d2           .byte 'rclR'    Token 205
. 8294  52 4c 55 cd           .byte 'rluM'    Token 206
. 8298  4a 4f d9              .byte 'joY'     Token 207
. 829b  52 44 4f d4           .byte 'rdoT'    Token 208
. 829f  44 45 c3              .byte 'deC'     Token 209
. 82a2  48 45 58 a4           .byte 'hex$'    Token 210
. 82a6  45 52 52 a4           .byte 'err$'    Token 211
. 82aa  49 4e 53 54 d2        .byte 'instR'   Token 212
. 82af  45 4c 53 c5           .byte 'elsE'    Token 213
. 82b3  52 45 53 55 4d c5     .byte 'resumE'  Token 214
. 82b9  54 52 41 d0           .byte 'traP'    Token 215
. 82bd  54 52 4f ce           .byte 'troN'    Token 216
. 82c1  54 52 4f 46 c6        .byte 'troff'   Token 217
. 82c6  53 4f 55 4e c4        .byte 'sounD'   Token 218
. 82cb  56 4f cc              .byte 'voL'     Token 219
. 82ce  41 55 54 cf           .byte 'autO'    Token 220
. 82d2  50 55 44 45 c6        .byte 'pudeF'   Token 221
. 82d7  47 52 41 50 48 49 c3  .byte 'graphiC' Token 222
. 82de  50 41 49 4e d4        .byte 'painT'   Token 223
. 82e3  43 48 41 d2           .byte 'chaR'    Token 224
. 82e7  42 4f d8              .byte 'boX'     Token 225
. 82ea  43 49 52 43 4c c5     .byte 'circlE'  Token 226
. 82f0  47 53 48 41 50 c5     .byte 'gshapE'  Token 227
. 82f6  53 53 48 41 50 c5     .byte 'sshapE'  Token 228
. 82fc  44 52 41 d7           .byte 'draW'    Token 229
. 8300  4c 4f 43 41 54 c5     .byte 'locatE'  Token 230
. 8306  43 4f 4c 4f d2        .byte 'coloR'   Token 231
. 830b  53 43 4e 43 4c d2     .byte 'scnclR'  Token 232
```

```
. 8311  53 43 41 4c c5        .byte 'scalE'   Token 233
. 8316  48 45 4c d0           .byte 'helP'    Token 234
. 831a  44 cf                 .byte 'dO'      Token 235
. 831c  4c 4f 4f d0           .byte 'looP'    Token 236
. 8320  45 58 49 d4           .byte 'exiT'    Token 237
. 8324  44 49 52 45 43 54 4f 52 d9 .byte 'directorY' Token 238
. 832d  44 53 41 56 c5        .byte 'dsavE'   Token 239
. 8332  44 4c 4f 41 c4        .byte 'dloaD'   Token 240
. 8337  48 45 41 44 45 d2     .byte 'headeR'  Token 241
. 833d  53 43 52 41 54 43 c8 .byte 'scratcH' Token 242
. 8344  43 4f 4c 4c 45 43 d4 .byte 'collecT' Token 243
. 834b  43 4f 50 d9           .byte 'copY'    Token 244
. 834f  52 45 4e 41 4d c5     .byte 'renamE'  Token 245
. 8355  42 41 43 4b 55 d0     .byte 'backuP'  Token 246
. 835b  44 45 4c 45 54 c5     .byte 'deletE'  Token 247
. 8361  52 45 4e 55 4d 42 45 d2    .byte 'renumbeR'  Token 248
. 8369  4b 45 d9              .byte 'keY'     Token 249
. 836c  4d 4f 4e 49 54 4f d2 .byte 'monitoR' Token 250
. 8373  55 53 49 4e c7        .byte 'usinG'   Token 251
. 8378  55 4e 54 49 cc        .byte 'untiL'   Token 252
. 837d  57 48 49 4c c5        .byte 'whilE'   Token 253
. 8382  00                    .byte 0
. 8383  :
. 8383  *** ADRESSEN DER BASIC-ROUTINEN ***
. 8383  :
. 8383  d9 8c     .adr $8cd9  Token 128   Befehl 'end'
. 8385  c9 ad     .adr $adc9  Token 129   Befehl 'for'
. 8387  93 92     .adr $9293  Token 130   Befehl 'next'
. 8389  af 8d     .adr $8daf  Token 131   Befehl 'data'
. 838b  ed 90     .adr $90ed  Token 132   Befehl 'input#'
. 838d  07 91     .adr $9107  Token 133   Befehl 'input'
. 838f  9a 96     .adr $969a  Token 134   Befehl 'dim'
. 8391  4e 91     .adr $914e  Token 135   Befehl 'read'
. 8393  7b 8e     .adr $8e7b  Token 136   Befehl 'let'
. 8395  4c 8d     .adr $8d4c  Token 137   Befehl 'goto'
. 8397  bb 8b     .adr $8bbb  Token 138   Befehl 'run'
. 8399  e0 8d     .adr $8de0  Token 139   Befehl 'if'
. 839b  99 8c     .adr $8c99  Token 140   Befehl 'restore'
. 839d  2b 8d     .adr $8d2b  Token 141   Befehl 'gosub'
. 839f  82 8d     .adr $8d82  Token 142   Befehl 'return'
. 83a1  0a 8e     .adr $8e0a  Token 143   Befehl 'rem'
. 83a3  d7 8c     .adr $8cd7  Token 144   Befehl 'stop'
. 83a5  1a 8e     .adr $8e1a  Token 145   Befehl 'on'
. 83a7  69 9e     .adr $9e69  Token 146   Befehl 'wait'
. 83a9  f2 a7     .adr $a7f2  Token 147   Befehl 'load'
. 83ab  dd a7     .adr $a7dd  Token 148   Befehl 'save'
. 83ad  ef a7     .adr $a7ef  Token 149   Befehl 'verify'
. 83af  9c 9a     .adr $9a9c  Token 150   Befehl 'def'
. 83b1  11 9e     .adr $9e11  Token 151   Befehl 'poke'
. 83b3  df 8f     .adr $8fdf  Token 152   Befehl 'print#'
. 83b5  ff 8f     .adr $8fff  Token 153   Befehl 'print'
. 83b7  02 8d     .adr $8d02  Token 154   Befehl 'cont'
```

```
. 83b9  fe 8a    .adr $8afe  Token 155   Befehl 'list'
. 83bb  97 8a    .adr $8a97  Token 156   Befehl 'clr'
. 83bd  e5 8f    .adr $8fe5  Token 157   Befehl 'cmd'
. 83bf  b4 a7    .adr $a7b4  Token 158   Befehl 'sys'
. 83c1  4c a8    .adr $a84c  Token 159   Befehl 'open'
. 83c3  59 a8    .adr $a859  Token 160   Befehl 'close'
. 83c5  b7 90    .adr $90b7  Token 161   Befehl 'get'
. 83c7  78 8a    .adr $8a78  Token 162   Befehl 'new'

. 83c9  0a 8e    .adr $8e0a  Token 213   Befehl 'else'
. 83cb  3f b4    .adr $b43f  Token 214   Befehl 'resume'
. 83cd  2a b4    .adr $b42a  Token 215   Befehl 'trap'
. 83cf  51 b6    .adr $b651  Token 216   Befehl 'tron'
. 83d1  54 b6    .adr $b654  Token 217   Befehl 'troff'
. 83d3  48 b8    .adr $b848  Token 218   Befehl 'sound'
. 83d5  bc b8    .adr $b8bc  Token 219   Befehl 'vol'
. 83d7  cc b6    .adr $b6cc  Token 220   Befehl 'auto'
. 83d9  43 b5    .adr $b543  Token 221   Befehl 'pudef'
. 83db  c2 c5    .adr $c5c2  Token 222   Befehl 'graphic'
. 83dd  d0 b8    .adr $b8d0  Token 223   Befehl 'paint'
. 83df  d3 b9    .adr $b9d3  Token 224   Befehl 'char'
. 83e1  e1 ba    .adr $bae1  Token 225   Befehl 'box'
. 83e3  1d c0    .adr $c01d  Token 226   Befehl 'circle'
. 83e5  34 bd    .adr $bd34  Token 227   Befehl 'gshape'
. 83e7  28 be    .adr $be28  Token 228   Befehl 'sshape'
. 83e9  d8 c4    .adr $c4d8  Token 229   Befehl 'draw'
. 83eb  0e c5    .adr $c50e  Token 230   Befehl 'locate'
. 83ed  19 c5    .adr $c519  Token 231   Befehl 'color'
. 83ef  66 c5    .adr $c566  Token 232   Befehl 'scnclr'
. 83f1  b7 c5    .adr $c5b7  Token 233   Befehl 'scale'
. 83f3  e7 b6    .adr $b6e7  Token 234   Befehl 'help'
. 83f5  56 b5    .adr $b556  Token 235   Befehl 'do'
. 83f7  02 b6    .adr $b602  Token 236   Befehl 'loop'
. 83f9  ab b5    .adr $b5ab  Token 237   Befehl 'exit'
. 83fb  bb c8    .adr $c8bb  Token 238   Befehl 'directory'
. 83fd  40 c9    .adr $c940  Token 239   Befehl 'dsave'
. 83ff  50 c9    .adr $c950  Token 240   Befehl 'dload'
. 8401  67 c9    .adr $c967  Token 241   Befehl 'header'
. 8403  9b c9    .adr $c99b  Token 242   Befehl 'scratch'
. 8405  cb c9    .adr $c9cb  Token 243   Befehl 'collect'
. 8407  d9 c9    .adr $c9d9  Token 244   Befehl 'copy'
. 8409  f3 c9    .adr $c9f3  Token 245   Befehl 'rename'
. 840b  ff c9    .adr $c9ff  Token 246   Befehl 'backup'
. 840d  59 ae    .adr $ae59  Token 247   Befehl 'delete'
. 840f  8e ab    .adr $ab8e  Token 248   Befehl 'renumber'
. 8411  28 b7    .adr $b728  Token 249   Befehl 'key'
. 8413  51 ff    .adr $ff51  Token 250   Befehl 'monitor'

. 8415  be a2    .adr $a2be  Token 180   Funktion 'sgn'
. 8417  58 a3    .adr $a358  Token 181   Funktion 'int'
. 8419  dd a2    .adr $a2dd  Token 182   Funktion 'abs'
. 841b  00 05    .adr $0500  Token 183   Funktion 'usr'
```

```
. 841d  62 9a     .adr $9a62  Token 184   Funktion 'fre'
. 841f  7d 9a     .adr $9a7d  Token 185   Funktion 'pos'
. 8421  e4 a5     .adr $a5e4  Token 186   Funktion 'sqr'
. 8423  Ø7 a7     .adr $a7Ø7  Token 187   Funktion 'rnd'
. 8425  1e aØ     .adr $aØ1e  Token 188   Funktion 'log'
. 8427  6Ø a6     .adr $a66Ø  Token 189   Funktion 'exp'
. 8429  7Ø aa     .adr $aa7Ø  Token 19Ø   Funktion 'cos'
. 842b  77 aa     .adr $aa77  Token 191   Funktion 'sin'
. 842d  cØ aa     .adr $aacØ  Token 192   Funktion 'tan'
. 842f  1a ab     .adr $ab1a  Token 193   Funktion 'atn'
. 8431  fa 9d     .adr $9dfa  Token 194   Funktion 'peek'
. 8433  61 9d     .adr $9d61  Token 195   Funktion 'len'
. 8435  66 9b     .adr $9b66  Token 196   Funktion 'str$'
. 8437  93 9d     .adr $9d93  Token 197   Funktion 'val'
. 8439  7Ø 9d     .adr $9d7Ø  Token 198   Funktion 'asc'
. 843b  bb 9c     .adr $9cbb  Token 199   Funktion 'chr$'
. 843d  cf 9c     .adr $9ccf  Token 2ØØ   Funktion 'left$'
. 843f  Ø3 9d     .adr $9dØ3  Token 2Ø1   Funktion 'right$'
. 8441  15 9d     .adr $9d15  Token 2Ø2   Funktion 'mid$'
. 8443  79 bf     .adr $bf79  Token 2Ø4   Funktion 'rgr'
. 8445  85 bf     .adr $bf85  Token 2Ø5   Funktion 'rclr'
. 8447  87 bf     .adr $bf87  Token 2Ø6   Funktion 'rlum'
. 8449  c1 bf     .adr $bfc1  Token 2Ø7   Funktion 'joy'
. 844b  fd bf     .adr $bffd  Token 2Ø8   Funktion 'rdot'
. 844d  1b 9e     .adr $9e1b  Token 2Ø9   Funktion 'dec'
. 844f  Ø7 b5     .adr $b5Ø7  Token 21Ø   Funktion 'hex$'
. 8451  be b4     .adr $b4be  Token 211   Funktion 'err$'

. 8453  79        .byte $79
. 8454  9d 9e     .adr $9e9d  Token 17Ø   Rech.-Operation '+'
. 8456  79        .byte $79
. 8457  86 9e     .adr $9e86  Token 171   Rech.-Operation '-'
. 8459  7b        .byte $7b
. 845a  7a aØ     .adr $aØ7a  Token 172   Rech.-Operation '*'
. 845c  7b        .byte $7b
. 845d  96 a1     .adr $a196  Token 173   Rech.-Operation '/'
. 845f  7f        .byte $7f
. 8460  ed a5     .adr $a5ed  Token 174   Rech.-Operation ''
. 8462  5Ø        .byte $5Ø
. 8463  fa 95     .adr $95fa  Token 175   Rech.-Operation 'and'
. 8465  46        .byte $46
. 8466  f7 95     .adr $95f7  Token 176   Rech.-Operation 'or'
. 8468  7d        .byte $7d
. 8469  26 a6     .adr $a626  Vorzeichenwechsel
. 846b  5a        .byte $5a
. 846c  64 94     .adr $9464  Token 168   Rech.-Operation 'not'
. 846e  64        .byte $64
. 846f  27 96     .adr $9627  Token 177-179 Vergleiche 'Größer/Gleich/Kleiner'
. 8471  :
. 8471  *** TABELLE DER FEHLER-MELDUNGEN ***
. 8471  :
. 8471  .byte 'too many fileS'          Meldung 1
```

```
. 847f  .byte 'file opeN'              Meldung 2
. 8488  .byte 'file not opeN'          Meldung 3
. 8495  .byte 'file not founD'         Meldung 4
. 84a3  .byte 'device not presenT'     Meldung 5
. 84b5  .byte 'not input filE'         Meldung 6
. 84c3  .byte 'not output filE'        Meldung 7
. 84d2  .byte 'missing file namE'      Meldung 8
. 84e3  .byte 'illegal device numbeR'  Meldung 9
. 84f8  .byte 'next without foR'       Meldung 10
. 8508  .byte 'syntaX'                 Meldung 11
. 850e  .byte 'return without gosuB'   Meldung 12
. 8522  .byte 'out of datA'            Meldung 13
. 852d  .byte 'illegal quantitY'       Meldung 14
. 853d  .byte 'overfloW'               Meldung 15
. 8545  .byte 'out of memorY'          Meldung 16
. 8552  .byte 'undef'd statemenT'      Meldung 17
. 8563  .byte 'bad subscripT'          Meldung 18
. 8570  .byte 'redim'd arraY'          Meldung 19
. 857d  .byte 'division by zerO'       Meldung 20
. 858d  .byte 'illegal direcT'         Meldung 21
. 859b  .byte 'type mismatcH'          Meldung 22
. 85a8  .byte 'string too lonG'        Meldung 23
. 85b7  .byte 'file datA'              Meldung 24
. 85c0  .byte 'formula too compleX'    Meldung 25
. 85d3  .byte 'can't continuE'         Meldung 26
. 85e1  .byte 'undef'd functioN'       Meldung 27
. 85f1  .byte 'verifY'                 Meldung 28
. 85f7  .byte 'loaD'                   Meldung 29
. 85fb  .byte 'break',0,' '            Meldung 30
. 8602  .byte 'can't resumE'           Meldung 31
. 860e  .byte 'loop not founD'         Meldung 32
. 861c  .byte 'loop without dO'        Meldung 33
. 862b  .byte 'direct mode onlY'       Meldung 34
. 863b  .byte 'no graphics areA'       Meldung 35
. 864b  .byte 'bad disK'               Meldung 36
. 8653  :
. 8653  *** ZEIGER AUF MELDUNG SETZEN ***
. 8653  :
. 8653  aa       tax          Fehlernummer ins X-Reg.
. 8654  a0 00    ldy #$00
. 8656  a9 71    lda #$71     Adresse der
. 8658  85 24    sta $24      ... Fehlertabelle ($8471)
. 865a  a9 84    lda #$84     ... nach $24/$25
. 865c  85 25    sta $25
. 865e  ca       dex          Richtige Fehlermeldung ?
. 865f  30 1c    bmi $867d    Ja, dann zurück
. 8661  b1 24    lda ($24),y  Nächste
. 8663  48       pha          ... Fehlermeldung
. 8664  e6 24    inc $24      ... überlesen
. 8666  d0 02    bne $866a
. 8668  e6 25    inc $25
. 866a  68       pla          Ende der Meldung ?
```

```
. 866b  10 f4     bpl $8661     Nein, dann weiter überlesen
. 866d  30 ef     bmi $865e     Ja, dann auf richtige Meldung prüfen

. 866f  20 4f ff  jsr $ff4f     Meldung ausgeben

. 8672  0d 0a               .byte (CR)(CD)
. 8674  52 45 41 44 59 2e  .byte 'ready.'
. 867a  0d 0a               .byte (CR)(CD)
. 867c  00        .byte $00

. 867d  60        rts           Zurück

. 867e  :
. 867e  *** FEHLERMELDUNG AUSGEBEN ***
. 867e  :
. 867e  a2 80     ldx #$80      'ready.'-Meldung ins X-Reg.
. 8680  44        .byte $44
. 8681  a2 10     ldx #$10      'out of memory'-Meldung ins X-Reg.
. 8683  6c 00 03  jmp ($0300)   Fehlermeldung ausgeben ($8686)
. 8686  8a        txa
. 8687  30 7a     bmi $8703     'READY.' ausgeben
. 8689  8e ef 04  stx $04ef     Fehlernummer für Fehlerbehandlung abspeichern
. 868c  24 81     bit $81       Programm-Modus ?
. 868e  10 35     bpl $86c5     Nein, dann Meldung ausgeben
. 8690  a0 01     ldy #$01      Adresse und
. 8692  b9 39 00  lda $0039,y   ... Zeilennummer
. 8695  99 f0 04  sta $04f0,y   ... für Fehlerbehandlung
. 8698  b9 5b 02  lda $025b,y   ... (TRAP, RESUME)
. 869b  99 f5 04  sta $04f5,y   ... zwischenspeichern
. 869e  88        dey
. 869f  10 f1     bpl $8692
. 86a1  e0 11     cpx #$11      Ist es Fehler 17 (UNDEF'D STATEMENT) ?
. 86a3  f0 20     beq $86c5     Ja, dann Meldung ausgeben
. 86a5  ac f3 04  ldy $04f3     Holt On-Error-Flag
. 86a8  c8        iny           On-Error-Flag gesetzt ?
. 86a9  f0 1a     beq $86c5     Nein, dann Meldung ausgeben
. 86ab  88        dey
. 86ac  84 15     sty $15       Zeilennummer
. 86ae  8c f4 04  sty $04f4     ... für Fehlerbehandlung
. 86b1  ac f2 04  ldy $04f2     ... während der
. 86b4  84 14     sty $14       ... Behandlung zwischenspeichern
. 86b6  a2 ff     ldx #$ff      On-Error-Flag
. 86b8  8e f3 04  stx $04f3     ... zurücksetzen
. 86bb  ae f7 04  ldx $04f7     Stack-Zeiger
. 86be  9a        txs           Für TRAP-Befehl
. 86bf  20 69 8d  jsr $8d69     Fehlerbehandlung
. 86c2  4c dc 8b  jmp $8bdc     Hauptschleife

. 86c5  :
. 86c5  *** MELDUNG AUSGEBEN ***
. 86c5  :
. 86c5  ca        dex
```

```
. 86c6  8a       txa
. 86c7  48       pha
. 86c8  a9 00    lda #$00      Grafik-Modus
. 86ca  85 83    sta $83       ... ausschalten
. 86cc  20 c9 c7 jsr $c7c9     Generiert normale Speicheraufteilung
. 86cf  68       pla
. 86d0  20 53 86 jsr $8653     Zeiger auf Fehlermeldung positionieren
. 86d3  20 cc ff jsr $ffcc     (CLRCH) Ein-/Ausgabe-Kanal schließen
. 86d6  a9 00    lda #$00      Eingabegerät gleich
. 86d8  85 13    sta $13       ... Tastatur
. 86da  20 3e 90 jsr $903e     Zeilenvorschub
. 86dd  20 b0 90 jsr $90b0     Fragezeichen (?) ausgeben
. 86e0  a0 00    ldy #$00      Fehlermeldung
. 86e2  b1 24    lda ($24),y   ... zeichenweise
. 86e4  48       pha           ... ausgeben
. 86e5  29 7f    and #$7f      ... dabei immer BIT 7 löschen
. 86e7  20 b2 90 jsr $90b2
. 86ea  c8       iny
. 86eb  68       pla
. 86ec  10 f4    bpl $86e2
. 86ee  20 d8 8a jsr $8ad8     Stack löschen
. 86f1  20 4f ff jsr $ff4f     Meldung ausgeben

. 86f4  20 45 52 52 4f 52      .byte ' error'       MELDUNG
. 86fa  00                     .byte $00

. 86fb  a4 3a    ldy $3a       Direktmodus ?
. 86fd  c8       iny
. 86fe  f0 03    beq $8703     Ja, dann $8703
. 8700  20 53 a4 jsr $a453     Ausgabe von 'IN' und Zeilennummer
. 8703  20 6f 86 jsr $866f     Direktmodus einschalten
. 8706  a9 80    lda #$80      Flag für Einschalten
. 8708  20 90 ff jsr $ff90     (SETMSG) Ausgabe einschalten
. 870b  a9 00    lda #$00      Flag für
. 870d  85 81    sta $81       Programm-Modus zurücksetzen
. 870f  :
. 870f  *** READY. ***
. 870f  :
. 870f  6c 02 03 jmp ($0302)   READY. ($8712)
. 8712  a2 ff    ldx #$ff      Direktmodus
. 8714  86 3a    stx $3a       ... einschalten
. 8716  20 5a 88 jsr $885a     Eingabeschleife
. 8719  86 3b    stx $3b       Textzeiger
. 871b  84 3c    sty $3c       ... setzen
. 871d  20 73 04 jsr $0473     CHRGET-Routine
. 8720  aa       tax
. 8721  f0 ec    beq $870f     Eingabe leer ?, dann $870f
. 8723  90 09    bcc $872e     Zeilennummer ? Ja, dann umwandeln und übergeben
. 8725  20 53 89 jsr $8953     Token generieren
. 8728  20 79 04 jsr $0479     CHRGOT-Routine
. 872b  4c d9 8b jmp $8bd9     Hauptschl.; Anweisung im Direktmodus ausführen
```

```
. 872e  20 3e 8e jsr $8e3e    Zeilennummer in Adr.-Format wandeln und übergeben
. 8731  20 53 89 jsr $8953    Token generieren
. 8734  84 0b    sty $0b      Zeilenlänge speichern
. 8736  20 3d 8a jsr $8a3d    Zeile suchen
. 8739  90 4a    bcc $8785    Noch nicht vorhanden, dann $8785
. 873b  :
. 873b  *** ZEILE LÖSCHEN ***
. 873b  :
. 873b  a0 01    ldy #$01
. 873d  20 d1 04 jsr $04d1    Holt Linkadresse (High)
. 8740  85 23    sta $23      In Hilfszeiger ablegen
. 8742  a5 2d    lda $2d      Holt Zeiger für Variablenanfang (Low)
. 8744  85 22    sta $22      In Hilfszeiger ablegen
. 8746  a5 60    lda $60      Holt Textpointer (Zeilenanfang) (High)
. 8748  85 25    sta $25      In Hilfszeiger ablegen
. 874a  88       dey
. 874b  20 d1 04 jsr $04d1    Holt Linkadresse (Low)
. 874e  18       clc          Subtrahiert
. 874f  e5 5f    sbc $5f      ... Zeilenanfang (Low)
. 8751  49 ff    eor #$ff     Invertieren
. 8753  18       clc          Gleich neg. Zeilenlänge
. 8754  65 2d    adc $2d      Addiert Variablenanfang (Low)
. 8756  85 2d    sta $2d      Gleich neuer Variablenanfang (Low)
. 8758  85 24    sta $24      In Hilfszeiger ablegen
. 875a  a5 2e    lda $2e      Holt Variablenanfang (High)
. 875c  69 ff    adc $ff      Addiert $ff
. 875e  85 2e    sta $2e      Speichert Variablenanfang (High)
. 8760  e5 60    sbc $60      Subtrahiert Zeilenanfang (High)
. 8762  aa       tax          Zahl der zu verschiebenden Seiten
. 8763  38       sec
. 8764  a5 5f    lda $5f      Holt Zeilenanfang (Low)
. 8766  e5 2d    sbc $2d      Subtrahiert Variablenanfang (Low)
. 8768  a8       tay          Gleich Restabschnitt
. 8769  b0 03    bcs $876e    Kleiner gleich $FF, dann $876e
. 876b  e8       inx          Seitenzahl um 1 erhöhen
. 876c  c6 25    dec $25      Hilfszeiger
. 876e  18       clc          ... für
. 876f  65 22    adc $22      ... Transport
. 8771  90 03    bcc $8776    ... vorbereiten
. 8773  c6 23    dec $23
. 8775  18       clc
. 8776  20 b0 04 jsr $04b0    Indirekter Speicherzugriff (LDA ($22),Y)
. 8779  91 24    sta ($24),y  An neue Adresse zurückschreiben
. 877b  c8       iny          Ende der Seite ?
. 877c  d0 f8    bne $8776    Nein, dann weitertransportieren
. 877e  e6 23    inc $23      Erhöht Hilfszeiger
. 8780  e6 25    inc $25      Erhöht Hilfszeiger
. 8782  ca       dex          Seitenzahl um 1 erniedrigen
. 8783  d0 f1    bne $8776    Noch nicht 0 ?, dann weitertransportieren
. 8785  20 9a 8a jsr $8a9a    CLR
. 8788  20 18 88 jsr $8818    Linkadresse erzeugen
. 878b  a0 00    ldy #$00     Holt 1. Zeichen
```

```
. 878d  20 a5 04 jsr $04a5     ... aus Puffer
. 8790  f0 8f    beq $8721     Ende ?, dann 'READY.' ausgeben

. 8792  :
. 8792  *** ZEILE EINFÜGEN ***
. 8792  :
. 8792  18       clc
. 8793  a5 2d    lda $2d       Holt Variablenanfang (Low)
. 8795  a4 2e    ldy $2e       Holt Variablenanfang (High)
. 8797  85 5a    sta $5a       Im Bereich für
. 8799  84 5b    sty $5b       ... numerische Operationen ablegen
. 879b  65 0b    adc $0b       Addiert
. 879d  90 01    bcc $87a0     ... Zeilenlänge
. 879f  c8       iny
. 87a0  18       clc
. 87a1  69 04    adc #$04      Addiert
. 87a3  90 01    bcc $87a6     ... Länge des
. 87a5  c8       iny           ... Zeilenkopfes (4 Byte)
. 87a6  85 58    sta $58       Im Bereich für
. 87a8  84 59    sty $59       ... numerische Operationen ablegen
. 87aa  20 c0 88 jsr $88c0     Verschiebe-Routine
. 87ad  a0 00    ldy #$00
. 87af  a9 01    lda #$01
. 87b1  91 5f    sta ($5f),y   Schreibt Linkadresse (Low)
. 87b3  c8       iny
. 87b4  91 5f    sta ($5f),y   Schreibt Linkadresse (High)
. 87b6  c8       iny
. 87b7  a5 14    lda $14
. 87b9  91 5f    sta ($5f),y   Schreibt Zeilennummer (Low)
. 87bb  a5 15    lda $15
. 87bd  c8       iny
. 87be  91 5f    sta ($5f),y   Schreibt Zeilennummer (High)
. 87c0  c8       iny
. 87c1  98       tya
. 87c2  18       clc           Zeiger
. 87c3  65 5f    adc $5f       ... um 4
. 87c5  85 5f    sta $5f       ... erhöhen
. 87c7  90 02    bcc $87cb     ... (Zeilenkopf
. 87c9  e6 60    inc $60       ... überspringen)
. 87cb  a5 31    lda $31       Ende
. 87cd  a4 32    ldy $32       ... der Arrays
. 87cf  85 2d    sta $2d       ... gleich Variablenanfang
. 87d1  84 2e    sty $2e       ... setzen
. 87d3  a4 0b    ldy $0b       Holt Zeilenlänge
. 87d5  88       dey
. 87d6  20 a5 04 jsr $04a5     Indirekter Speicherzugriff (LDA (3b),Y)
. 87d9  91 5f    sta ($5f),y   ... Zeile in
. 87db  88       dey           ... freigemachten
. 87dc  10 f8    bpl $87d6     ... Bereich kopieren
. 87de  20 18 88 jsr $8818     Linkadresse erzeugen
. 87e1  20 93 8a jsr $8a93     Textzeiger zurücksetzen, CLR
. 87e4  a5 73    lda $73       Holt AUTO-Schrittweite (Low)
```

```
. 87e6  05 74     ora $74       Logische ODER-Verknüpfung mit High-Byte
. 87e8  f0 2b     beq $8815     AUTO nicht aktiv ?, dann $8815
. 87ea  a5 14     lda $14       Holt Zeilennummer (Low)
. 87ec  18        clc           Addiert
. 87ed  65 73     adc $73       ... AUTO-Schrittweite (Low)
. 87ef  85 63     sta $63       Schreibt neue Zeilennummer in FAC 1
. 87f1  a5 15     lda $15       Holt Zeilennummer (High)
. 87f3  65 74     adc $74       Addiert AUTO-Schrittweite (High)
. 87f5  85 62     sta $62       Schreibt neue Zeilennummer in FAC 1
. 87f7  a2 90     ldx #$90      Exponent
. 87f9  38        sec           Keine Invertierung
. 87fa  20 ce a2  jsr $a2ce     Wandelt Integer in Gleitkomma-Zahl
. 87fd  20 6f a4  jsr $a46f     Erzeugt Zahlenstring
. 8800  a2 00     ldx #$00
. 8802  bd 01 01  lda $0101,x   Holt Zahlenstring vom Prozessorstack
. 8805  f0 06     beq $880d     Ende ?, dann $880d
. 8807  9d 27 05  sta $0527,x   Ziffer im Tastaturpuffer ablegen
. 880a  e8        inx
. 880b  d0 f5     bne $8802     Nächstes Zeichen holen
. 880d  a9 1d     lda #$1d      Holt Cursor-Rechts-Code
. 880f  9d 27 05  sta $0527,x   In Tastaturpuffer ablegen
. 8812  e8        inx
. 8813  86 ef     stx $ef       Setzt Anzahl der Zeichen in Pufferindex
. 8815  4c 0f 87  jmp $870f     READY.

. 8818  :
. 8818  *** LINKADRESSEN ERZEUGEN ***
. 8818  :
. 8818  a5 2b     lda $2b       Holt Programmanfang (Low)
. 881a  a4 2c     ldy $2c       Holt Programmanfang (High)
. 881c  85 22     sta $22       Programmanfang in
. 881e  84 23     sty $23       ... Hilfszeiger ablegen
. 8820  18        clc
. 8821  a0 00     ldy #$00
. 8823  20 b0 04  jsr $04b0     Holt Linkadresse (Low)
. 8826  d0 06     bne $882e     Adresse ungleich 0 ?, dann $882e
. 8828  c8        iny
. 8829  20 b0 04  jsr $04b0     Holt Linkadresse (High)
. 882c  f0 2b     beq $8859     Beide 0 ?, dann Ende
. 882e  a0 04     ldy #$04      Erste 4 Byte überspringen
. 8830  c8        iny
. 8831  20 b0 04  jsr $04b0     Holt Zeichen aus Zeile
. 8834  d0 fa     bne $8830     Zeilenende ?; Nein, dann weiter
. 8836  c8        iny           Linkadresse (Low)
. 8837  98        tya           ... neu
. 8838  65 22     adc $22       ... berechnen
. 883a  aa        tax           ... und
. 883b  a0 00     ldy #$00      ...
. 883d  91 22     sta ($22),y   ... zurückschreiben
. 883f  98        tya           Linkadresse (High)
. 8840  65 23     adc $23       ... neu berechnen
. 8842  c8        iny           ... und
```

```
. 8843  91 22     sta ($22),y  ... zurückschreiben
. 8845  86 22     stx $22      Neue Linkadresse
. 8847  85 23     sta $23      ... im Hilfszeiger ablegen
. 8849  90 d6     bcc $8821    Auf jeden Fall nach $8821

. 884b  18        clc
. 884c  a5 22     lda $22      Holt Programmende (Low) aus Hilfszeiger
. 884e  a4 23     ldy $23      Holt Programmende (High) aus Hilfszeiger
. 8850  69 02     adc #$02     Addiert 2 Byte
. 8852  90 01     bcc $8855
. 8854  c8        iny
. 8855  85 2d     sta $2d      Schreibt Variablenanfang (Low)
. 8857  84 2e     sty $2e      Schreibt Variablenanfang (High)
. 8859  60        rts          ENDE

. 885a  :
. 885a  *** EINGABESCHLEIFE ***
. 885a  :
. 885a  a2 00     ldx #$00     Eingabezeiger gleich 0
. 885c  20 91 a7  jsr $a791    Zeichen vom Bildschirm holen
. 885f  c9 0d     cmp #$0d     Zeilenende (RETURN) ?
. 8861  f0 0b     beq $886e    Ja, dann Eingabepuffer abschließen
. 8863  9d 00 02  sta $0200,x  Zeichen in Eingabepuffer schreiben
. 8866  e8        inx          Eingabezeiger erhöhen
. 8867  e0 59     cpx #$59     Schon 89 Zeichen eingegeben ?
. 8869  90 f1     bcc $885c    Nein, dann weiter Einlesen
. 886b  4c 4c cc  jmp $cc4c    'STRING TOO LONG ERROR' ausgeben

. 886e  4c 31 90  jmp $9031    Eingabepuffer abschließen

. 8871  :
. 8871  *** SUCHEN IM BASIC-STACK ***
. 8871  :
. 8871  20 60 a7  jsr $a760    Stackzeiger in Suchzeiger umkopieren
. 8874  a5 3d     lda $3d      Holt Suchzeiger (Low)
. 8876  c9 b0     cmp #$b0     Zeiger auf Stackanfang ?
. 8878  d0 06     bne $8880    Nein, dann $8880
. 887a  a5 3e     lda $3e      Holt Suchzeiger (High)
. 887c  c9 07     cmp #$07     Zeiger auf Stackanfang ?
. 887e  f0 3d     beq $88bd    Ja, dann Stack leer - Ende
. 8880  a0 00     ldy #$00
. 8882  a5 02     lda $02      Holt gesuchtes Token
. 8884  c9 81     cmp #$81     Token gleich 'FOR' ?
. 8886  d0 1b     bne $88a3    Nein, dann $88a3
. 8888  d1 3d     cmp ($3d),y  Token im Stack ?
. 888a  d0 33     bne $88bf    Nein, dann Ende
. 888c  a0 02     ldy #$02
. 888e  a5 4a     lda $4a      Holt FOR-NEXT-Variable (High)
. 8890  c9 ff     cmp #$ff     Kein $FF bei 'NEXT' angegeben ?
. 8892  f0 2b     beq $88bf    Nein, dann ENDE
. 8894  d1 3d     cmp ($3d),y  Variable im Stack vorhanden
. 8896  d0 07     bne $889f    Nein, dann $889f
```

```
. 8898  88       dey
. 8899  a5 49    lda $49       Holt FOR-NEXT-Variable (Low)
. 889b  d1 3d    cmp ($3d),y   Variable im Stack vorhanden
. 889d  f0 20    beq $88bf     Ja, dann Ende
. 889f  a2 12    ldx #$12      Stacklänge bei 'FOR'
. 88a1  d0 0e    bne $88b1

. 88a3  b1 3d    lda ($3d),y   Holt Token aus Stack
. 88a5  c5 02    cmp $02       Gleich gesuchtes Token ?
. 88a7  f0 16    beq $88bf     Ja, dann Ende
. 88a9  a2 12    ldx #$12      Stacklänge bei 'FOR'
. 88ab  c9 81    cmp #$81      Token gleich 'FOR' ?
. 88ad  f0 02    beq $88b1     Ja, dann $88b1
. 88af  a2 05    ldx #$05      Stacklänge bei 'GOSUB'
. 88b1  8a       txa           Stackzeiger
. 88b2  18       clc           ... um Stacklänge
. 88b3  65 3d    adc $3d       ... erhöhen
. 88b5  85 3d    sta $3d       ... und
. 88b7  90 bb    bcc $8874     ... nach
. 88b9  e6 3e    inc $3e       ... Adresse $8874
. 88bb  d0 b7    bne $8874     ... springen

. 88bd  a0 01    ldy #$01      Setzt Z-Bit zurück
. 88bf  60       rts           ENDE

. 88c0  :
. 88c0  *** BLOCKTRANSPORT ***
. 88c0  :
. 88c0  20 23 89 jsr $8923     Prüft auf Platz im Speicher
. 88c3  85 31    sta $31       Schreibt neues Variablenende (Low)
. 88c5  84 32    sty $32       Schreibt neues Variablenende (High)
. 88c7  38       sec
. 88c8  a5 5a    lda $5a       Holt Quellbereich-Ende (Low)
. 88ca  e5 5f    sbc $5f       Subtrahiert Quellbereich-Anfang (Low) + 1
. 88cc  85 22    sta $22       Schreibt Blocklänge (Low)
. 88ce  a8       tay           Y-Reg. gleich Blocklänge (Low)
. 88cf  a5 5b    lda $5b       Holt Quellbereich-Ende (High)
. 88d1  e5 60    sbc $60       Subtrahiert Quellbereich-Anfang (High) + 1
. 88d3  aa       tax           X-Reg. gleich Blocklänge (High) (volle Pages)
. 88d4  e8       inx
. 88d5  98       tya
. 88d6  f0 25    beq $88fd     Teilpage vorhanden ? Nein, dann $88fd
. 88d8  a5 5a    lda $5a       Holt Quellbereich-Ende (Low)
. 88da  38       sec
. 88db  e5 22    sbc $22       Subtrahiert Blocklänge (Low) + 1
. 88dd  85 5a    sta $5a       Schreibt Ende der letzten vollen Quellpage (Low)+1
. 88df  b0 03    bcs $88e4
. 88e1  c6 5b    dec $5b       Quelltext-Ende (High) um 1 erniedrigen
. 88e3  38       sec
. 88e4  a5 58    lda $58       Holt Zielbereich-Ende (Low)
. 88e6  e5 22    sbc $22       Subtrahiert Länge der Teilpage (Low)
. 88e8  85 58    sta $58       Gleich Ende letzte Zielpage (Low) + 1
```

```
. 88ea  bØ Ø9     bcs $88f5
. 88ec  c6 59     dec $59       Zielbereich-Ende (High) um 1 erniedrigen
. 88ee  9Ø Ø5     bcc $88f5

. 88fØ  2Ø 89 81  jsr $8189     Indirekter Speicherzugriff (LDA (Quellpage),Y)
. 88f3  91 58     sta ($58),y   Byte in Zielbereich schreiben
. 88f5  88        dey           Teilpage fertig ?
. 88f6  dØ f8     bne $88fØ     Nein, dann weiter verschieben
. 88f8  2Ø 89 81  jsr $8189     Indirekter Speicherzugriff (LDA (Quellpage),Y)
. 88fb  91 58     sta ($58),y   Letztes Byte der Teilpage schreiben
. 88fd  c6 5b     dec $5b       Quellzeiger (High) um 1 erniedrigen
. 88ff  c6 59     dec $59       Zielzeiger (High) um 1 erniedrigen
. 8901  ca        dex           Seitenzähler um 1 erniedrigen
. 8902  dØ f1     bne $88f5     Noch nicht Ø ?; dann weiter verschieben
. 8904  6Ø        Uts           ENDE

. 8905  :
. 8905  *** PLATZ IM BASIC-STACK PRÜFEN ***
. 8905  :
. 8905  8c f4 Ø7  sty $Ø7f4     Schreibt Platzbedarf in Byte
. 8908  38        sec
. 8909  a5 7c     lda $7c       Holt Basic-Stack-Pointer (Low)
. 890b  ed f4 Ø7  sbc $Ø7f4     Subtrahiert Platzbedarf + 1
. 890e  85 7c     sta $7c       Schreibt Basic-Stack-Pointer (Low) zurück
. 8910  a5 7d     lda $7d       Holt Basic-Stack-Pointer (High)
. 8912  e9 ØØ     sbc #$ØØ      Subtrahiert Übertrag
. 8914  85 7d     sta $7d       Schreibt Basic-Stack-Pointer (High) zurück
. 8916  c9 Ø6     cmp #$Ø6      Untergrenze
. 8918  9Ø 36     bcc $895Ø     ... des
. 891a  dØ Ø6     bne $8922     ... Basic-Stack
. 891c  a5 7c     lda $7c       ... unterschritten ?
. 891e  c9 ec     cmp #$ec      ... dann
. 8920  9Ø 2e     bcc $895Ø     ... 'OUT OF MEMORY ERROR'
. 8922  6Ø        rts           ENDE

. 8923  :
. 8923  *** PLATZ IM SPEICHER PRÜFEN ***
. 8923  :
. 8923  c4 34     cpy $34       Sind Akku
. 8925  9Ø 28     bcc $894f     ... und Y-Reg.
. 8927  dØ Ø4     bne $892d     ... kleiner
. 8929  c5 33     cmp $33       ... String-Anfangsadresse ?
. 892b  9Ø 22     bcc $894f     Ja, dann fertig
. 892d  48        pha           Akku auf Stack
. 892e  a2 Ø9     ldx #$Ø9
. 8930  98        tya           Y-Reg. auf Stack
. 8931  48        pha
. 8932  b5 57     lda $57,x     Adressen
. 8934  ca        dex           ... $57 bis $6Ø
. 8935  1Ø fa     bpl $8931     ... auf Stack
. 8937  2Ø 54 a9  jsr $a954     GARBAGE COLLECT
. 893a  a2 f7     ldx #$f7      Adressen
```

```
. 893c  68       pla          ... $57 bis $60
. 893d  95 61    sta $61,x    ... wieder
. 893f  e8       inx          ... vom Stack
. 8940  30 fa    bmi $893c    ... holen
. 8942  68       pla          Y-Reg. vom
. 8943  a8       tay          ... Stack holen
. 8944  68       pla          Akku vom Stack holen
. 8945  c4 34    cpy $34      Sind jetzt Akku
. 8947  90 06    bcc $894f    ... und Y-Reg.
. 8949  d0 05    bne $8950    ... kleiner
. 894b  c5 33    cmp $33      ... String-Anfangsadresse ?
. 894d  b0 01    bcs $8950    Nein, dann Fehler
. 894f  60       rts          ENDE

. 8950  4c 81 86 jmp $8681    'OUT OF MEMORY ERROR' ausgeben

. 8953  :
. 8953  *** TOKEN GENERIEREN ***
. 8953  :
. 8953  6c 04 03 jmp ($0304)  Token generieren ($8956)

. 8956  a5 3b    lda $3b      Textpointer
. 8958  48       pha          ... retten
. 8959  a5 3c    lda $3c
. 895b  48       pha
. 895c  20 79 04 jsr $0479    CHRGOT-Routine
. 895f  4c 65 89 jmp $8965    CHRGET-Routine überspringen

. 8962  20 73 04 jsr $0473    CHRGET-Routine
. 8965  90 fb    bcc $8962    Wenn Zeichen gleich Ziffer, dann weiterlesen

. 8967  :
. 8967  *** USER-TOKEN GENERIEREN ***
. 8967  :
. 8967  6c 0c 03 jmp ($030c)  User-Token generieren ($896a)

. 896a  90 68    bcc $89d4    User-Keywort ?, dann User-Zeichen auf Stack
. 896c  c9 00    cmp #$00     Endmarke ?
. 896e  f0 55    beq $89c5    Ja, dann Textpointer wiederherstellen und ENDE
. 8970  c9 3a    cmp #$3a     Doppelpunkt (als Trennzeichen) ?
. 8972  f0 ee    beq $8962    Ja, dann nächstes Zeichen holen
. 8974  c9 3f    cmp #$3f     Fragezeichen ?
. 8976  d0 04    bne $897c    Nein, dann $897c
. 8978  a9 99    lda #$99     Durch 'PRINT'-Token ersetzen
. 897a  d0 2e    bne $89aa    Springt immer nach $89aa

. 897c  c9 80    cmp #$80     SHIFT-Zeichen ?
. 897e  90 0b    bcc $898b    Nein, dann $898b
. 8980  c9 ff    cmp #$ff     PI-Zeichen ?
. 8982  f0 de    beq $8962    Ja, dann weiterlesen
. 8984  a0 01    ldy #$01     Y-Reg gleich 1
. 8986  20 ea 89 jsr $89ea    Zeilenrest um (Y) vorziehen
```

```
. 8989  f0 d1     beq $895c    Springt immer zum Anfang

. 898b  c9 22     cmp #$22     Anführungszeichen (") ?
. 898d  d0 0d     bne $899c    Nein, dann $899c
. 898f  20 73 04  jsr $0473    CHRGET-Routine
. 8992  c9 00     cmp #$00     Endmarke ?
. 8994  f0 2f     beq $89c5    Ja, dann Stackpointer wiederherstellen und ENDE
. 8996  c9 22     cmp #$22     Anführungszeichen (") ?
. 8998  f0 c8     beq $8962    Ja, dann weiter lesen
. 899a  d0 f3     bne $898f    Sonst nächstes Zeichen der Zeichenkette holen

. 899c  20 03 8a  jsr $8a03    Keywort in Tabelle suchen
. 899f  90 c1     bcc $8962    Nicht gefunden, dann weiterlesen
. 89a1  c0 00     cpy #$00     Keywort aus einem Zeichen ?
. 89a3  f0 03     beq $89a8    Ja, dann $89a8
. 89a5  20 ea 89  jsr $89ea    Zeilenrest um (Y) vorziehen
. 89a8  a5 0b     lda $0b      Holt Token
. 89aa  a0 00     ldy #$00
. 89ac  91 3b     sta ($3b),y  Speichert Token
. 89ae  c9 8f     cmp #$8f     'REM'-Token ?
. 89b0  f0 0d     beq $89bf    Ja, dann $89bf
. 89b2  c9 83     cmp #$83     'DATA'-Token ?
. 89b4  d0 ac     bne $8962    Nein, dann weiterlesen
. 89b6  20 73 04  jsr $0473    CHRGET-Routine
. 89b9  20 b0 8d  jsr $8db0    Stackpointer auf nächstes Trennzeichen setzen
. 89bc  4c 5c 89  jmp $895c    Zurück zum Anfang

. 89bf  20 73 04  jsr $0473    CHRGET-Routine
. 89c2  20 0b 8e  jsr $8e0b    Stackpointer auf Zeilenende
. 89c5  a6 3b     ldx $3b      Stackpointer
. 89c7  68        pla          ... wiederherstellen
. 89c8  85 3c     sta $3c
. 89ca  68        pla
. 89cb  85 3b     sta $3b
. 89cd  38        sec
. 89ce  8a        txa          Zeilenende
. 89cf  e5 3b     sbc $3b      Subtrahiert Zeilenanfang
. 89d1  a8        tay
. 89d2  c8        iny          Zeilenlänge
. 89d3  60        rts          ENDE

. 89d4  :
. 89d4  *** USER-TOKEN EINBAUEN ***
. 89d4  :
. 89d4  48        pha          User-Zeichen auf Stack
. 89d5  88        dey
. 89d6  88        dey
. 89d7  20 ea 89  jsr $89ea    Zeilenrest um (Y) vorziehen
. 89da  a0 00     ldy #$00
. 89dc  a9 fe     lda #$fe     User-Token ($fe)
. 89de  91 3b     sta ($3b),y  Token in Zeile schreiben
. 89e0  c8        iny
```

```
. 89e1  68       pla          User-Zeichen vom Stack holen
. 89e2  91 3b    sta ($3b),y  ... und in Zeile schreiben
. 89e4  20 73 04 jsr $0473    CHRGET-Routine
. 89e7  4c 62 89 jmp $8962    Weitermachen

. 8967  :
. 8967  *** ZEILENREST VORZIEHEN ***
. 8967  :
. 89ea  18       clc
. 89eb  98       tya          Offset aus Y-Reg.
. 89ec  65 3b    adc $3b      Wird zum Textzeiger (Low) addiert
. 89ee  85 22    sta $22      Ergebnis: Quellzeiger (Low)
. 89f0  a5 3c    lda $3c      Holt Textzeiger (High)
. 89f2  69 00    adc #$00     Addiert Übertrag
. 89f4  85 23    sta $23      Schreibt Quellzeiger (High)
. 89f6  a0 00    ldy #$00
. 89f8  20 b0 04 jsr $04b0    Holt Zeichen aus Zeile (LDA($22),Y)
. 89fb  91 3b    sta ($3b),y  Umkopieren
. 89fd  c8       iny
. 89fe  c9 00    cmp #$00     Zeilenende ?
. 8a00  d0 f6    bne $89f8    Nein, dann weiter umkopieren
. 8a02  60       rts          ENDE

. 8a03  :
. 8a03  *** KEYWORT SUCHEN ***
. 8a03  :
. 8a03  a9 81    lda #$81     Tabellenanfang ($818e)
. 8a05  a0 8e    ldy #$8e     ... nach Akku und Y-Reg retten
. 8a07  85 23    sta $23      Adresse in
. 8a09  84 22    sty $22      ... Hilfszeiger ablegen
. 8a0b  a0 00    ldy #$00
. 8a0d  84 0b    sty $0b      Wortzähler gleich Null
. 8a0f  88       dey
. 8a10  c8       iny
. 8a11  20 a5 04 jsr $04a5    Holt Zeichen aus Text (LDA($22),Y)
. 8a14  38       sec
. 8a15  f1 22    sbc ($22),y  Subtrahiert Zeichen aus Tabelle
. 8a17  f0 f7    beq $8a10    Zeichen gleich, dann Vergleich fortsetzen
. 8a19  c9 80    cmp #$80     Wortende ?
. 8a1b  f0 1b    beq $8a38    Ja, dann ENDE
. 8a1d  b1 22    lda ($22),y  Holt Zeichen aus Tabelle
. 8a1f  30 03    bmi $8a24    Endmarke ?, dann $8a24
. 8a21  c8       iny
. 8a22  d0 f9    bne $8a1d    Springt immer nach $8a1d

. 8a24  c8       iny
. 8a25  e6 0b    inc $0b      Wortzähler um 1 erhöhen
. 8a27  18       clc
. 8a28  98       tya
. 8a29  65 22    adc $22      Elemente addieren
. 8a2b  85 22    sta $22      Zeiger auf
. 8a2d  90 02    bcc $8a31    ... nächstes Wort
```

```
. 8a2f  e6 23     inc $23       ... setzen
. 8a31  18        clc
. 8a32  a0 00     ldy #$00
. 8a34  b1 22     lda ($22),y   Holt Zeichen aus Tabelle
. 8a36  d0 d9     bne $8a11     Tabelle noch nicht zu Ende, dann vergleichen
. 8a38  05 0b     ora $0b       Wenn Wort gefunden, dann Bit 7 setzen
. 8a3a  85 0b     sta $0b       Ergibt Token
. 8a3c  60        rts           ENDE

. 8a3d  :
. 8a3d  *** ZEILE SUCHEN ***
. 8a3d  :
. 8a3d  a5 2b     lda $2b       Holt Programmanfang (Low)
. 8a3f  a6 2c     ldx $2c       Holt Programmanfang (High)
. 8a41  a0 01     ldy #$01
. 8a43  85 5f     sta $5f       Schreibt Suchzeiger (Low)
. 8a45  86 60     stx $60       Schreibt Suchzeiger (High)
. 8a47  20 d1 04  jsr $04d1     Holt Linkadresse (High) (LDA($5f),Y)
. 8a4a  f0 2b     beq $8a77     Programmende ?, dann Zeile nicht gefunden
. 8a4c  c8        iny
. 8a4d  c8        iny
. 8a4e  20 d1 04  jsr $04d1     Holt Zeilennummer (High) (LDA($5f),Y)
. 8a51  85 78     sta $78       Im Arbeitsbereich ablegen
. 8a53  a5 15     lda $15       Holt gesuchte Zeilennummer (High)
. 8a55  c5 78     cmp $78       Vergleich mit gelesener Zeilennummer (High)
. 8a57  90 1f     bcc $8a78     Gesuchte Zeile kleiner?, dann ENDE
. 8a59  f0 03     beq $8a5e     Gesuchte Zeile gleich?, dann $8a5e
. 8a5b  88        dey           Y-Reg. gleich 2
. 8a5c  d0 0e     bne $8a6c     Springt immer nach $8a6c

. 8a5e  88        dey
. 8a5f  20 d1 04  jsr $04d1     Holt Zeilennummer (Low) (LDA($5f),Y)
. 8a62  85 78     sta $78       Im Arbeitsbereich ablegen
. 8a64  a5 14     lda $14       Holt gesuchte Zeilennummer (Low)
. 8a66  c5 78     cmp $78       Vergleich mit gelesener Zeilennummer (Low)
. 8a68  90 0e     bcc $8a78     Gesuchte Zeile kleiner?, dann ENDE
. 8a6a  f0 0c     beq $8a78     Gesuchte Zeile gleich?, dann ENDE
. 8a6c  88        dey
. 8a6d  20 d1 04  jsr $04d1     Holt zugeh. Linkadr. (High) (LDA($5f),Y)
. 8a70  aa        tax           ... zum Vorrücken des Suchzeigers
. 8a71  88        dey
. 8a72  20 d1 04  jsr $04d1     Holt zugeh. Linkadr. (Low) (LDA($5f),Y)
. 8a75  b0 ca     bcs $8a41     Springt immer nach $8a41

. 8a77  18        clc           Flag für Zeile nicht gefunden
. 8a78  60        rts           ENDE

. 8a79  :
. 8a79  *** N E W ***
. 8a79  :
. 8a79  d0 fd     bne $8a78     Folgt Trennzeichen ?, Wenn Nein, dann Fehler
. 8a7b  a9 00     lda #$00      Am Programmanfang
```

```
. 8a7d  a8       tay           ... die Endmarke
. 8a7e  91 2b    sta ($2b),y   ... $ØØ
. 8aØØ  c8       iny
. 8a81  91 2b    sta ($2b),y   ... anbringen
. 8a83  8d eb Ø2 sta $Ø2eb     Schaltet 'TRACE' aus
. 8a86  a5 2b    lda $2b       Programmanfang
. 8a88  18       clc           ... laden
. 8a89  69 Ø2    adc #$Ø2      ... 2 addieren
. 8a8b  85 2d    sta $2d       ... und
. 8a8d  a5 2c    lda $2c       ... als
. 8a8f  69 ØØ    adc #$ØØ      ... Variablenanfang
. 8a91  85 2e    sta $2e       ... zurückschreiben
. 8a93  2Ø f1 8a jsr $8af1     Textpointer zurücksetzen
. 8a96  a9 ØØ    lda #$ØØ      Verhindert Verzweigung
. 8a98  :
. 8a98  *** C L R ***
. 8a98  :
. 8a98  dØ 52    bne $8aec     Folgt Trennzeichen ?, Wenn Nein, dann Fehler
. 8a9a  2Ø e7 ff jsr $ffe7     (CLALL) schließt alle Kanäle
. 8a9d  aØ ØØ    ldy #$ØØ
. 8a9f  84 79    sty $79       Löscht DS$
. 8aa1  88       dey
. 8aa2  8c f3 Ø4 sty $Ø4f3     On-Error-Flag initialisieren
. 8aa5  8c fØ Ø4 sty $Ø4fØ     'TRAP'-Fehlerzeile (Low) initialisieren
. 8aa8  8c f1 Ø4 sty $Ø4f1     'TRAP'-Fehlerzeile (High) initialisieren
. 8aab  8c ef Ø4 sty $Ø4ef     Fehlercode initialisieren
. 8aae  a5 37    lda $37       Speichergrenze
. 8abØ  a4 38    ldy $38       ... auf
. 8ab2  85 33    sta $33       ... String-Anfang
. 8ab4  84 34    sty $34       ... setzen
. 8ab6  a9 bØ    lda #$bØ      Setzt
. 8ab8  aØ Ø7    ldy #$Ø7      ... Basic-Pseudo-Stack
. 8aba  85 7c    sta $7c       ... auf
. 8abc  84 7d    sty $7d       ... Adresse $Ø7bØ
. 8abe  a5 2d    lda $2d       Setzt
. 8acØ  a4 2e    ldy $2e       ... Variablen-Anfang
. 8ac2  85 2f    sta $2f       ... auf
. 8ac4  84 3Ø    sty $3Ø       ... Anfang der Arrays
. 8ac6  85 31    sta $31       ... und
. 8ac8  84 32    sty $32       ... Ende der Arrays
. 8aca  a2 Ø3    ldx #$Ø3      4 PRINT USING-Parameter
. 8acc  bd ed 8a lda $8aed,x   ... von $8aed
. 8acf  9d e7 Ø4 sta $Ø4e7,x   ... nach $Ø4e7
. 8ad2  ca       dex
. 8ad3  1Ø f7    bpl $8acc     ... kopieren
. 8ad5  2Ø b1 8c jsr $8cb1     RESTORE ausführen
. 8ad8  a2 19    ldx #$19      String-Stapel-Zeiger
. 8ada  86 16    stx $16       ... initialisieren
. 8adc  68       pla           Letzte Rücksprungadresse
. 8add  a8       tay           ... vom Stack
. 8ade  68       pla           ... nehmen
. 8adf  a2 fa    ldx #$fa      Stack
```

```
. 8ae1  9a       txs          ... initialisieren
. 8ae2  48       pha          Letzte Rücksprungadresse
. 8ae3  98       tya          ... wieder auf
. 8ae4  48       pha          ... Stack legen
. 8ae5  a9 00    lda #$00     Logische Adresse für Ausgabe
. 8ae7  8d 5c 02 sta $025c    ... und Integer-Sperre
. 8aea  85 10    sta $10      ... zurücksetzen
. 8aec  60       rts          ENDE

. 8aed  20 2c 2e 24  .byte ' ,.$'   Zeichen für 'PUDEF'

. 8af1  :
. 8af1  *** TEXTPOINTER INITIALISIEREN ***
. 8af1  :
. 8af1  18       clc          Holt
. 8af2  a5 2b    lda $2b      ... Programmanfang (Low, High)
. 8af4  69 ff    adc #$ff     ... subtrahiert 1
. 8af6  85 3b    sta $3b      ... und schreibt
. 8af8  a5 2c    lda $2c      ... als
. 8afa  69 ff    adc #$ff     ... Textpointer
. 8afc  85 3c    sta $3c      ... zurück
. 8afe  60       rts          ENDE

. 8aff  :
. 8aff  *** L I S T ***
. 8aff  :
. 8aff  20 ca ae jsr $aeca    Zeilenbereich analysieren
. 8b02  a0 01    ldy #$01
. 8b04  20 d1 04 jsr $04d1    Holt Linkadresse (High) (LDA($5f),Y)
. 8b07  d0 06    bne $8b0f    Ungleich 0 ?, dann $8b0f
. 8b09  88       dey
. 8b0a  20 d1 04 jsr $04d1    Holt Linkadresse (Low) (LDA($5f),Y)
. 8b0d  f0 2e    beq $8b3d    Beide 0 ?, dann Programmende erreicht
. 8b0f  20 c0 8c jsr $8cc0    Stoptaste prüfen
. 8b12  20 3e 90 jsr $903e    Neue Zeile ausgeben
. 8b15  a0 02    ldy #$02
. 8b17  20 d1 04 jsr $04d1    Holt Zeilennummer (Low) (LDA($5f),Y)
. 8b1a  aa       tax          Zeilennummer (Low) ins X-Reg.
. 8b1b  c8       iny
. 8b1c  20 d1 04 jsr $04d1    Holt Zeilennummer (High) (LDA($5f),Y)
. 8b1f  c5 15    cmp $15      Vergleicht Zeilennummer
. 8b21  d0 04    bne $8b27    ... mit Ende
. 8b23  e4 14    cpx $14      ... des LIST-Bereichs
. 8b25  f0 02    beq $8b29    Endzeile erreicht ?, dann $8b29
. 8b27  b0 14    bcs $8b3d    Endzeile überschritten ?, dann $8b3d
. 8b29  20 40 8b jsr $8b40    Zeile listen
. 8b2c  a0 00    ldy #$00
. 8b2e  20 d1 04 jsr $04d1    Holt Linkadresse (Low) (LDA($5f),Y)
. 8b31  aa       tax          Schreibt Linkadresse (Low) ins X-Reg.
. 8b32  c8       iny
. 8b33  20 d1 04 jsr $04d1    Holt Linkadresse (High) (LDA($5f),Y)
. 8b36  86 5f    stx $5f      Zeiger für indirekten
```

```
. 8b38  85 60     sta $60       Speicherzugriff neu einrichten
. 8b3a  4c 02 8b  jmp $8b02     weiterlisten

. 8b3d  4c 3e 90  jmp $903e     Neue Zeile ausgeben

. 8b40  a0 03     ldy #$03
. 8b42  84 49     sty $49       Y-Reg. zwischenspeichern
. 8b44  84 0f     sty $0f       'LIST'-Flag initialisieren
. 8b46  20 5f a4  jsr $a45f     Zeilennummer ausgeben
. 8b49  a9 20     lda #$20      Holt Leerzeichen
. 8b4b  a4 49     ldy $49       Holt Y-Reg. zurück
. 8b4d  29 7f     and #$7f      Löscht BIT 7
. 8b4f  20 b2 90  jsr $90b2     Zeichen ausgeben
. 8b52  c9 22     cmp #$22      Anführungszeichen (") ausgegeben ?
. 8b54  d0 06     bne $8b5c     Nein, dann $8b5c
. 8b56  a5 0f     lda $0f       Text-Flag flippen
. 8b58  49 ff     eor #$ff
. 8b5a  85 0f     sta $0f
. 8b5c  c8        iny           Zeiger in Zeile um 1 erhöhen
. 8b5d  f0 de     beq $8b3d     Zeile ohne Ende ?, dann Abbruch
. 8b5f  24 53     bit $53       HELP-Modus ?
. 8b61  10 03     bpl $8b66     Nein, dann $8b66
. 8b63  20 0c b7  jsr $b70c     Setzt 'FLASH ON' vor Fehlerstelle
. 8b66  20 d1 04  jsr $04d1     Holt Zeichen (LDA($5f),Y)
. 8b69  f0 50     beq $8bbb     Zeilenende ?, dann ENDE

. 8b6b  :
. 8b6b  *** KEYWORT ERZEUGEN ***
. 8b6b  :
. 8b6b  6c 06 03  jmp ($0306)   Keywort erzeugen ($8b6e)

. 8b6e  10 df     bpl $8b4f     Kein Token ?, dann Zeichen ausgeben
. 8b70  c9 ff     cmp #$ff      PI-Zeichen ?
. 8b72  f0 db     beq $8b4f     Ja, dann Zeichen ausgeben
. 8b74  24 0f     bit $0f       Textmodus ?
. 8b76  30 d7     bmi $8b4f     Ja, dann Zeichen ausgeben
. 8b78  c9 fe     cmp #$fe      User-Token ?
. 8b7a  d0 17     bne $8b93     Nein, dann $8b93
. 8b7c  c8        iny
. 8b7d  20 d1 04  jsr $04d1     Holt nächstes Zeichen (LDA($5f),Y)
. 8b80  f0 0c     beq $8b8e     Zeilenende ?, dann $8b8e
. 8b82  84 49     sty $49       Offset zwischenspeichern
. 8b84  38        sec

. 8b85  6c 0e 03  jmp ($030e)   User-Keywort erzeugen ($8b88)

. 8b88  b0 c5     bcs $8b4f     Keine User-Verzweigung, dann Fehler
. 8b8a  a0 00     ldy #$00
. 8b8c  f0 24     beq $8bb2     Springt immer nach $8bb2
. 8b8e  88        dey
. 8b8f  a9 fe     lda #$fe      User-Token wiederherstellen
. 8b91  d0 bc     bne $8b4f     Springt immer nach $8b4f
```

```
. 8b93  aa       tax          Token als Zähler ins X-Reg.
. 8b94  84 49    sty $49      Y-Reg. zwischenspeichern
. 8b96  a0 81    ldy #$81     Adresse der
. 8b98  84 23    sty $23      ... Keywort-Tabelle ($818e)
. 8b9a  a0 8e    ldy #$8e     ... nach $22, $23 (Hilfszeiger)
. 8b9c  84 22    sty $22      ... kopieren
. 8b9e  a0 00    ldy #$00
. 8ba0  ca       dex          Zähler um 1 erniedrigen
. 8ba1  10 0f    bpl $8bb2    Kleiner als $80?, dann $8bb2
. 8ba3  b1 22    lda ($22),y  Holt Zeichen aus Keywort-Tabelle
. 8ba5  48       pha          Auf Stack
. 8ba6  e6 22    inc $22      Hilfszeiger
. 8ba8  d0 02    bne $8bac    ... um 1
. 8baa  e6 23    inc $23      ... erhöhen
. 8bac  68       pla          Holt Zeichen vom Stack
. 8bad  10 f4    bpl $8ba3    Kein Wortende, dann $8ba3
. 8baf  30 ef    bmi $8ba0    Wortende, dann $8ba0
. 8bb1  c8       iny
. 8bb2  b1 22    lda ($22),y  Holt nächstes Zeichen aus Tabelle
. 8bb4  30 95    bmi $8b4b    Wortende, dann $8b4b
. 8bb6  20 b2 90 jsr $90b2    Zeichen ausgeben
. 8bb9  d0 f6    bne $8bb1    Springt immer nach $8bb1

. 8bbb  60       rts          ENDE

. 8bbc  :
. 8bbc  *** R U N ***
. 8bbc  :
. 8bbc  d0 06    bne $8bc4    Wenn Zeilennummer folgt, dann $8bc4
. 8bbe  20 20 8d jsr $8d20    PRG-Modus einsch.,AUTO und Meld.(SETMSG) aussch.
. 8bc1  4c 93 8a jmp $8a93    Textpointer initialisieren und CLR

. 8bc4  20 9a 8a jsr $8a9a    CLR-Befehl
. 8bc7  20 79 04 jsr $0479    CHRGOT
. 8bca  20 4d 8d jsr $8d4d    GOTO
. 8bcd  20 20 8d jsr $8d20    PRG-Modus einsch.,AUTO und Meld.(SETMSG) aussch.
. 8bd0  4c dc 8b jmp $8bdc    Hauptschleife

. 8bd3  :
. 8bd3  *** HAUPTSCHLEIFE ***
. 8bd3  :
. 8bd3  6c 08 03 jmp ($0308)  Hauptschleife ($8bd6)

. 8bd6  20 73 04 jsr $0473    CHRGET-Routine
. 8bd9  20 25 8c jsr $8c25    Aufruf der Basic-Routinen
. 8bdc  20 c0 8c jsr $8cc0    Prüft STOP-Taste
. 8bdf  24 81    bit $81      RUN-Modus ?
. 8be1  10 07    bpl $8bea    Nein, normal weiter
. 8be3  20 1a 8c jsr $8c1a    Textpointer zwischenspeichern
. 8be6  ba       tsx          Textpointer
. 8be7  8e f7 04 stx $04f7    ... für TRAP zwischenspeichern
. 8bea  a0 00    ldy #$00
```

```
. 8bec  20 a5 04 jsr $04a5    Holt erstes Zeichen (LDA($3b),Y)
. 8bef  f0 03    beq $8bf4    Endmarke ?, dann JMP-Befehl überspringen
. 8bf1  4c 93 8c jmp $8c93    Prüft auf Doppelpunkt (:)

. 8bf4  24 81    bit $81      RUN-Modus ?
. 8bf6  10 1f    bpl $8c17    Nein, dann 'READY.'
. 8bf8  a0 02    ldy #$02
. 8bfa  20 a5 04 jsr $04a5    Holt Linkadresse (High) (LDA($3b),Y)
. 8bfd  f0 18    beq $8c17    Programmende ?, dann 'READY.'
. 8bff  c8       iny
. 8c00  20 a5 04 jsr $04a5    Holt Zeilennummer (Low) (LDA($3b),Y)
. 8c03  85 39    sta $39      Als laufende Basic-Zeilennummer (Low) speichern
. 8c05  c8       iny
. 8c06  20 a5 04 jsr $04a5    Holt Zeilennummer (High) (LDA($3b),Y)
. 8c09  85 3a    sta $3a      Als laufende Basic-Zeilennummer (High) speichern
. 8c0b  98       tya
. 8c0c  18       clc
. 8c0d  65 3b    adc $3b      Setzt Textpointer
. 8c0f  85 3b    sta $3b      ... hinter Zeilenkopf
. 8c11  90 c0    bcc $8bd3    ... und springt
. 8c13  e6 3c    inc $3c      ... zurück
. 8c15  d0 bc    bne $8bd3    ... zum Anfang

. 8c17  4c 7e 86 jmp $867e    'READY.' ausgeben

. 8c1a  a5 3b    lda $3b      Textpointer als
. 8c1c  a4 3c    ldy $3c      ... aktuelle Zeilenadresse
. 8c1e  8d 5b 02 sta $025b    ... im Basic-Puffer
. 8c21  8c 5c 02 sty $025c    ... speichern (für CONT)
. 8c24  60       rts          ENDE

. 8c25  f0 fd    beq $8c24    Endmarke ?, dann $8c24
. 8c27  2c eb 02 bit $02eb    Prüft ob TRACE-FLAG gesetzt
. 8c2a  10 13    bpl $8c3f    Nein, dann $8c3f
. 8c2c  24 81    bit $81      RUN-Modus ?
. 8c2e  10 0f    bpl $8c3f    Nein, dann $8c3f
. 8c30  48       pha
. 8c31  a9 5b    lda #$5b     Holt 'Eckige Klammer auf'
. 8c33  20 b2 90 jsr $90b2    Zeichen ausgeben
. 8c36  20 5b a4 jsr $a45b    Gibt Zeilennummer aus
. 8c39  a9 5d    lda #$5d     Holt 'Eckige Klammer zu'
. 8c3b  20 b2 90 jsr $90b2    Zeichen ausgeben
. 8c3e  68       pla
. 8c3f  c9 fe    cmp #$fe     User-Token ?
. 8c41  f0 3f    beq $8c82    Ja, dann $8c82
. 8c43  c9 cb    cmp #$cb     'GO'-Token ?
. 8c45  f0 2d    beq $8c74    Ja, dann $8c74
. 8c47  c9 ca    cmp #$ca     'MID$'-Token ?
. 8c49  f0 20    beq $8c6b    Ja, dann $8c6b
. 8c4b  c9 fb    cmp #$fb     Funktionstoken ?
. 8c4d  b0 3e    bcs $8c8d
. 8c4f  c9 a3    cmp #$a3
```

```
. 8c51  90 06    bcc $8c59
. 8c53  c9 d5    cmp #$d5
. 8c55  90 36    bcc $8c8d     Ja, dann 'SYNTAX ERROR'
. 8c57  e9 32    sbc #$32      Token um $32 vermindern
. 8c59  38       sec
. 8c5a  e9 80    sbc #$80      Token um $80 vermindern
. 8c5c  90 32    bcc $8c90     Kein Token, Token gleich 'LET'
. 8c5e  0a       asl           Verdoppelt Ergebnis
. 8c5f  a8       tay           Ergibt Zeiger in Adresstabelle
. 8c60  b9 84 83 lda $8384,y   Adresse
. 8c63  48       pha           ... auf
. 8c64  b9 83 83 lda $8383,y   ... Stack
. 8c67  48       pha           ... legen
. 8c68  4c 73 04 jmp $0473     CHRGET-ROUTINE und ENDE

. 8c6b  a9 b6    lda #$b6      Adresse der
. 8c6d  48       pha           ... MID$-Funktion
. 8c6e  a9 5a    lda #$5a      ... aus Stack
. 8c70  48       pha           ... legen
. 8c71  4c 73 04 jmp $0473     CHRGET-ROUTINE und ENDE

. 8c74  20 73 04 jsr $0473     CHRGET-ROUTINE
. 8c77  c9 a4    cmp #$a4      'TO'-Token ?
. 8c79  d0 12    bne $8c8d     Nein, dann 'SYNTAX ERROR' ausgeben
. 8c7b  20 73 04 jsr $0473     CHRGET-Routine
. 8c7e  4c 4d 8d jmp $8d4d     GOTO-Befehl

. 8c81  00       .byte $00

. 8c82  20 73 04 jsr $0473     CHRGET-Routine
. 8c85  f0 06    beq $8c8d     Endmarke erreicht ?, dann 'SYNTAX ERROR'
. 8c87  38       sec
. 8c88  6c 10 03 jmp ($0310)   USER-Token bearbeiten ($8c8b)

. 8c8b  90 e4    bcc $8c71     USER-Token bearbeitet ?, dann $8c71
. 8c8d  4c a1 94 jmp $94a1     'SYNTAX ERROR' ausgeben

. 8c90  4c 7c 8e jmp $8e7c     LET-Befehl

. 8c93  c9 3a    cmp #$3a      Folgt Trennzeichen (:) ?
. 8c95  d0 f6    bne $8c8d     Nein, dann 'SYNTAX ERROR'
. 8c97  4c d3 8b jmp $8bd3     Zurück zum Anfang der Hauptschleife

. 8c9a  :
. 8c9a  *** R E S T O R E ***
. 8c9a  :
. 8c9a  f0 15    beq $8cb1     Ohne Zeilennummer, dann $8cb1
. 8c9c  20 e1 9d jsr $9de1     Holt Zeilennummer
. 8c9f  84 14    sty $14       Zeilennummer
. 8ca1  85 15    sta $15       ... zwischenspeichern
. 8ca3  20 3d 8a jsr $8a3d     Zeile suchen
. 8ca6  b0 03    bcs $8cab     Gefunden ?, dann keine Fehlerausgabe
```

```
. 8ca8  4c 8f 8d jmp $8d8f    'UNDEFINED STATEMENT ERROR' ausgeben

. 8cab  a5 5f    lda $5f      Holt Zeilenanfang (Low)
. 8cad  a4 60    ldy $60      Holt Zeilenanfang (High)
. 8caf  b0 05    bcs $8cb6    Springt immer nach $8cb6

. 8cb1  38       sec
. 8cb2  a5 2b    lda $2b      Holt Basic-Anfang (Low)
. 8cb4  a4 2c    ldy $2c      Holt Basic-Anfang (High)
. 8cb6  e9 01    sbc #$01     Subtrahiert
. 8cb8  b0 01    bcs $8cbb    ... 1 vom
. 8cba  88       dey          ... Basic-Anfang
. 8cbb  85 41    sta $41      Schreibt als aktuellen
. 8cbd  84 42    sty $42      ... DATA-Zeiger zurück
. 8cbf  60       rts          ENDE

. 8cc0  :
. 8cc0  *** STOP TRAP ***
. 8cc0  :
. 8cc0  20 e1 ff jsr $ffe1    STOP-Taste gedrückt ?
. 8cc3  d0 fa    bne $8cbf    Nein, dann zurück
. 8cc5  08       php
. 8cc6  ac f3 04 ldy $04f3    Holt On-Error-Flag
. 8cc9  c8       iny          On-Error-Flag gesetzt ?
. 8cca  f0 0b    beq $8cd7    Nein, dann STOP ausführen
. 8ccc  20 e1 ff jsr $ffe1    STOP-Taste noch immer gedrückt ?
. 8ccf  f0 fb    beq $8ccc    Ja, dann warten...
. 8cd1  28       plp
. 8cd2  a2 1e    ldx #$1e     Holt Fehlermeldung 30 ('BREAK')
. 8cd4  4c 83 86 jmp $8683    'BREAK ERROR' ausgeben und ENDE

. 8cd7  :
. 8cd7  *** S T O P ***
. 8cd7  :
. 8cd7  28       plp
. 8cd8  b0 01    bcs $8cdb
. 8cda  :
. 8cda  *** E N D ***
. 8cda  :
. 8cda  18       clc
. 8cdb  d0 e2    bne $8cbf
. 8cdd  24 81    bit $81      Prüft auf RUN-Modus
. 8cdf  10 0d    bpl $8cee    Nein, dann Meldung ausgeben und ENDE
. 8ce1  20 1a 8c jsr $8c1a    Textpointer speichern
. 8ce4  a5 39    lda $39      Holt laufende
. 8ce6  a4 3a    ldy $3a      ... Basic-Zeilennummer
. 8ce8  8d 59 02 sta $0259    Schreibt in Basic-Puffer
. 8ceb  8c 5a 02 sty $025a    ... für CONT-Befehl
. 8cee  68       pla          Holt Rücksprungadresse
. 8cef  68       pla          ... vom Stack
. 8cf0  90 0e    bcc $8d00    'END' ?, dann 'READY.' ausgeben
. 8cf2  20 4f ff jsr $ff4f    Meldung ausgeben
```

```
. 8cf5  0d 0a 42              .byte $13 $10
. 8cf7  42 52 45 41 4b        .byte 'BREAK'
. 8cfc  00                    .byte $00

. 8cfd  4c fb 86 jmp $86fb    Gibt 'IN' und Zeilennummer aus und ENDE

. 8d00  4c 7e 86 jmp $867e    'READY.' ausgeben

. 8d03  :
. 8d03  *** C O N T ***
. 8d03  :
. 8d03  d0 ba    bne $8cbf    Kein Trennzeichen ?, dann ENDE
. 8d05  a2 1a    ldx #$1a     Holt Fehlernummer 26 ('CAN'T CONTINUE')
. 8d07  ac 5c 02 ldy $025c    CONT-Sperre gesetzt ?
. 8d0a  d0 03    bne $8d0f    Nein, dann keine Fehlermeldung
. 8d0c  4c 83 86 jmp $8683    'CAN'T CONTINUE ERROR' ausgeben

. 8d0f  ad 5b 02 lda $025b    Textpointer
. 8d12  85 3b    sta $3b      ... wieder-
. 8d14  84 3c    sty $3c      ... herstellen
. 8d16  ad 59 02 lda $0259    Zeilen-
. 8d19  ac 5a 02 ldy $025a    ... nummer
. 8d1c  85 39    sta $39      ... wieder-
. 8d1e  84 3a    sty $3a      ... herstellen
. 8d20  a9 80    lda #$80     Schaltet
. 8d22  85 81    sta $81      ... RUN-Modus ein
. 8d24  0a       asl          Akku gleich 0
. 8d25  85 73    sta $73      AUTO-Befehl
. 8d27  85 74    sta $74      ... initialisieren
. 8d29  4c 90 ff jmp $ff90    (SETMSG) Meldung abschalten

. 8d2c  :
. 8d2c  *** G O S U B ***
. 8d2c  :
. 8d2c  a0 05    ldy #$05     Schafft Platz für 5 Byte
. 8d2e  20 05 89 jsr $8905    ... im Basic-Stack
. 8d31  88       dey          Schreibt
. 8d32  a5 3c    lda $3c      ... Textpointer
. 8d34  91 7c    sta ($7c),y  ... in
. 8d36  88       dey          ... Basic-
. 8d37  a5 3b    lda $3b
. 8d39  91 7c    sta ($7c),y  ... Stack
. 8d3b  88       dey          Schreibt
. 8d3c  a5 3a    lda $3a      ... laufende
. 8d3e  91 7c    sta ($7c),y  ... Basic-
. 8d40  88       dey          ... Zeilennummer
. 8d41  a5 39    lda $39      ... in
. 8d43  91 7c    sta ($7c),y  Basic-Stack
. 8d45  88       dey          Schreibt
. 8d46  a9 8d    lda #$8d     ... 'GOSUB'-Token
. 8d48  91 7c    sta ($7c),y  ... in Basic-Stack
```

```
. 8d4a  20 79 04 jsr $0479    CHRGOT-Routine
. 8d4d  :
. 8d4d  *** G O T O ***
. 8d4d  :
. 8d4d  20 3e 8e jsr $8e3e    Holt Zeilennummer
. 8d50  20 c1 8d jsr $8dc1    Y-Reg. gleich Offset zum Zeilenende
. 8d53  38       sec
. 8d54  a5 39    lda $39      Aktuelle Zeilennummer
. 8d56  e5 14    sbc $14      ... größer als
. 8d58  a5 3a    lda $3a      ... gerufene
. 8d5a  e5 15    sbc $15      ... Zeilennummer ?
. 8d5c  b0 0b    bcs $8d69    Ja, dann $8d69
. 8d5e  98       tya          Setzt
. 8d5f  38       sec          ... Textpointer
. 8d60  65 3b    adc $3b      ... auf nächste
. 8d62  a6 3c    ldx $3c      ... Zeile und
. 8d64  90 07    bcc $8d6d    ... verzweigt
. 8d66  e8       inx          ... nach
. 8d67  b0 04    bcs $8d6d    ... $8d6d

. 8d69  a5 2b    lda $2b      Holt Programmanfang (Low)
. 8d6b  a6 2c    ldx $2c      Holt Programmanfang (High)
. 8d6d  20 41 8a jsr $8a41    Zeile ab Adresse (Akku, X-Reg.) suchen
. 8d70  90 1d    bcc $8d8f    Nicht gefunden ?, dann $8d8f
. 8d72  a5 5f    lda $5f      Holt Zeilenadresse,
. 8d74  e9 01    sbc #$01     ... subtrahiert 1
. 8d76  85 3b    sta $3b      ... und
. 8d78  a5 60    lda $60      ... schreibt
. 8d7a  e9 00    sbc #$00     ... als Textpointer
. 8d7c  85 3c    sta $3c      ... zurück
. 8d7e  24 81    bit $81      RUN-Modus ?
. 8d80  10 9e    bpl $8d20    PRG-Modus einsch.,AUTO und Meld (SETMSG) aussch.
. 8d82  60       rts          ENDE

. 8d83  :
. 8d83  *** R E T U R N ***
. 8d83  :
. 8d83  a9 8d    lda #$8d     Holt 'GOSUB'-Token
. 8d85  85 02    sta $02      Zur Suche im Stack zwischenspeichern
. 8d87  20 71 88 jsr $8871    Sucht nach 'GOSUB'-Token im Stack
. 8d8a  f0 08    beq $8d94    Gefunden ?, dann $8d94
. 8d8c  a2 0c    ldx #$0c     Holt Fehlernummer 12 ('RETURN WITHOUT GOSUB')
. 8d8e  2c       .byte $2c

. 8d8f  a2 11    ldx #$11     Holt Fehlernummer 17 ('UNDEFINED STATEMENT')
. 8d91  4c 83 86 jmp $8683    Gibt Fehlermeldung aus

. 8d94  20 69 a7 jsr $a769    Kopiert Stack-Pointer
. 8d97  a0 05    ldy #$05     Zahl der Bytes im Stack
. 8d99  20 72 a7 jsr $a772    Stack-Pointer um Y-Reg. erhöhen
. 8d9c  88       dey          Textpointer
. 8d9d  b1 3d    lda ($3d),y
```

```
. 8d9f  85 3c    sta $3c
. 8da1  88       dey
. 8da2  b1 3d    lda ($3d),y
. 8da4  85 3b    sta $3b        ... wiederherstellen
. 8da6  88       dey            Stellt
. 8da7  b1 3d    lda ($3d),y    ... aktuelle
. 8da9  20 7f cd jsr $cd7f      ... (Adresse $cd7f gleich Rucksack für LOOP)
. 8dac  b1 3d    lda ($3d),y    ... Zeilennummer
. 8dae  85 39    sta $39        ... her
. 8db0  :
. 8db0  *** D A T A ***
. 8db0  :
. 8db0  20 be 8d jsr $8dbe      Y-Reg. gleich Offset zum Trennzeichen
. 8db3  98       tya            Offset in Akku
. 8db4  18       clc            Programmzeiger
. 8db5  65 3b    adc $3b        ... auf
. 8db7  85 3b    sta $3b        ... nächstes
. 8db9  90 02    bcc $8dbd      ... Trennzeichen
. 8dbb  e6 3c    inc $3c        ... setzen
. 8dbd  60       rts            ENDE

. 8dbe  :
. 8dbe  *** Y-Reg. gleich Offset zum Trennzeichen ***
. 8dbe  :
. 8dbe  a2 3a    ldx #$3a       Holt Trennzeichen (:)
. 8dc0  2c       .byte $2c
. 8dc1  a2 00    ldx #$00       Holt Zeilenendemarke
. 8dc3  86 07    stx $07
. 8dc5  a0 00    ldy #$00       Einziges Trennzeichen innerhalb
. 8dc7  84 08    sty $08        ... eines Textstrings
. 8dc9  a5 08    lda $08        Trennzeichen
. 8dcb  a6 07    ldx $07
. 8dcd  85 07    sta $07
. 8dcf  86 08    stx $08        ... austauschen
. 8dd1  20 a5 04 jsr $04a5      Holt Zeichen aus Zeile (LDA($3b),Y)
. 8dd4  f0 e7    beq $8dbd      Endmarke ?, dann ENDE
. 8dd6  c5 08    cmp $08        Trennzeichen bzw. Endmarke ?
. 8dd8  f0 e3    beq $8dbd      Ja, dann ENDE
. 8dda  c8       iny
. 8ddb  c9 22    cmp #$22       Anführungszeichen (") ?
. 8ddd  d0 f2    bne $8dd1      Nein, dann nächstes Zeichen holen
. 8ddf  f0 e8    beq $8dc9      Sonst Trennzeichen ignorieren

. 8de1  :
. 8de1  *** I F ***
. 8de1  :
. 8de1  20 2c 93 jsr $932c      (FRMEVL) Ausdruck auswerten
. 8de4  20 79 04 jsr $0479      CHRGOT-Routine
. 8de7  c9 89    cmp #$89       'GOTO'-Token ?
. 8de9  f0 05    beq $8df0      Ja, dann $8df0
. 8deb  a9 a7    lda #$a7       Holt 'THEN'-Token
. 8ded  20 93 94 jsr $9493      Auswertung
```

```
. 8df0  a5 61    lda $61      Holt Exponent von FAC
. 8df2  d0 1c    bne $8e10    Bedingung erfüllt, dann $8e10
. 8df4  20 b0 8d jsr $8db0    Textpointer auf Trennzeichen setzen
. 8df7  a0 00    ldy #$00
. 8df9  20 a5 04 jsr $04a5    Holt Zeichen aus Zeile (LDA($3b),Y)
. 8dfc  f0 0d    beq $8e0b    Endmarke ?, dann $8e0b
. 8dfe  20 73 04 jsr $0473    CHRGET-Routine
. 8e01  c9 d5    cmp #$d5     'ELSE'-Token ?
. 8e03  d0 ef    bne $8df4    Nein, dann $8df4
. 8e05  20 73 04 jsr $0473    CHRGET-Routine
. 8e08  4c 10 8e jmp $8e10    Sprung nach $8e10

. 8e0b  :
. 8e0b  *** R E M ,   E L S E ***
. 8e0b  :
. 8e0b  20 c1 8d jsr $8dc1    Textpointer auf Zeilenende setzen
. 8e0e  f0 a3    beq $8db3    Springt immer nach $8db3

. 8e10  20 79 04 jsr $0479    CHRGOT-Routine
. 8e13  b0 03    bcs $8e18    keine Zeilennummer, dann $8e18
. 8e15  4c 4d 8d jmp $8d4d    GOTO

. 8e18  4c 25 8c jmp $8c25    Routinenaufruf (in Hauptschleife)

. 8e1b  :
. 8e1b  *** O N ***
. 8e1b  :
. 8e1b  20 84 9d jsr $9d84    Numerischen Ausdruck auswerten
. 8e1e  48       pha          Holt nächstes Zeichen
. 8e1f  c9 8d    cmp #$8d     'GOSUB'-Token ?
. 8e21  f0 07    beq $8e2a    Ja, dann kein Fehler
. 8e23  c9 89    cmp #$89     'GOTO'-Token ?
. 8e25  f0 03    beq $8e2a    Ja, dann kein Fehler
. 8e27  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. 8e2a  c6 65    dec $65      Parameter um 1 erniedrigen
. 8e2c  d0 04    bne $8e32    Noch nicht 0, dann $8e32
. 8e2e  68       pla          Holt nächstes Zeichen
. 8e2f  4c 3f 8c jmp $8c3f    Hauptschleife

. 8e32  20 73 04 jsr $0473    CHRGET-Routine
. 8e35  20 3e 8e jsr $8e3e    Holt Zeilennummer
. 8e38  c9 2c    cmp #$2c     Folgt Komma (,) ?
. 8e3a  f0 ee    beq $8e2a    Ja, dann $8e2a
. 8e3c  68       pla          Stackzeiger wiederherstellen
. 8e3d  60       rts          ENDE

. 8e3e  :
. 8e3e  *** Zeilennummer in Adreßformat wandeln ***
. 8e3e  :
. 8e3e  a2 00    ldx #$00
. 8e40  86 08    stx $08      Flag für 'Zeilennummer vorhanden' initialis.
```

```
. 8e42  86 14     stx $14      Zeilennummer
. 8e44  86 15     stx $15      ... initialisieren
. 8e46  b0 f5     bcs $8e3d    Keine Ziffer im Akku ?, dann ENDE
. 8e48  e6 08     inc $08      Flag für 'Zeilennummer vorhanden' setzen
. 8e4a  e9 2f     sbc #$2f     Subtrahiert $30 wenn Carry gleich 0
. 8e4c  85 07     sta $07      Stellenwert zwischenspeichern
. 8e4e  a5 15     lda $15      Holt Zwischenergebnis (High)
. 8e50  85 22     sta $22      Speichert Zwischenergebnis in Hilfszeiger
. 8e52  c9 19     cmp #$19     Zeilennummer größer 63999 ?
. 8e54  b0 cd     bcs $8e23    Ja, dann 'SYNTAX ERROR'
. 8e56  a5 14     lda $14      Multipliziert
. 8e58  0a        asl          ... Zwischenergebnis (Low)
. 8e59  26 22     rol $22
. 8e5b  0a        asl
. 8e5c  26 22     rol $22      ... mit 4
. 8e5e  65 14     adc $14      Addiert Ergebnis zum Zwischenergebnis
. 8e60  85 14     sta $14      Gleich
. 8e62  a5 22     lda $22      ... Zwischenergebnis
. 8e64  65 15     adc $15
. 8e66  85 15     sta $15      ... mal 5
. 8e68  06 14     asl $14      Multipliziert Zwischenergebnis mit 2
. 8e6a  26 15     rol $15      Gleich Zwischenergebnis mal 10
. 8e6c  a5 14     lda $14
. 8e6e  65 07     adc $07      Addiert nächsten Stellenwert
. 8e70  85 14     sta $14      Gleich
. 8e72  90 02     bcc $8e76    ... neues
. 8e74  e6 15     inc $15      ... Zwischenergebnis
. 8e76  20 73 04  jsr $0473    CHRGET-Routine
. 8e79  4c 46 8e  jmp $8e46    Zurück zum Anfang

. 8e7c  :
. 8e7c  *** L E T ***
. 8e7c  :
. 8e7c  20 a5 96  jsr $96a5    Variable suchen oder anlegen
. 8e7f  85 49     sta $49      Schreibt
. 8e81  84 4a     sty $4a      ... Variablenzeiger
. 8e83  a9 b2     lda #$b2     Holt '='-Token
. 8e85  20 93 94  jsr $9493    Auswertung
. 8e88  a5 0e     lda $0e      Holt Datentyp ($80=Integer, $00=Fließkomma)
. 8e8a  48        pha          auf Stack
. 8e8b  a5 0d     lda $0d      Holt Datentyp ($FF=String, $00=Numerisch)
. 8e8d  48        pha          auf Stack
. 8e8e  20 2c 93  jsr $932c    Ausdruck auswerten
. 8e91  68        pla          Holt String-Flag
. 8e92  2a        rol          Setzt Carry bei String
. 8e93  20 1b 93  jsr $931b    Variablentyp prüfen
. 8e96  d0 18     bne $8eb0    String ?, dann $8eb0
. 8e98  68        pla          Integer oder Fließkomma ?
. 8e99  10 12     bpl $8ead    Fließkomma, dann $8ead
. 8e9b  20 a0 a2  jsr $a2a0    FAC runden
. 8e9e  20 86 98  jsr $9886    Fließkomma-Integer-Wandlung
. 8ea1  a0 00     ldy #$00     Integer in
```

```
. 8ea3  a5 64     lda $64       ... Variable schreiben
. 8ea5  91 49     sta ($49),y   Variable
. 8ea7  c8        iny
. 8ea8  a5 65     lda $65
. 8eaa  91 49     sta ($49),y   ... ablegen
. 8eac  60        rts           ENDE

. 8ead  4c 55 a2  jmp $a255     FAC in Variable übertragen
. 8ead  :
. 8ead  *** Stringzuweisung N$ = A$ ***
. 8ead  :
. 8eb0  68        pla
. 8eb1  a4 4a     ldy $4a       Holt Zeiger (High) von N$
. 8eb3  c0 04     cpy #$04      Deskriptor in Page 4 ?
. 8eb5  d0 72     bne $8f29     Nein, dann $8f29
. 8eb7  :
. 8eb7  *** Stringzuweisung an TI$ ***
. 8eb7  :
. 8eb7  20 4e 9c  jsr $9c4e     (FRESTR)
. 8eba  c9 06     cmp #$06      Stringlänge gleich 6 ?
. 8ebc  d0 3e     bne $8efc     Nein, dann 'ILLEGAL QUANTITY ERROR'
. 8ebe  a0 00     ldy #$00      FAC
. 8ec0  84 61     sty $61       ... Exponent
. 8ec2  84 66     sty $66       ... initialisieren
. 8ec4  84 71     sty $71       Stellenzähler initialisieren
. 8ec6  20 f4 8e  jsr $8ef4     Zeichen gleich Ziffer ?
. 8ec9  20 62 a1  jsr $a162     FAC gleich FAC mal 10
. 8ecc  e6 71     inc $71       Stellenzähler um 1 erhöhen
. 8ece  a4 71     ldy $71       Holt Stellenzähler
. 8ed0  20 f4 8e  jsr $8ef4     Zeichen gleich Ziffer ?
. 8ed3  20 91 a2  jsr $a291     ARG gleich FAC
. 8ed6  aa        tax
. 8ed7  f0 05     beq $8ede     FAC gleich 0 ?, dann $8ede
. 8ed9  e8        inx
. 8eda  8a        txa
. 8edb  20 6d a1  jsr $a16d     FAC gleich FAC + ARG
. 8ede  a4 71     ldy $71       Holt Stellenzähler
. 8ee0  c8        iny           Erhöht Stellenzähler
. 8ee1  c0 06     cpy #$06      Schon 6 Stellen ?
. 8ee3  d0 df     bne $8ec4     Nein, dann $8ec4
. 8ee5  20 62 a1  jsr $a162     FAC gleich FAC mal 10
. 8ee8  20 27 a3  jsr $a327     Gleitkomma-Integer-Wandlung
. 8eeb  a6 64     ldx $64       Holt Uhrzeit nach X-Register,
. 8eed  a4 63     ldy $63       ... Y-Register
. 8eef  a5 65     lda $65       ... und Akku
. 8ef1  4c db ff  jmp $ffdb     Uhr stellen

. 8ef4  20 b0 04  jsr $04b0     Indirekter Speicherzugriff (LDA($22),Y)
. 8ef7  20 85 04  jsr $0485     Zeichen gleich Ziffer ?
. 8efa  90 03     bcc $8eff     Ja, dann kein Fehler
. 8efc  4c 1c 99  jmp $991c     'ILLEGAL QUANTITY ERROR' ausgeben
```

```
. 8eff  e9 2f     sbc #$2f      subtrahiert $30
. 8f01  4c 0a a4  jmp $a40a     Ergebnis nach FAC und ARG

. 8f04  :
. 8f04  *** N$ gleich A$ ***
. 8f04  :
. 8f04  68        pla           Holt Adresse (High) nach A$
. 8f05  c8        iny
. 8f06  c5 34     cmp $34       Vergleich mit Stringbereich-Anfang (High)
. 8f08  90 18     bcc $8f22     A$ nicht im Stringbereich ?, dann $8f22
. 8f0a  d0 08     bne $8f14
. 8f0c  88        dey
. 8f0d  20 dc 04  jsr $04dc     Holt Adresse (Low) von A$ (LDA($64),Y)
. 8f10  c5 33     cmp $33       Vergleich mit Stringbereich-Anfang (Low)
. 8f12  90 0e     bcc $8f22     A$ nicht im Stringbereich ?, dann $8f22
. 8f14  a4 65     ldy $65       Holt Zeichen (High) von A$
. 8f16  c4 2e     cpy $2e       Vergleich mit Variablenanfang (High)
. 8f18  90 08     bcc $8f22     Deskript.v. A$ nicht in Var.-Tab.?, dann $8f22
. 8f1a  d0 24     bne $8f40
. 8f1c  a5 64     lda $64       Holt Zeiger (Low) von A$
. 8f1e  c5 2d     cmp $2d       Vergleich mit Variablenanfang (Low)
. 8f20  b0 1e     bcs $8f40     Deskript.v. A$ nicht in Var.-Tab.?, dann $8f40
. 8f22  a5 64     lda $64       Holt Zeiger (Low) von A$
. 8f24  a4 65     ldy $65       Holt Zeiger (High) von A$
. 8f26  4c 5e 8f  jmp $8f5e     Sprung nach $8f5e
. 8f29  a0 02     ldy #$02
. 8f2b  20 dc 04  jsr $04dc     Holt Adresse (High) von A$ (LDA($64),Y)
. 8f2e  c5 7b     cmp $7b       Gleich Adresse (High) von D$ ?
. 8f30  d0 d4     bne $8f06     Nein, dann $8f06
. 8f32  48        pha
. 8f33  88        dey
. 8f34  20 dc 04  jsr $04dc     Holt Adresse (Low) von A$ (LDA($64),Y)
. 8f37  c5 7a     cmp $7a       Gleich Adresse (Low) von D$ ?
. 8f39  d0 c9     bne $8f04     Nein, dann $8f04
. 8f3b  a5 79     lda $79       Holt Länge von D$
. 8f3d  f0 c5     beq $8f04     Gleich 0, dann $8f04
. 8f3f  68        pla           Stack bereinigen
. 8f40  :
. 8f40  *** String kopieren ***
. 8f40  :
. 8f40  a0 00     ldy #$00
. 8f42  20 dc 04  jsr $04dc     Holt Länge von A$ (LDA($64),Y)
. 8f45  20 54 9b  jsr $9b54     Reserviert Platz
. 8f48  a5 50     lda $50       Holt Zeiger (Low) von A$
. 8f4a  a4 51     ldy $51       Holt Zeiger (High) von A$
. 8f4c  85 6f     sta $6f       In FAC
. 8f4e  84 70     sty $70       ... ablegen
. 8f50  20 1b 9c  jsr $9c1b     A$ in Stringbereich kopieren
. 8f53  a5 6f     lda $6f       Holt Zeiger (Low) von A$
. 8f55  a4 70     ldy $70       Holt Zeiger (High) von A$
. 8f57  20 aa 9c  jsr $9caa     Holt Deskriptoren von A$ aus String-Stack
. 8f5a  a9 61     lda #$61      Holt Zeiger (Low) von A$
```

```
. 8f5c  a0 00     ldy #$00      Holt Zeiger (High) von A$
. 8f5e  85 50     sta $50
. 8f60  84 51     sty $51
. 8f62  85 22     sta $22
. 8f64  84 23     sty $23
. 8f66  20 aa 9c  jsr $9caa     Holt Deskriptoren von A$ aus String-Stack
. 8f69  20 9c 8f  jsr $8f9c     Prüft ob R-Zeiger vorhanden
. 8f6c  90 0b     bcc $8f79     Nein, dann $8f79
. 8f6e  a0 00     ldy #$00
. 8f70  a5 49     lda $49       Holt
. 8f72  91 22     sta ($22),y   ... Zeiger
. 8f74  c8        iny           ... von R$
. 8f75  a5 4a     lda $4a       ... und
. 8f77  91 22     sta ($22),y   ... kopiert
. 8f79  a5 49     lda $49       ... in
. 8f7b  85 22     sta $22
. 8f7d  a5 4a     lda $4a
. 8f7f  85 23     sta $23       ... R-Zeiger
. 8f81  20 9c 8f  jsr $8f9c     Prüft ob R-Zeiger vorhanden
. 8f84  90 09     bcc $8f8f     Nein, dann $8f8f
. 8f86  88        dey
. 8f87  a9 ff     lda #$ff      Flag für ungültigen String
. 8f89  91 22     sta ($22),y   Alten Inhalt von N$ ungültig machen
. 8f8b  88        dey
. 8f8c  8a        txa           Länge des alten Inhaltes von N$
. 8f8d  91 22     sta ($22),y
. 8f8f  a0 02     ldy #$02
. 8f91  a9 50     lda #$50
. 8f93  20 94 04  jsr $0494     Indirekter Speicherzugriff (LDA(Akku),Y-Reg.)
. 8f96  91 49     sta ($49),y   Deskriptoren von N$ gleich Deskriptoren von A$
. 8f98  88        dey
. 8f99  10 f6     bpl $8f91
. 8f9b  60        rts           ENDE

. 8f9c  :
. 8f9c  *** String auf R-Zeiger prüfen ***
. 8f9c  :
. 8f9c  a0 00     ldy #$00
. 8f9e  20 b0 04  jsr $04b0     Holt Stringlänge (LDA($22),Y)
. 8fa1  48        pha           Auf Stack legen
. 8fa2  f0 39     beq $8fdd     Stringlänge gleich 0, dann $8fdd
. 8fa4  c8        iny
. 8fa5  20 b0 04  jsr $04b0     Holt Stringadresse (Low) (LDA($22),Y)
. 8fa8  aa        tax
. 8fa9  c8        iny
. 8faa  20 b0 04  jsr $04b0     Holt Stringadresse (High) (LDA($22),Y)
. 8fad  c5 38     cmp $38       String
. 8faf  90 06     bcc $8fb7     ... oberhalb
. 8fb1  d0 2a     bne $8fdd     ... der
. 8fb3  e4 37     cpx $37       ... Basic-Obergrenze ?
. 8fb5  b0 26     bcs $8fdd     Ja, dann $8fdd
. 8fb7  20 b0 04  jsr $04b0     Holt Stringadresse (High) (LDA($22),Y)
```

```
. 8fba  c5 34     cmp $34       String
. 8fbc  90 1f     bcc $8fdd     ... unterhalb
. 8fbe  d0 04     bne $8fc4     ... des
. 8fc0  e4 33     cpx $33       ... String-Bereichs
. 8fc2  90 19     bcc $8fdd     Ja, dann $8fdd
. 8fc4  c5 7b     cmp $7b       String
. 8fc6  d0 04     bne $8fcc     ... identisch
. 8fc8  e4 7a     cpx $7a       ... mit DS$ ?
. 8fca  f0 11     beq $8fdd     Ja, dann $8fdd
. 8fcc  86 22     stx $22       Zeiger ($22,$23)
. 8fce  85 23     sta $23       ... auf Stringanfang
. 8fd0  68        pla           Stringlänge
. 8fd1  aa        tax
. 8fd2  18        clc
. 8fd3  65 22     adc $22       ... addieren
. 8fd5  85 22     sta $22
. 8fd7  90 02     bcc $8fdb     ... und auf
. 8fd9  e6 23     inc $23       ... R-Zeiger setzen
. 8fdb  38        sec           Flag für 'R-Zeiger vorhanden'
. 8fdc  60        rts           ENDE
. 8fdd  68        pla
. 8fde  18        clc           Flag für 'R-Zeiger nicht vorhanden'
. 8fdf  60        rts           ENDE

. 8fe0  :
. 8fe0  *** P R I N T # ***
. 8fe0  :
. 8fe0  20 e6 8f jsr $8fe6     CMD-Befehl
. 8fe3  4c fe 90 jmp $90fe     (CLRCH) Ein-/Ausgabe-Kanal schließen

. 8fe6  :
. 8fe6  *** C M D ***
. 8fe6  :
. 8fe6  20 84 9d jsr $9d84     Holt logische Adresse
. 8fe9  f0 05     beq $8ff0    Folgt Trennzeichen ?, dann $8ff0
. 8feb  a9 2c     lda #$2c     Holt Komma (,)
. 8fed  20 93 94 jsr $9493     Auswertung
. 8ff0  08        php
. 8ff1  86 13     stx $13      Ein-/Ausgabe-Kanal
. 8ff3  20 97 a7 jsr $a797     (CKOUT)
. 8ff6  28        plp
. 8ff7  4c 00 90 jmp $9000     PRINT

. 8ffa  20 8b 90 jsr $908b     Gibt String aus
. 8ffd  20 79 04 jsr $0479     CHRGOT
. 9000  :
. 9000  *** P R I N T ***
. 9000  :
. 9000  f0 3c     beq $903e    Folgt Trennzeichen ?, dann $903e
. 9002  c9 fb     cmp #$fb     'USING'-Token
. 9004  d0 03     bne $9009    Nein, dann $9009
. 9006  4c f7 ae jmp $aef7     PRINT USING
```

```
. 9009  f0 43     beq $904e
. 900b  c9 a3     cmp #$a3      'TAB('-Token ?
. 900d  f0 50     beq $905f     Ja, dann $905f
. 900f  c9 a6     cmp #$a6      'SPC('-Token ?
. 9011  18        clc
. 9012  f0 4b     beq $905f     Ja, dann $905f
. 9014  c9 2c     cmp #$2c      Komma (,) ?
. 9016  f0 37     beq $904f     Ja, dann $904f (10'er Tab)
. 9018  c9 3b     cmp #$3b      Strichpunkt (;) ?
. 901a  f0 5e     beq $907a     Ja, dann $907a
. 901c  20 2c 93  jsr $932c     Ausdruck auswerten
. 901f  24 0d     bit $0d       String ?
. 9021  30 d7     bmi $8ffa     Ja, dann String ausgeben
. 9023  20 6f a4  jsr $a46f     FAC in Zahlenstring umwandeln
. 9026  20 74 9b  jsr $9b74     String übernehmen
. 9029  20 8b 90  jsr $908b     String ausgeben
. 902c  20 a6 90  jsr $90a6     'Cursor nach rechts' ausgeben
. 902f  d0 cc     bne $8ffd     Springt immer nach $8ffd

. 9031  :
. 9031  *** Eingabepuffer abschließen ***
. 9031  :
. 9031  a9 00     lda #$00      Holt Endmarke
. 9033  9d 00 02  sta $0200,x   Schreibt Endmarke in Puffer
. 9036  a2 ff     ldx #$ff      Pufferadresse -1
. 9038  a0 01     ldy #$01      ... nach (X-Reg., Y-Reg.)
. 903a  a5 13     lda $13       Holt Ein-/Ausgabe-Kanal
. 903c  d0 10     bne $904e     Nicht Tastatur ?, dann $904e
. 903e  a9 0d     lda #$0d      Holt CR-Code
. 9040  20 b2 90  jsr $90b2     Gibt CR-Code aus
. 9043  24 13     bit $13       Logische Adresse kleiner 128 ?
. 9045  10 05     bpl $904c     Ja, dann $904c
. 9047  a9 0a     lda #$0a      Holt 'IF'-Code
. 9049  20 b2 90  jsr $90b2     ausgeben
. 904c  49 ff     eor #$ff      invertieren
. 904e  60        rts           ENDE

. 904f  :
. 904f  *** 10'er TAB ***
. 904f  :
. 904f  38        sec
. 9050  20 f0 ff  jsr $fff0     (PLOT) Holt Cursorspalte nach Y-Reg.
. 9053  98        tya           Von Cursorspalte (Y-Reg.)
. 9054  38        sec           ... wird so lange
. 9055  e9 0a     sbc #$0a      ... 10 subtrahiert
. 9057  b0 fc     bcs $9055     ... bis Ergebnis negativ
. 9059  49 ff     eor #$ff      Vorzeichenwechsel
. 905b  69 01     adc #$01      Ergibt Schritte bis Tab-Position
. 905d  d0 16     bne $9075     Springt immer nach $9075

. 905f  :
```

```
. 905f  *** T A B ( ,   S P C ( ***
. 905f  :
. 905f  08       php          TAB( : Carry=1; SPC( :Carry=0
. 9060  38       sec
. 9061  20 f0 ff jsr $fff0    (PLOT) Holt Cursorspalte nach Y-Reg.
. 9064  84 09    sty $09      Cursorspalte zwischenspeichern
. 9066  20 81 9d jsr $9d81    Holt Parameter
. 9069  c9 29    cmp #$29     Nächstes Zeichen ')' ?
. 906b  d0 13    bne $9080    Nein, dann 'SYNTAX ERROR'
. 906d  28       plp          TAB/SPC-Flag
. 906e  90 06    bcc $9076    SPC( ?, dann $9076
. 9070  8a       txa          Holt Parameterwert
. 9071  e5 09    sbc $09      Subtrahiert Cursorspalte
. 9073  90 05    bcc $907a    Kleiner 0 ?, dann $907a
. 9075  aa       tax          Differenz als Zähler verwenden
. 9076  e8       inx
. 9077  ca       dex          So oft 'Cursor nach rechts'
. 9078  d0 09    bne $9083    ... bis Zähler gleich 0
. 907a  20 73 04 jsr $0473    CHRGET-Routine
. 907d  4c 09 90 jmp $9009    PRINT

. 9080  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. 9083  20 a6 90 jsr $90a6    'Cursor nach rechts' ausgeben
. 9086  d0 ef    bne $9077    Springt immer nach $9077

. 9088  :
. 9088  *** String ausgeben ***
. 9088  :
. 9088  20 74 9b jsr $9b74    String übernehmen
. 908b  20 4e 9c jsr $9c4e    (FRESTR)
. 908e  aa       tax          Stringlänge nach X-Reg.
. 908f  a0 00    ldy #$00
. 9091  e8       inx
. 9092  ca       dex
. 9093  f0 b9    beq $904e    String fertig ausgegeben ?, dann $904e
. 9095  20 b0 04 jsr $04b0    Holt Zeichen aus String (LDA($22),Y)
. 9098  20 b2 90 jsr $90b2    Zeichen ausgeben
. 909b  c8       iny
. 909c  c9 0d    cmp #$0d     CR-Code ?
. 909e  d0 f2    bne $9092    Nein, dann $9092
. 90a0  20 4c 90 jsr $904c    Zeichen invertieren
. 90a3  4c 92 90 jmp $9092    Holt nächstes Zeichen
. 90a6  :
. 90a6  *** Zeichenausgabe ***
. 90a6  :
. 90a6  a5 13    lda $13      Holt Ein-/Ausgabe-Kanal
. 90a8  f0 03    beq $90ad    Eingabegerät gleich Tastatur ?, dann $90ad
. 90aa  a9 20    lda #$20     Holt Leerzeichen (an Peripherie)

. 90ac  2c       .byte $2c
```

```
. 90ad  a9 1d     lda #$1d      Holt 'Cursor nach rechts' (an Bildschirm)

. 90af  2c        .byte $2c

. 90b0  a9 3f     lda #$3f      Holt Fragezeichen (?)
. 90b2  20 8b a7  jsr $a78b     Zeichenausgabe
. 90b5  29 ff     and #$ff      Setzt Flags
. 90b7  60        rts           ENDE

. 90b8  :
. 90b8  *** G E T ***
. 90b8  :
. 90b8  20 86 9a  jsr $9a86     GET im Direktmodus ?, dann Fehler
. 90bb  85 80     sta $80       Nachfolgendes Zeichen zwischenspeichern
. 90bd  c9 23     cmp #$23      '#'-Zeichen ?
. 90bf  f0 0a     beq $90cb     Ja, dann $90cb
. 90c1  c9 f9     cmp #$f9      'KEY'-Token ?
. 90c3  d0 16     bne $90db     Nein, dann $90db
. 90c5  20 73 04  jsr $0473     CHRGET-Routine
. 90c8  4c db 90  jmp $90db     Sprung nach $90db

. 90cb  20 73 04  jsr $0473     CHRGET-Routine
. 90ce  20 84 9d  jsr $9d84     (GETBYT) Holt logische Adresse
. 90d1  a9 2c     lda #$2c      Holt Komma (,)
. 90d3  20 93 94  jsr $9493     Auswertung
. 90d6  86 13     stx $13       Ein-/Ausgabe-Kanal gleich logische Adresse
. 90d8  20 a6 a7  jsr $a7a6     (CHKIN) Eingabe-Umlenkung
. 90db  a2 01     ldx #$01      Zeiger in
. 90dd  a0 02     ldy #$02      ... Eingabepuffer
. 90df  a9 00     lda #$00      Holt Endmarke ($00)
. 90e1  8d 01 02  sta $0201     Schreibt Endmarke hinter 1. Pufferbyte
. 90e4  a9 40     lda #$40      Holt 'GET'-Flag
. 90e6  20 58 91  jsr $9158     Bringt Zeichen in Variable
. 90e9  a6 13     ldx $13       Holt Ein-/Ausgabe-Kanal
. 90eb  d0 13     bne $9100     Eingabegerät nicht Tastatur ?, dann $9100
. 90ed  60        rts           ENDE

. 90ee  :
. 90ee  *** I N P U T # ***
. 90ee  :
. 90ee  20 84 9d  jsr $9d84     (GETBYT) Holt logische Adresse
. 90f1  a9 2c     lda #$2c      Holt Komma (,)
. 90f3  20 93 94  jsr $9493     Auswertung
. 90f6  86 13     stx $13       Ein-/Ausgabe-Kanal gleich logische Adresse
. 90f8  20 a6 a7  jsr $a7a6     (CHKIN) Eingabe-Umlenkung
. 90fb  20 17 91  jsr $9117     INPUT
. 90fe  a5 13     lda $13       Holt Ein-/Ausgabe-Kanal
. 9100  20 cc ff  jsr $ffcc     (CLRCH) Ein-/Ausgabe-Kanal schließen
. 9103  a2 00     ldx #$00      Eingabegerät
. 9105  86 13     stx $13       ... gleich Tastatur
. 9107  60        rts           ENDE
```

```
. 9108  :
. 9108  *** I N P U T ***
. 9108  :
. 9108  c9 22    cmp #$22     Folgt Anführungszeichen (") ?
. 910a  d0 0b    bne $9117    Nein, dann $9117
. 910c  20 4e 94 jsr $944e    Übernimmt Fragestring
. 910f  a9 3b    lda #$3b     Holt Strichpunkt (;)
. 9111  20 93 94 jsr $9493    Auswertung
. 9114  20 8b 90 jsr $908b    String ausgeben
. 9117  20 86 9a jsr $9a86    INPUT im Direktmodus ?, dann Fehler
. 911a  a9 2c    lda #$2c     Holt Komma (,)
. 911c  8d ff 01 sta $01ff    Vor Eingabepuffer schreiben
. 911f  20 42 91 jsr $9142    Fragezeichen ausgeben und zur Hauptschl. springen
. 9122  a5 13    lda $13      Holt Ein-/Ausgabe-Kanal
. 9124  f0 0d    beq $9133    Eingabegerät gleich Tastatur ?, dann $9133
. 9126  20 b7 ff jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. 9129  29 02    and #$02     TIMEOUT ?
. 912b  f0 06    beq $9133    Nein, dann $9133
. 912d  20 fe 90 jsr $90fe    Ein-/Ausgabe-Kanal schließen und Tastatur einsch.
. 9130  4c b0 8d jmp $8db0    Setzt Textpointer auf Trennzeichen

. 9133  ad 00 02 lda $0200    Holt 1. Zeichen im Puffer
. 9136  d0 1e    bne $9156    Keine Endemarke ?, dann $9156
. 9138  a5 13    lda $13      Holt Ein-/Ausgabe-Kanal
. 913a  d0 e3    bne $911f    Eingabegerät ungleich Tastatur ?, dann $911f
. 913c  20 be 8d jsr $8dbe    Sucht nächstes Trennzeichen
. 913f  4c b3 8d jmp $8db3    Setzt Textpointer auf nächstes Trennzeichen

. 9142  a5 13    lda $13      Holt Ein-/Ausgabe-Kanal
. 9144  d0 06    bne $914c    Eingabegerät ungleich Tastatur ?, dann $914c
. 9146  20 b0 90 jsr $90b0    Fragezeichen (?) ausgeben
. 9149  20 aa 90 jsr $90aa    'Cursor nach rechts' ausgeben
. 914c  4c 5a 88 jmp $885a    Sprung zur Eingabeschleife

. 914f  :
. 914f  *** R E A D ***
. 914f  :
. 914f  a6 41    ldx $41      Holt Adresse des
. 9151  a4 42    ldy $42      ... aktuellen Data-Elementes
. 9153  a9 98    lda #$98     Holt Flag-Wert für 'READ'

. 9155  2c       .byte $2c

. 9156  a9 00    lda #$00     Holt Flag-Wert für 'INPUT'
. 9158  85 11    sta $11      Im Eingabe-Flag speichern
. 915a  86 43    stx $43      Adresse in
. 915c  84 44    sty $44      ... Eingabezeiger schreiben
. 915e  20 a5 96 jsr $96a5    Variable suchen oder anlegen
```

```
. 9161  85 49     sta $49       Schreibt
. 9163  84 4a     sty $4a       ... Variablenzeiger
. 9165  a2 01     ldx #$01
. 9167  b5 3b     lda $3b,x     Textpointer
. 9169  95 4b     sta $4b,x     ... retten
. 916b  b5 43     lda $43,x     ... und
. 916d  95 3b     sta $3b,x     ... durch
. 916f  ca        dex           ... Eingabezeiger
. 9170  10 f5     bpl $9167     ... ersetzen
. 9172  20 79 04  jsr $0479     CHRGOT-Routine
. 9175  d0 31     bne $91a8     Keine Endmarke ?, dann $91a8
. 9177  24 11     bit $11       Holt Eingabe-Flag
. 9179  50 1a     bvc $9195     Ungleich 'GET' ?, dann $9195
. 917b  a5 80     lda $80       Holt letztes Token
. 917d  c9 f9     cmp #$f9      'KEY'-Token ?
. 917f  d0 08     bne $9189     Nein, dann $9189
. 9181  20 af a7  jsr $a7af     (GETIN)
. 9184  aa        tax           Tastencode eingegeben ?
. 9185  f0 fa     beq $9181     Nein, dann warten
. 9187  d0 03     bne $918c     Ja, dann $918c

. 9189  20 af a7  jsr $a7af     (GETIN)
. 918c  8d 00 02  sta $0200     Schreibt Code in Eingabepuffer
. 918f  a2 ff     ldx #$ff      Zeiger auf Komma
. 9191  a0 01     ldy #$01      ... vor den Eingabepuffer setzen
. 9193  d0 0f     bne $91a4     Springt immer nach $91a4

. 9195  10 03     bpl $919a     'INPUT'-Token ?, dann $919a
. 9197  4c 40 92  jmp $9240     Fortsetzung 'READ'

. 919a  a5 13     lda $13       Holt Ein-/Ausgabe-Kanal
. 919c  d0 03     bne $91a1     Eingabegerät nicht Tastatur ?, dann $91a1
. 919e  20 b0 90  jsr $90b0     Fragezeichen (?) ausgeben
. 91a1  20 42 91  jsr $9142     Fragezeichen und 'Cursor n. rechts'; Eingabe
. 91a4  86 3b     stx $3b       Textpointer vor
. 91a6  84 3c     sty $3c       ... Eingabepuffer setzen
. 91a8  20 73 04  jsr $0473     CHRGET-Routine
. 91ab  24 0d     bit $0d       Stringvariable ?
. 91ad  10 31     bpl $91e0     Nein, dann $91e0
. 91af  24 11     bit $11       Eingabeflag
. 91b1  50 09     bvc $91bc     Nicht 'GET', dann $91bc
. 91b3  e8        inx           Textpointer um
. 91b4  86 3b     stx $3b       ... 1 erhöhen
. 91b6  a9 00     lda #$00      Holt Endmarke
. 91b8  85 07     sta $07       Endmarke zwischenspeichern
. 91ba  f0 0c     beq $91c8     Springt immer nach $91c8

. 91bc  85 07     sta $07       Schreibt Anfangscode
. 91be  c9 22     cmp #$22      Gleich Anführungszeichen (") ?
. 91c0  f0 07     beq $91c9     Ja, dann $91c9
. 91c2  a9 3a     lda #$3a      Holt Doppelpunkt (:) als Endmarke
. 91c4  85 07     sta $07       Endmarke zwischenspeichern
```

```
. 91c6  a9 2c     lda #$2c      Holt Komma (,) als weiteres Abbruchkriterium
. 91c8  18        clc
. 91c9  85 08     sta $08
. 91cb  a5 3b     lda $3b       Holt
. 91cd  a4 3c     ldy $3c       ... Textpointer
. 91cf  69 00     adc #$00      Addiert Carry
. 91d1  90 01     bcc $91d4
. 91d3  c8        iny
. 91d4  20 7a 9b  jsr $9b7a     String übernehmen
. 91d7  20 c6 9d  jsr $9dc6     Textpointer hinter String setzen
. 91da  20 b1 8e  jsr $8eb1     Deskriptor in Variablentabelle schreiben
. 91dd  4c e8 91  jmp $91e8     Sprung nach $91e8

. 91e0  20 7f a3  jsr $a37f     String-Gleitkomma-Wandlung
. 91e3  a5 0e     lda $0e       Holt Integer-Flag
. 91e5  20 99 8e  jsr $8e99     FAC in Variable übertragen
. 91e8  20 79 04  jsr $0479     CHRGOT-Routine
. 91eb  f0 3b     beq $9228     Trennzeichen ?, dann $9228
. 91ed  c9 2c     cmp #$2c      Gleich Komma (,) ?
. 91ef  f0 37     beq $9228     Ja, dann $9228
. 91f1  a5 11     lda $11       Holt Eingabe-Flag
. 91f3  f0 0a     beq $91ff     Bei 'INPUT', nach $91ff
. 91f5  30 04     bmi $91fb     Bei 'READ', nach $91fb
. 91f7  a6 13     ldx $13       Holt Ein-/Ausgabe-Kanal
. 91f9  d0 08     bne $9203     Eingabegerät ungleich Tastatur ?, dann $9203
. 91fb  a2 16     ldx #$16      Holt Fehlernummer 22 ('TYPE MISMATCH')
. 91fd  d0 06     bne $9205     Springt immer nach $9205

. 91ff  a5 13     lda $13       Holt Ein-/Ausgabe-Kanal
. 9201  f0 05     beq $9208     Eingabegerät gleich Tastatur ?, dann $9208
. 9203  a2 18     ldx #$18      Holt Fehlernummer 24 ('FILE DATA')
. 9205  4c 83 86  jmp $8683     Fehlerausgabe

. 9208  20 4f ff  jsr $ff4f     Meldung ausgeben

. 920b            .byte '?REDO FROM START(CR)'
. 921c  00        .byte $00

. 921d  ad 5b 02  lda $025b     Holt Anfangsadresse
. 9220  ac 5c 02  ldy $025c     ... der Eingabeanweisung
. 9223  85 3b     sta $3b       Im Textpointer
. 9225  84 3c     sty $3c       ... ablegen
. 9227  60        rts           ENDE

. 9228  a2 01     ldx #$01      Textpointer
. 922a  b5 3b     lda $3b,x     ... in
. 922c  95 43     sta $43,x     ... Eingabezeiger
. 922e  b5 4b     lda $4b,x     ... bringen
. 9230  95 3b     sta $3b,x     ... und
. 9232  ca        dex           ... Ausgangswert
. 9233  10 f5     bpl $922a     ... wiederherstellen
. 9235  20 79 04  jsr $0479     CHRGOT-Routine
```

```
. 9238  f0 30    beq $926a    Trennzeichen ?, dann $926a
. 923a  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. 923d  4c 5e 91 jmp $915e    Nächste Eingabe

. 9240  20 be 8d jsr $8dbe    Y-Reg. gleich Offset zum Trennzeichen
. 9243  c8       iny
. 9244  aa       tax          Zeilenende ?
. 9245  d0 15    bne $925c    Nein, dann $925c
. 9247  a2 0d    ldx #$0d     Holt Fehlernummer 13 ('OUT OF DATA')
. 9249  c8       iny
. 924a  20 a5 04 jsr $04a5    Holt Linkadresse (High) (LDA($3b),Y)
. 924d  f0 6c    beq $92bb    Programm-Ende ?, dann Fehler
. 924f  c8       iny
. 9250  20 a5 04 jsr $04a5    Holt Zeilennummer (Low) (LDA($3b),Y)
. 9253  85 3f    sta $3f      Schreibt Zeilennr. (Low) d. akt. DATA-Zeile
. 9255  c8       iny
. 9256  20 a5 04 jsr $04a5    Holt Zeilennummer (High) (LDA($3b),Y)
. 9259  c8       iny
. 925a  85 40    sta $40      Schreibt Zeilennr. (High) d. akt. DATA-Zeile
. 925c  20 b3 8d jsr $8db3    Textpointer um Y-Reg. erhöhen
. 925f  20 79 04 jsr $0479    CHRGOT-Routine
. 9262  aa       tax
. 9263  e0 83    cpx #$83     'DATA'-Token ?
. 9265  d0 d9    bne $9240    Nein, dann weitersuchen
. 9267  4c a8 91 jmp $91a8    Dateneingabe fortsetzen

. 926a  a5 43    lda $43      Holt Eingabezeiger (Low)
. 926c  a4 44    ldy $44      Holt Eingabezeiger (High)
. 926e  a6 11    ldx $11      Holt Eingabe-Flag
. 9270  10 03    bpl $9275    Ungleich 'READ' ?, dann $9275
. 9272  4c bb 8c jmp $8cbb    Sprung nach $8cbb

. 9275  a0 00    ldy #$00
. 9277  20 55 81 jsr $8155    Holt nächstes Zeichen (LDA($43),Y)
. 927a  f0 17    beq $9293    Endmarke ?, dann $9293
. 927c  a5 13    lda $13      Holt Ein-/Ausgabe-Kanal
. 927e  d0 13    bne $9293    Eingabegerät ungleich Tastatur ?, dann $9293
. 9280  20 4f ff jsr $ff4f    Meldung ausgeben

. 9283           .byte '?extra ignored(CR)'
. 9292  00       .byte $00

. 9293  60       rts          ENDE

. 9294  :
. 9294  *** N E X T ***
. 9294  :
. 9294  d0 13    bne $92a9    Folgt Variablenname ?, dann $92a9
. 9296  a0 ff    ldy #$ff     Setzt Flag für 'Kein Variablenname'
. 9298  d0 14    bne $92ae    Springt immer nach $92ae

. 929a  a0 12    ldy #$12     Holt Anzahl der Bytes im Basic-Stack (bei 'FOR')
```

```
. 929c  20 72 a7 jsr $a772    Stackpointer um Y-Reg. erhöhen
. 929f  20 79 04 jsr $0479    CHRGOT-Routine
. 92a2  c9 2c    cmp #$2c     Folgt Komma (,) ?
. 92a4  d0 6d    bne $9313    Nein, dann $9313
. 92a6  20 73 04 jsr $0473    CHRGET-Routine
. 92a9  20 a5 96 jsr $96a5    Variable suchen oder anlegen
. 92ac  85 49    sta $49      Schreibt Variablenzeiger
. 92ae  84 4a    sty $4a      ... für FOR-NEXT
. 92b0  a0 81    ldy #$81     Holt 'FOR'-Token
. 92b2  84 02    sty $02      Zur Suche im Basic-Stack speichern
. 92b4  20 71 88 jsr $8871    Sucht Daten im Stack
. 92b7  f0 05    beq $92be    Gefunden ?, dann $92be
. 92b9  a2 0a    ldx #$0a     Holt Fehlernummer 10 ('NEXT WITHOUT FOR')
. 92bb  4c 83 86 jmp $8683    Fehlerausgabe

. 92be  20 69 a7 jsr $a769    Stackzeiger einrichten
. 92c1  a5 3d    lda $3d      Akku
. 92c3  18       clc          ... und
. 92c4  69 03    adc #$03     ... Y-Reg.
. 92c6  a4 3e    ldy $3e      ... auf STEP-Wert
. 92c8  90 01    bcc $92cb    ... richten
. 92ca  c8       iny
. 92cb  20 1f a2 jsr $a21f    Holt STEP-Wert nach FAC
. 92ce  a0 08    ldy #$08
. 92d0  b1 3d    lda ($3d),y  Holt Vorzeichen
. 92d2  85 66    sta $66      In Vorzeichenbyte von FAC ablegen
. 92d4  a0 01    ldy #$01
. 92d6  b1 3d    lda ($3d),y  Setzt
. 92d8  48       pha          ... (Akku,
. 92d9  aa       tax          ... Y-Reg.)
. 92da  c8       iny          ... gleich
. 92db  b1 3d    lda ($3d),y  ... FOR-NEXT-
. 92dd  48       pha          ... Variablenzeiger
. 92de  a8       tay
. 92df  8a       txa
. 92e0  20 9b 9e jsr $9e9b    Addiert Variablenwert zu FAC
. 92e3  68       pla          (X-Reg, Y-Reg.)
. 92e4  a8       tay          gleich
. 92e5  68       pla          FOR-Next-
. 92e6  aa       tax          Variablenzeiger
. 92e7  20 59 a2 jsr $a259    FAC in Basic-Stack übertragen
. 92ea  a5 3d    lda $3d      (Akku,
. 92ec  18       clc          ... Y-Reg.)
. 92ed  69 09    adc #$09     ... auf
. 92ef  a4 3e    ldy $3e      ... Endwert
. 92f1  90 01    bcc $92f4    ... setzen
. 92f3  c8       iny
. 92f4  20 e0 a2 jsr $a2e0    Variablenwert mit Endwert vergleichen
. 92f7  a0 08    ldy #$08
. 92f9  38       sec
. 92fa  f1 3d    sbc ($3d),y  Endwert überschritten ?
. 92fc  f0 9c    beq $929a    Ja, dann $929a
```

```
. 92fe  aØ 11    ldy #$11
. 9300  b1 3d    lda ($3d),y  Kopiert
. 9302  85 3b    sta $3b      ... Schleifen-
. 9304  88       dey          ... Anfangsadresse
. 9305  b1 3d    lda ($3d),y  ... in
. 9307  85 3c    sta $3c      ... Textpointer
. 9309  88       dey
. 930a  b1 3d    lda ($3d),y  Schreibt
. 930c  85 3a    sta $3a      ... Anfangs-
. 930e  88       dey          ... Zeilennummer
. 930f  b1 3d    lda ($3d),y  ... als aktuelle
. 9311  85 39    sta $39      ... Zeilennummer
. 9313  6Ø       rts          ENDE

. 9314  :
. 9314  *** FRMNUM ***
. 9314  :
. 9314  2Ø 2c 93 jsr $932c    Ausdruck auswerten
. 9317  :
. 9317  *** CHKNUM ***
. 9317  :
. 9317  18       clc          Prüft ob
. 9318  9Ø Ø1    bcc $931b    ... numerisch
. 931a  :
. 931a  *** CHKSTR ***
. 931a  :
. 931a  38       sec          Prüft ob String-Flag
. 931b  24 Ød    bit $Ød      ... gesetzt ist
. 931d  3Ø Ø3    bmi $9322    Ja, dann $9322
. 931f  bØ Ø3    bcs $9324    Nein, dann $9324
. 9321  6Ø       rts          ENDE

. 9322  bØ fd    bcs $9321
. 9324  a2 16    ldx #$16     Holt Fehlernummer 22 ('TYPE MISMATCH')

. 9326  2c       .byte $2c

. 9327  a2 19    ldx #$19     Holt Fehlernummer 25 ('FORMULA TOO COMPLEX')
. 9329  4c 83 86 jmp $8683    Fehlerausgabe

. 932c  :
. 932c  *** FRMEVL ***
. 932c  :
. 932c  a6 3b    ldx $3b      Textpointer
. 932e  dØ Ø2    bne $9332    ... initialisieren
. 9330  c6 3c    dec $3c
. 9332  c6 3b    dec $3b
. 9334  a2 ØØ    ldx #$ØØ
. 9336  24 48    bit $48
. 9338  8a       txa
. 9339  48       pha
. 933a  ba       tsx
```

```
. 933b  eØ 28     cpx #$28      Genug Platz im Stack ?
. 933d  9Ø e8     bcc $9327     Nein, dann Fehler
. 933f  2Ø 14 94  jsr $9414     (EVAL) nächsten Term auswerten
. 9342  a9 ØØ     lda #$ØØ     Akku für Vergleichssymbole
. 9344  85 4d     sta $4d       Initialisieren
. 9346  2Ø 79 Ø4  jsr $Ø479     CHRGOT-Routine
. 9349  38        sec
. 934a  e9 b1     sbc #$b1      'Größer', 'Gleich' oder 'Kleiner'-Zeichen ?
. 934c  9Ø 17     bcc $9365
. 934e  c9 Ø3     cmp #$Ø3
. 9350  bØ 13     bcs $9365     Nein, dann $9365
. 9352  c9 Ø1     cmp #$Ø1      Akku für Vergleichssymbole
. 9354  2a        rol           ... nimmt für
. 9355  49 Ø1     eor #$Ø1      ... jeden Vergleich
. 9357  45 4d     eor $4d       ... folgende Werte an:
. 9359  c5 4d     cmp $4d       ... Größer      =1, Gleich       =2, Kleiner =3
. 935b  9Ø 61     bcc $93be     ... Größergleich=4, Kleinergleich=5, Ungleich=6
. 935d  85 4d     sta $4d       ... sonst Ø
. 935f  2Ø 73 Ø4  jsr $Ø473     CHRGET-Routine
. 9362  4c 49 93  jmp $9349     Sprung nach $9349
. 9365  a6 4d     ldx $4d       Akku für Vergleichssymbole gleich Ø ?
. 9367  dØ 2c     bne $9395     Nein, dann $9395
. 9369  bØ 7e     bcs $93e9     Symbol gleich '+', '-', '*', '/', '', and, or ?
. 936b  69 Ø7     adc #$Ø7
. 936d  9Ø 7a     bcc $93e9     Nein, dann $93e9
. 936f  65 Ød     adc $Ød       String ?
. 9371  dØ Ø3     bne $9376     Nein, dann $9376
. 9373  4c da 9b  jmp $9bda     String-Verkettung

. 9376  69 ff     adc #$ff      Wiederherstellung
. 9378  85 22     sta $22       ... des Token
. 937a  Øa        asl           ... und
. 937b  65 22     adc $22       ... Offset-Zeiger
. 937d  a8        tay           ... erzeugen
. 937e  68        pla           Oberster Wert im Stack
. 937f  d9 53 84  cmp $8453,y   ... kleiner PRIO-Flag der Operation ?
. 9382  bØ 6a     bcs $93ee     Nein, dann $93ee
. 9384  2Ø 17 93  jsr $9317     Nicht numerisch ?, dann Fehler
. 9387  48        pha
. 9388  2Ø ae 93  jsr $93ae     Aufruf der Routine
. 938b  68        pla
. 938c  a4 4b     ldy $4b
. 938e  1Ø 17     bpl $93a7
. 9390  aa        tax
. 9391  fØ 59     beq $93ec
. 9393  dØ 62     bne $93f7

. 9395  46 Ød     lsr $Ød       Setzt String-Flag zurück
. 9397  8a        txa
. 9398  2a        rol
. 9399  a6 3b     ldx $3b       Textpointer
. 939b  dØ Ø2     bne $939f     ... um
```

```
. 939d  c6 3c    dec $3c       ... 1 Schritt
. 939f  c6 3b    dec $3b       ... zurücksetzen
. 93a1  a0 1b    ldy #$1b      Schreibt Offset der PRIO-Flags
. 93a3  85 4d    sta $4d       ... von 'Kleiner', 'Gleich' oder 'Größer'
. 93a5  d0 d7    bne $937e     Springt immer nach $937e

. 93a7  d9 53 84 cmp $8453,y   Vergleich
. 93aa  b0 4b    bcs $93f7     ... mit
. 93ac  90 d9    bcc $9387     ... PRIO-Flag

. 93ae  b9 55 84 lda $8455,y   Holt Routinenadresse (High)
. 93b1  48       pha           Auf Stack
. 93b2  b9 54 84 lda $8454,y   Holt Routinenadresse (Low)
. 93b5  48       pha           Auf Stack
. 93b6  20 c1 93 jsr $93c1     Schiebt Operanden auf Stack
. 93b9  a5 4d    lda $4d       Holt Akku für Vergleichssymbole
. 93bb  4c 37 93 jmp $9337     Zurück zum Anfang

. 93be  4c a1 94 jmp $94a1     'SYNTAX ERROR' ausgeben

. 93c1  a5 66    lda $66       Holt Vorzeichen von FAC
. 93c3  be 53 84 ldx $8453,y   Holt PRIO-Flag
. 93c6  a8       tay
. 93c7  18       clc
. 93c8  68       pla           Rücksprungadresse
. 93c9  69 01    adc #$01      ... in
. 93cb  85 22    sta $22       ... Hilfszeiger
. 93cd  68       pla           ... zwischenspeichern
. 93ce  69 00    adc #$00
. 93d0  85 23    sta $23
. 93d2  98       tya
. 93d3  48       pha
. 93d4  20 a0 a2 jsr $a2a0     FAC runden
. 93d7  a5 65    lda $65       Fac
. 93d9  48       pha           ... auf
. 93da  a5 64    lda $64       ... Stack
. 93dc  48       pha           ... legen
. 93dd  a5 63    lda $63
. 93df  48       pha
. 93e0  a5 62    lda $62
. 93e2  48       pha
. 93e3  a5 61    lda $61
. 93e5  48       pha
. 93e6  6c 22 00 jmp ($0022)   Rücksprung (Adr. in Hilfszeiger) durchführen

. 93e9  a0 ff    ldy #$ff
. 93eb  68       pla
. 93ec  f0 23    beq $9411
. 93ee  c9 64    cmp #$64
. 93f0  f0 03    beq $93f5
. 93f2  20 17 93 jsr $9317     Nicht numerisch ?, dann Fehler
. 93f5  84 4b    sty $4b
```

```
. 93f7  68       pla          ARG
. 93f8  4a       lsr          ... von
. 93f9  85 12    sta $12      ... Stack
. 93fb  68       pla          ... holen...
. 93fc  85 69    sta $69
. 93fe  68       pla
. 93ff  85 6a    sta $6a
. 9401  68       pla
. 9402  85 6b    sta $6b
. 9404  68       pla
. 9405  85 6c    sta $6c
. 9407  68       pla
. 9408  85 6d    sta $6d
. 940a  68       pla
. 940b  85 6e    sta $6e
. 940d  45 66    eor $66
. 940f  85 6f    sta $6f      Vorzeichen von ARG multipliziert mit FAC
. 9411  a5 61    lda $61      Holt Exponent von FAC (als Null-Flag)
. 9413  60       rts          ENDE

. 9414  :
. 9414  *** EVAL ***
. 9414  :
. 9414  6c 0a 03 jmp ($030a)  Eval ($9417)

. 9417  a9 00    lda #$00     Löscht
. 9419  85 0d    sta $0d      ... String-Flag
. 941b  20 73 04 jsr $0473    CHRGET-Routine
. 941e  b0 03    bcs $9423    Keine Ziffer ?, dann $9423
. 9420  4c 7f a3 jmp $a37f    Schreibt Zahl nach FAC

. 9423  20 3a 97 jsr $973a    Zeichen gleich Buchstabe ?
. 9426  90 03    bcc $942b    Nein, dann $942b
. 9428  4c ad 94 jmp $94ad    Variable auswerten

. 942b  c9 ff    cmp #$ff     Zeichen gleich PI ?
. 942d  d0 0f    bne $943e    Nein, dann $943e
. 942f  a9 39    lda #$39     Adresse $9439 (gleich Adresse von PI)
. 9431  a0 94    ldy #$94     ... nach Akku,Y-Reg.
. 9433  20 21 a2 jsr $a221    Schreibt Konstante PI nach FAC
. 9436  4c 73 04 jmp $0473    CHRGET-Routine

. 9439  .byte $82,$49,$0f,$da,a1    Konstante PI

. 943e  c9 2e    cmp #$2e     Zeichen gleich Dezimalpunkt ?
. 9440  f0 de    beq $9420    Ja, dann $9420
. 9442  c9 ab    cmp #$ab     Zeichen gleich '-'-Token ?
. 9444  f0 60    beq $94a6    Ja, dann $94a6
. 9446  c9 aa    cmp #$aa     Zeichen gleich '+'-Token ?
. 9448  f0 d1    beq $941b    Ja, dann $941b
. 944a  c9 22    cmp #$22     Zeichen gleich Anführungszeichen (") ?
. 944c  d0 0f    bne $945d    Nein, dann $945d
```

```
. 944e  a5 3b     lda $3b      Holt
. 9450  a4 3c     ldy $3c      ... Textpointer
. 9452  69 00     adc #$00     Um
. 9454  90 01     bcc $9457    ... 1
. 9456  c8        iny          ... erhöhen
. 9457  20 74 9b  jsr $9b74    String übernehmen
. 945a  4c c6 9d  jmp $9dc6    Textpointer hinter String setzen

. 945d  c9 a8     cmp #$a8     Zeichen gleich 'NOT'-Token ?
. 945f  d0 16     bne $9477    Nein, dann $9477
. 9461  a0 18     ldy #$18     Holt Offset von PRIO-Flag
. 9463  d0 43     bne $94a8    Springt immer nach $94a8

. 9465  :
. 9465  *** N O T ***
. 9465  :
. 9465  20 86 98  jsr $9886    Integer-Gleitkomma-Wandlung
. 9468  a5 65     lda $65      Integer-Zahl
. 946a  49 ff     eor #$ff     ... in
. 946c  a8        tay          ... FAC
. 946d  a5 64     lda $64      ... invertieren
. 946f  49 ff     eor #$ff
. 9471  20 92 9a  jsr $9a92    FAC vorbereiten (A,Y) als Integer nach FAC
. 9474  4c c9 a2  jmp $a2c9    Integer-Gleitkomma-Wandlung

. 9477  c9 a5     cmp #$a5     Zeichen gleich 'FN'-Token
. 9479  d0 03     bne $947e    Nein, dann $947e
. 947b  4c de 9a  jmp $9ade    FN

. 947e  c9 b4     cmp #$b4     Zeichen gleich Funktionstoken ?
. 9480  90 03     bcc $9485    Nein, dann $9485
. 9482  4c 99 95  jmp $9599    Funktion auswerten

. 9485  :
. 9485  *** KLAMMERTERM AUSWERTEN ***
. 9485  :
. 9485  20 8e 94  jsr $948e    Prüft auf 'Klammer auf'
. 9488  20 2c 93  jsr $932c    (FRMEVL) Ausdruck auswerten
. 948b  a9 29     lda #$29     Holt 'Klammer zu'

. 948d  2c        .byte $2c

. 948e  a9 28     lda #$28     Holt 'Klammer auf'

. 9490  2c        .byte $2c
. 9491  :
. 9491  *** CHKCOM ***
. 9491  :
. 9491  a9 2c     lda #$2c     Holt 'Komma'
. 9493  a0 00     ldy #$00
. 9495  85 78     sta $78      Zeichencode zwischenspeichern
. 9497  20 a5 04  jsr $04a5    Holt Zeile aus Programm (LDA($3b),Y)
```

```
. 949a  c5 78     cmp $78      Vergleich mit gespeichertem Zeichen-Code
. 949c  dØ Ø3     bne $94a1    Ungleich ?, dann Fehler
. 949e  4c 73 Ø4  jmp $Ø473    CHRGET-Routine
. 94a1  a2 Øb     ldx #$Øb     Holt Fehlernummer 11 ('SYNTAX')
. 94a3  4c 83 86  jmp $8683    Fehlerausgabe
. 94a6  aØ 15     ldy #$15     PRIO-Flag Offset für Zeichenwechsel
. 94a8  68        pla
. 94a9  68        pla
. 94aa  4c 88 93  jmp $9388    Zurück zur Auswertung

. 94ad  :
. 94ad  *** VARIABLE AUSWERTEN ***
. 94ad  :
. 94ad  2Ø a5 96  jsr $96a5    Variable suchen oder anlegen
. 94bØ  85 64     sta $64      Schreibt
. 94b2  84 65     sty $65      ... Variablenzeiger
. 94b4  a6 45     ldx $45      Holt Zeiger
. 94b6  a4 46     ldy $46      ... für Variablennamen
. 94b8  a5 Ød     lda $Ød      Prüft ob String
. 94ba  fØ 45     beq $95Ø1    Nein, dann $95Ø1
. 94bc  a9 ØØ     lda #$ØØ     Rundungsstelle
. 94be  85 7Ø     sta $7Ø      ... von FAC initialisieren
. 94cØ  eØ 54     cpx #$54     Variable
. 94c2  dØ 24     bne $94e8    ... TI$ ?
. 94c4  cØ c9     cpy #$c9
. 94c6  dØ 76     bne $953e    Nein, dann $953e
. 94c8  a5 64     lda $64      Zeigt
. 94ca  c9 a2     cmp #$a2     ... Variablenzeiger
. 94cc  dØ 7Ø     bne $953e    ... auf
. 94ce  a5 65     lda $65      ... Systemkostante Ø ?
. 94dØ  c9 Ø4     cmp #$Ø4
. 94d2  dØ 6a     bne $953e    Nein, dann $953e
. 94d4  2Ø 31 95  jsr $9531    (RDTIM) Zeit nach FAC
. 94d7  84 5e     sty $5e      Auf Ø setzen
. 94d9  88        dey
. 94da  84 71     sty $71      Auf 255 setzen
. 94dc  aØ Ø6     ldy #$Ø6
. 94de  84 5d     sty $5d      Auf 6 setzen
. 94eØ  aØ 24     ldy #$24     Offset in Umrechnungstabelle
. 94e2  2Ø fa a4  jsr $a4fa    Zeit in Stunden, Minuten und Sekunden umrechnen
. 94e5  4c 7Ø 9b  jmp $9b7Ø    STR$

. 94e8  eØ 44     cpx #$44     Variable
. 94ea  dØ 52     bne $953e    ... DS$ ?
. 94ec  cØ d3     cpy #$d3
. 94ee  dØ 4e     bne $953e    Nein, dann $953e
. 94fØ  2Ø fa 94  jsr $94fa    Wenn DS$ ungültig, dann neu einlesen
. 94f3  a5 7a     lda $7a      Holt Adresse
. 94f5  a4 7b     ldy $7b      ... von DS$
. 94f7  4c 74 9b  jmp $9b74    Übernimmt String

. 94fa  a5 79     lda $79      Holt Länge von DS$
```

```
. 94fc  dØ 4Ø     bne $953e    Ungleich Ø ?, dann DS$ gültig
. 94fe  4c cf cc  jmp $cccf    DS$ einlesen

. 95Ø1  24 Øe     bit $Øe      Integer ?
. 95Ø3  1Ø Øf     bpl $9514    Nein, dann $9514
. 95Ø5  aØ ØØ     ldy #$ØØ
. 95Ø7  2Ø dc Ø4  jsr $Ø4dc    Holt Integer (High) (LDA($64),Y)
. 95Øa  aa        tax
. 95Øb  c8        iny
. 95Øc  2Ø dc Ø4  jsr $Ø4dc    Holt Integer (Low) (LDA($64),Y)
. 95Øf  a8        tay
. 9510  8a        txa
. 9511  4c 71 94  jmp $9471    Integer-Gleitkomma-Wandlung
. 9514  a5 65     lda $65      Zeigt
. 9516  c9 Ø4     cmp #$Ø4     ... Variablen-
. 9518  dØ 78     bne $9592    ... zeiger
. 951a  a5 64     lda $64      ... auf
. 951c  c9 a2     cmp #$a2     ... Adresse $Ø4a2 ?
. 951e  dØ 72     bne $9592    Nein, dann $9592
. 9520  eØ 54     cpx #$54     Variable
. 9522  dØ 1b     bne $953f    ... TI$ ?
. 9524  cØ 49     cpy #$49
. 9526  dØ 6a     bne $9592    Nein, dann $9592
. 9528  2Ø 31 95  jsr $9531    Zeit nach FAC
. 952b  98        tya
. 952c  a2 aØ     ldx #$aØ     Holt Exponent
. 952e  4c d4 a2  jmp $a2d4    FAC normalisieren

. 9531  2Ø de ff  jsr $ffde    (RDTIM) Zeit nach (A,X,Y)
. 9534  86 64     stx $64      Zeit
. 9536  84 63     sty $63      ... in
. 9538  85 65     sta $65      ... FAC
. 953a  aØ ØØ     ldy #$ØØ     ... übertragen
. 953c  84 62     sty $62
. 953e  6Ø        rts          ENDE

. 953f  eØ 53     cpx #$53     Variable
. 9541  dØ Øa     bne $954d    ... ST ?
. 9543  cØ 54     cpy #$54
. 9545  dØ 4b     bne $9592    Nein, dann $9592
. 9547  2Ø b7 ff  jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. 954a  4c c1 a2  jmp $a2c1    Byte-Gleitkomma-Wandlung

. 954d  eØ 44     cpx #$44     Variable
. 954f  dØ 26     bne $9577    ... DS ?
. 9551  cØ 53     cpy #$53
. 9553  dØ 3d     bne $9592    Nein, dann $9592
. 9555  2Ø fa 94  jsr $94fa    DS$ einlesen, wenn ungültig
. 9558  aØ ØØ     ldy #$ØØ
. 955a  a9 7a     lda #$7a     Akku auf DS$
. 955c  2Ø 94 Ø4  jsr $Ø494    Holt 1. Fehlerbyte (LDA(A),Y)
. 955f  29 Øf     and #$Øf     Isoliert unteres Halbbyte
```

```
. 9561  0a       asl          Multipliziert
. 9562  85 0f    sta $0f      ... mit
. 9564  0a       asl          ... 2
. 9565  0a       asl          1. Fehlerbyte * 8
. 9566  65 0f    adc $0f      Addiert mit 1. Fehlerbyte * 2
. 9568  85 0f    sta $0f      Ergibt Zehnerstelle von DS
. 956a  c8       iny
. 956b  a9 7a    lda #$7a
. 956d  20 94 04 jsr $0494    Holt 2. Fehlerbyte (LDA(A),Y)
. 9570  29 0f    and #$0f     Isoliert unteres Halbbyte
. 9572  65 0f    adc $0f      Addiert mit Zehnerstelle ergibt DS
. 9574  4c c1 a2 jmp $a2c1    Byte-Gleitkomma-Wandlung

. 9577  e0 45    cpx #$45     Enthält Var. den Buchstaben 'e' für ER oder EL
. 9579  d0 17    bne $9592    Nein, dann $9592
. 957b  c0 52    cpy #$52     Enthält Var. den Buchstaben 'r' ?
. 957d  f0 0d    beq $958c    Ja, dann $958c
. 957f  c0 4c    cpy #$4c     Enthält Var. den Buchstaben 'l' ?
. 9581  d0 0f    bne $9592    Nein, dann $9592
. 9583  ad f1 04 lda $04f1    Holt Zeilennummer
. 9586  ac f0 04 ldy $04f0    ... der Fehlermeldung
. 9589  4c 76 9a jmp $9a76    Integer-Gleitkomma-Wandlung

. 958c  ad ef 04 lda $04ef    Trap-Flag
. 958f  4c c1 a2 jmp $a2c1    Byte-Gleitkomma-Wandlung

. 9592  a5 64    lda $64      Holt
. 9594  a4 65    ldy $65      ... Variablenzeiger
. 9596  4c 1f a2 jmp $a21f    Wert der Variablen nach FAC

. 9599  :
. 9599  *** FUNKTION AUSWERTEN ***
. 9599  :
. 9599  c9 d5    cmp #$d5     Token größer 212 ($d4) ?
. 959b  b0 58    bcs $95f5    Ja, dann keine Funktion
. 959d  c9 cb    cmp #$cb     Token ungleich 203 ? ($cb)
. 959f  90 02    bcc $95a3    Ja, dann $95a3
. 95a1  e9 01    sbc #$01     GO-Token (203) auslassen
. 95a3  48       pha
. 95a4  aa       tax
. 95a5  20 73 04 jsr $0473    CHRGET-Routine
. 95a8  e0 d3    cpx #$d3     'INSTR'-Token ?
. 95aa  f0 08    beq $95b4    Ja, dann $95b4
. 95ac  e0 cb    cpx #$cb     'LEFT$'-
. 95ae  b0 29    bcs $95d9    ... 'MID$'-
. 95b0  e0 c8    cpx #$c8     ... oder 'RIGHT$'-Token ?
. 95b2  90 25    bcc $95d9    Nein, dann $95d9
. 95b4  20 8e 94 jsr $948e    Prüft ob 'Klammer auf' folgt
. 95b7  20 2c 93 jsr $932c    (FRMEVL) Ausdruck auswerten
. 95ba  20 91 94 jsr $9491    (CHKOM) Prüft ob Komma folgt
. 95bd  20 1a 93 jsr $931a    (CHKSTR) Prüft auf String
. 95c0  68       pla
```

```
. 95c1  c9 d3     cmp #$d3      'INSTR'-Token ?
. 95c3  f0 2d     beq $95f2     Ja, dann $95f2
. 95c5  aa        tax
. 95c6  a5 65     lda $65       Holt Zeiger auf String-Deskriptor (High)
. 95c8  48        pha           Auf Stack legen
. 95c9  a5 64     lda $64       Holt Zeiger auf String-Deskriptor (Low)
. 95cb  48        pha           Auf Stack legen
. 95cc  8a        txa           Token auf
. 95cd  48        pha           ... Stack legen
. 95ce  20 84 9d  jsr $9d84     (GETBYT) Parameter holen
. 95d1  68        pla
. 95d2  a8        tay
. 95d3  8a        txa
. 95d4  48        pha
. 95d5  98        tya
. 95d6  4c dd 95  jmp $95dd     Ruft Funktionsroutine auf
. 95d9  20 85 94  jsr $9485     Klammerterm auswerten
. 95dc  68        pla
. 95dd  38        sec
. 95de  e9 b4     sbc #$b4      Bildet
. 95e0  0a        asl           ... aus Token
. 95e1  a8        tay           ... Tabellenoffset
. 95e2  b9 16 84  lda $8416,y   Holt Funktionsroutine (High)
. 95e5  85 56     sta $56       Schreibt in Sprungvektor für Funktionen
. 95e7  b9 15 84  lda $8415,y   Holt Funktionsroutine (Low)
. 95ea  85 55     sta $55       Schreibt in Sprungvektor für Funktionen
. 95ec  20 54 00  jsr $0054     Funktionsaufruf
. 95ef  4c 17 93  jmp $9317     Fehler, wenn nicht numerisch

. 95f2  4c 86 b3  jmp $b386     INSTR

. 95f5  4c a1 94  jmp $94a1     'SYNTAX ERROR' ausgeben

. 95f8  :
. 95f8  *** O R ***
. 95f8  :
. 95f8  a0 ff     ldy #$ff
. 95fa  2c        .byte $2c
. 95fb  :
. 95fb  *** A N D ***
. 95fb  :
. 95fb  a0 00     ldy #$00      Eingabepuffer
. 95fd  84 0b     sty $0b       ... initialisieren
. 95ff  20 86 98  jsr $9886     Gleitkomma-Integer-Wandlung
. 9602  a5 64     lda $64
. 9604  45 0b     eor $0b
. 9606  85 07     sta $07
. 9608  a5 65     lda $65
. 960a  45 0b     eor $0b
. 960c  85 08     sta $08
. 960e  20 81 a2  jsr $a281     Schreibt ARG nach FAC
. 9611  20 86 98  jsr $9886     Gleitkomma-Integer-Wandlung
```

```
. 9614  a5 65     lda $65
. 9616  45 Øb     eor $Øb
. 9618  25 Ø8     and $Ø8
. 961a  45 Øb     eor $Øb
. 961c  a8        tay
. 961d  a5 64     lda $64
. 961f  45 Øb     eor $Øb
. 9621  25 Ø7     and $Ø7
. 9623  45 Øb     eor $Øb
. 9625  4c 71 94  jmp $9471    Integer-Gleitkomma-Wandlung
. 9628  :
. 9628  *** ZAHLENVERGLEICH ***
. 9628  :
. 9628  2Ø 1b 93  jsr $931b    Prüft Variablentyp
. 962b  bØ 13     bcs $964Ø    String ?, dann $964Ø
. 962d  a5 6e     lda $6e      Holt
. 962f  Ø9 7f     ora #$7f     ... Vorzeichenbit
. 9631  25 6a     and $6a      Schreibt Vorzeichenbit
. 9633  85 6a     sta $6a      ... ins 1. Mantissenbyte von FAC
. 9635  a9 69     lda #$69     Holt Adresse
. 9637  aØ ØØ     ldy #$ØØ     ... von ARG
. 9639  2Ø eØ a2  jsr $a2eØ    Vergleicht ARG mit FAC
. 963c  aa        tax
. 963d  4c 73 96  jmp $9673    Bringt Wahrheitswert nach FAC

. 964Ø  :
. 964Ø  *** STRINGVERGLEICH ***
. 964Ø  :
. 964Ø  a9 ØØ     lda #$ØØ     String-Flag
. 9642  85 Ød     sta $Ød      ... initialisieren
. 9644  c6 4d     dec $4d      Akku für Vergleichssymbole um 1 veringern
. 9646  2Ø 4e 9c  jsr $9c4e    (FRESTR)
. 9649  85 61     sta $61      Schreibt Länge des 1. Strings
. 964b  86 62     stx $62      Schreibt
. 964d  84 63     sty $63      ... Stringadresse
. 964f  a5 6c     lda $6c      Holt Zeiger
. 9651  a4 6d     ldy $6d      ... auf 2. String
. 9653  2Ø 52 9c  jsr $9c52    (FRESTR)
. 9656  86 6c     stx $6c      Schreibt Adresse
. 9658  84 6d     sty $6d      ... des 2. Strings
. 965a  aa        tax
. 965b  38        sec
. 965c  e5 61     sbc $61      Längenvergleich beider Strings
. 965e  fØ Ø8     beq $9668    Gleichlang ?, dann $9668
. 966Ø  a9 Ø1     lda #$Ø1
. 9662  9Ø Ø4     bcc $9668    2. String kürzer ?, dann $9668
. 9664  a6 61     ldx $61      Holt Länge des 1. Strings
. 9666  a9 ff     lda #$ff
. 9668  85 66     sta $66      Schreibt Vorzeichenbyte von FAC
. 966a  aØ ff     ldy #$ff
. 966c  e8        inx
. 966d  c8        iny
```

```
. 966e  ca       dex          Zeichenweiser
. 966f  d0 07    bne $9678    ... Vergleich
. 9671  a6 66    ldx $66      ... beider
. 9673  30 17    bmi $968c    ... Strings
. 9675  18       clc
. 9676  90 14    bcc $968c
. 9678  20 85 81 jsr $8185    Holt Zeichen aus 2. String (LDA($6c),Y)
. 967b  48       pha
. 967c  20 7d 81 jsr $817d    Holt Zeichen aus 1. String (LDA($62),Y)
. 967f  85 78    sta $78
. 9681  68       pla
. 9682  c5 78    cmp $78      Vergleich beider Zeichen
. 9684  f0 e7    beq $966d    Gleich ?, dann weiter vergleichen
. 9686  a2 ff    ldx #$ff     Ende des Vergleichs
. 9688  b0 02    bcs $968c
. 968a  a2 01    ldx #$01
. 968c  e8       inx
. 968d  8a       txa
. 968e  2a       rol
. 968f  25 12    and $12
. 9691  f0 02    beq $9695
. 9693  a9 ff    lda #$ff
. 9695  4c c1 a2 jmp $a2c1    Byte-Gleitkomma-Wandlung

. 9698  :
. 9698  *** VARIABLEN-VERWALTUNG ***
. 9698  :
. 9698  20 91 94 jsr $9491    (CHKCOM) Prüft auf Komma
. 969b  :
. 969b  *** D I M ***
. 969b  :
. 969b  aa       tax
. 969c  20 aa 96 jsr $96aa    Dimensioniert Variable
. 969f  20 79 04 jsr $0479    CHRGOT-Routine
. 96a2  d0 f4    bne $9698
. 96a4  60       rts          ENDE

. 96a5  :
. 96a5  *** VARIABLE SUCHEN ODER ANLEGEN ***
. 96a5  :
. 96a5  a2 00    ldx #$00
. 96a7  20 79 04 jsr $0479    CHRGOT-Routine
. 96aa  86 0c    stx $0c      Setzt DIM-Flag zurück
. 96ac  85 45    sta $45      Schreibt 1. Byte des Variablennamens
. 96ae  20 79 04 jsr $0479    CHRGOT-Routine
. 96b1  20 3a 97 jsr $973a    Überprüft ob Buchstabe
. 96b4  b0 03    bcs $96b9    Buchstabe ?, dann $96b9
. 96b6  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben
. 96b9  a2 00    ldx #$00
. 96bb  86 0d    stx $0d      String-Flag initialisieren
. 96bd  86 0e    stx $0e      Integer-Flag initialisieren
. 96bf  20 73 04 jsr $0473    CHRGET-Routine
```

```
. 96c2  90 05    bcc $96c9    Ziffer ?, dann $96c9
. 96c4  20 3a 97 jsr $973a    Überprüft ob Buchstabe
. 96c7  90 0b    bcc $96d4    Kein Buchstabe ?, dann $96d4
. 96c9  aa       tax
. 96ca  20 73 04 jsr $0473    CHRGET-Routine
. 96cd  90 fb    bcc $96ca    Ziffer ?, dann $96ca
. 96cf  20 3a 97 jsr $973a    Überprüft ob Buchstabe
. 96d2  b0 f6    bcs $96ca    Buchstabe ?, dann $96ca
. 96d4  c9 24    cmp #$24     Folgt '$'-Zeichen (für String-Variable) ?
. 96d6  d0 06    bne $96de    Nein, dann $96de
. 96d8  a9 ff    lda #$ff     Setzt
. 96da  85 0d    sta $0d      ... String-Flag
. 96dc  d0 10    bne $96ee    Springt immer nach $96ee

. 96de  c9 25    cmp #$25     Folgt '%'-Zeichen (für Integer-Variable) ?
. 96e0  d0 13    bne $96f5    Nein, dann $96f5
. 96e2  a5 10    lda $10      Ist Integer zulässig ?
. 96e4  d0 d0    bne $96b6    Nein, dann 'SYNTAX ERROR' ausgeben
. 96e6  a9 80    lda #$80     Setzt
. 96e8  85 0e    sta $0e      ... Integer-Flag
. 96ea  05 45    ora $45      Setzt Bit 7
. 96ec  85 45    sta $45      ... im 1. Namensbyte
. 96ee  8a       txa          Setzt
. 96ef  09 80    ora #$80     ... Bit 7 im
. 96f1  aa       tax          ... 2. Namensbyte
. 96f2  20 73 04 jsr $0473    CHRGET-Routine
. 96f5  86 46    stx $46      Schreibt 2. Byte des Variablennamens
. 96f7  38       sec
. 96f8  05 10    ora $10      Integer-Array-Sperrflag
. 96fa  e9 28    sbc #$28     Feldvariable ?
. 96fc  d0 03    bne $9701    Nein, dann $9701
. 96fe  4c 9b 98 jmp $989b    Feldvariable behandeln

. 9701  a0 00    ldy #$00     Integer-Array-Sperrflag
. 9703  84 10    sty $10      ... initialisieren
. 9705  a5 2d    lda $2d      Holt Variablenanfang (Low)
. 9707  a6 2e    ldx $2e      Holt Variablenanfang (High)
. 9709  86 60    stx $60      Schreibt Variablenanfang
. 970b  85 5f    sta $5f      ... in Suchzeiger
. 970d  e4 30    cpx $30      Beginn
. 970f  d0 04    bne $9715    ... der Arrays
. 9711  c5 2f    cmp $2f      ... erreicht ?
. 9713  f0 2f    beq $9744    Ja, dann $9744
. 9715  20 d1 04 jsr $04d1    Holt 1. Namensbyte (LDA($5f),Y)
. 9718  85 78    sta $78      1. Namensbyte zwischenspeichern
. 971a  a5 45    lda $45      Mit aktuellem
. 971c  c5 78    cmp $78      ... Variablennamen vergleichen
. 971e  d0 10    bne $9730    Ungleich ?, dann $9730
. 9720  c8       iny
. 9721  20 d1 04 jsr $04d1    Holt 2. Namensbyte (LDA($5f),Y)
. 9724  85 78    sta $78      2. Namensbyte zwischenspeichern
. 9726  a5 46    lda $46      Mit aktuellem
```

```
. 9728  c5 78     cmp $78       ... Variablennamen vergleichen
. 972a  dØ Ø3     bne $972f     Ungleich ?, dann $972f
. 972c  4c 4c 98  jmp $984c     Variable gefunden ! nach $984c

. 972f  88        dey
. 9730  18        clc
. 9731  a5 5f     lda $5f       Suchzeiger
. 9733  69 Ø7     adc #$Ø7      ... auf nächste
. 9735  9Ø d4     bcc $97Øb     ... Variable
. 9737  e8        inx           ... setzen
. 9738  dØ cf     bne $97Ø9     ... und weitersuchen

. 973a  c9 41     cmp #$41      Wenn Zeichen
. 973c  9Ø Ø5     bcc $9743     ... gleich Buchstabe
. 973e  e9 5b     sbc #$5b      ... dann Carry gleich 1
. 9740  38        sec           ... sonst
. 9741  e9 a5     sbc #$a5      ... Carry gleich Ø
. 9743  6Ø        rts           Zurück

. 9744  68        pla
. 9745  48        pha
. 9746  c9 af     cmp #$af      Kommt Aufruf von Ausdrucksauswertung ?
. 9748  dØ 2a     bne $9774     Nein, dann $9774
. 974a  a9 a2     lda #$a2      (Akku, Y-Reg.) zeigen
. 974c  aØ Ø4     ldy #$Ø4      ... auf Konstante Ø
. 974e  6Ø        rts           Zurück

. 974f  cØ c9     cpy #$c9      Byte 2 gleich 'I' ?
. 9751  fØ f7     beq $974a     Ja, dann TI$
. 9753  cØ 49     cpy #$49      Byte 2 gleich 'i' ?
. 9755  dØ 31     bne $9788     Nein, dann $9788
. 9757  fØ 18     beq $9771     Ja, dann TI

. 9759  cØ d3     cpy #$d3      Byte 2 gleich 'S' ?
. 975b  fØ 14     beq $9771     Ja, dann DS$
. 975d  cØ 53     cpy #$53      Byte 2 gleich 's' ?
. 975f  dØ 27     bne $9788     Nein, dann $9788
. 9761  fØ Øe     beq $9771     Ja, dann DS

. 9763  cØ 54     cpy #$54      Byte 2 gleich 't' ?
. 9765  dØ 21     bne $9788     Nein, dann $9788
. 9767  fØ Ø8     beq $9771     Ja, dann TS

. 9769  cØ 52     cpy #$52      Byte 2 gleich 'r' ?
. 976b  fØ Ø4     beq $9771     Ja, dann ER
. 976d  cØ 4c     cpy #$4c      Byte 2 gleich 'l' ?
. 976f  dØ 17     bne $9788     Nein, dann nicht EL !
. 9771  4c a1 94  jmp $94a1     'SYNTAX ERROR' ausgeben

. 9774  a5 45     lda $45       Holt 1. Namensbyte
. 9776  a4 46     ldy $46       Holt 2. Namensbyte
. 9778  c9 54     cmp #$54      Byte 1 gleich 't' ?
```

```
. 977a  f0 d3     beq $974f     Ja, dann TI oder TI$ ?
. 977c  c9 53     cmp #$53      Byte 1 gleich 's' ?
. 977e  f0 e3     beq $9763     Ja, dann TS ?
. 9780  c9 45     cmp #$45      Byte 1 gleich 'e' ?
. 9782  f0 e5     beq $9769     Ja, dann EL ?
. 9784  c9 44     cmp #$44      Byte 1 gleich 'd' ?
. 9786  f0 d1     beq $9759     Ja, dann DS oder DS$ ?
. 9788  a5 2f     lda $2f       Holt Beginn
. 978a  a4 30     ldy $30       ... der Arrays
. 978c  85 5f     sta $5f       Beginn der Arrays
. 978e  84 60     sty $60       ... zwischenspeichern
. 9790  a5 31     lda $31       Holt Ende
. 9792  a4 32     ldy $32       ... der Arrays
. 9794  85 5a     sta $5a       Ende der Arrays
. 9796  84 5b     sty $5b       ... zwischenspeichern
. 9798  18        clc
. 9799  69 07     adc #$07      Addiert 7
. 979b  90 01     bcc $979e     ... zum Ende
. 979d  c8        iny           ... der Arrays
. 979e  85 58     sta $58       Zwischen-
. 97a0  84 59     sty $59       ... speichern
. 97a2  20 c0 88  jsr $88c0     Block verschieben
. 97a5  a5 58     lda $58
. 97a7  a4 59     ldy $59
. 97a9  c8        iny
. 97aa  85 2f     sta $2f       Schreibt neuen Anfang
. 97ac  84 30     sty $30       ... der Arrays
. 97ae  85 58     sta $58       Neuer Anfang
. 97b0  84 59     sty $59       ... zwischenspeichern
. 97b2  a5 58     lda $58       Holt neuen Anfang
. 97b4  a6 59     ldx $59       ... nach Akku, X-Reg.
. 97b6  e4 32     cpx $32       Variablen-Ende
. 97b8  d0 06     bne $97c0     ... erreicht ?
. 97ba  c5 31     cmp $31       ... Nein,
. 97bc  d0 02     bne $97c0     ... dann $97c0
. 97be  f0 78     beq $9838     Ja, dann fertig

. 97c0  85 22     sta $22       Feldkopf (Akku, X-Reg.)
. 97c2  86 23     stx $23       ... in Hilfszeiger ablegen
. 97c4  a0 00     ldy #$00
. 97c6  20 b0 04  jsr $04b0     Holt 1. Namensbyte (LDA($22),Y)
. 97c9  aa        tax           Schreibt ins X-Reg.
. 97ca  c8        iny
. 97cb  20 b0 04  jsr $04b0     Holt 2. Namensbyte (LDA($22),Y)
. 97ce  08        php
. 97cf  c8        iny
. 97d0  20 b0 04  jsr $04b0     Holt Feldlänge (Low) (LDA($22),Y)
. 97d3  65 58     adc $58       Addiert Feldlänge (Low)
. 97d5  85 58     sta $58       ... zu $58
. 97d7  c8        iny
. 97d8  20 b0 04  jsr $04b0     Holt Feldlänge (High) (LDA($22),Y)
. 97db  65 59     adc $59       Addiert Feldlänge (High)
```

```
. 97dd  85 59     sta $59       ... zu $59
. 97df  28        plp           Gleitkomma-Variable ?
. 97eØ  1Ø dØ     bpl $97b2     Ja, dann $97b2
. 97e2  8a        txa           Integer-Variable ?
. 97e3  3Ø cd     bmi $97b2     Ja, dann $97b2
. 97e5  c8        iny
. 97e6  2Ø bØ Ø4  jsr $Ø4bØ     Holt DIM-Zahl (LDA($22),Y)
. 97e9  aØ ØØ     ldy #$ØØ
. 97eb  Øa        asl           Multipliziert mit 2
. 97ec  69 Ø5     adc #$Ø5      Addiert 5
. 97ee  65 22     adc $22       Addiert zum Hilfszeiger
. 97fØ  85 22     sta $22       Setzt Zeiger
. 97f2  9Ø Ø2     bcc $97f6     ... auf das
. 97f4  e6 23     inc $23       ... 1. Feldelement
. 97f6  a6 23     ldx $23       Prüft
. 97f8  e4 59     cpx $59       ... ob
. 97fa  dØ Ø4     bne $98ØØ     ... Feldende
. 97fc  c5 58     cmp $58       ... erreicht
. 97fe  fØ b6     beq $97b6     Ja, dann $97b6
. 98ØØ  aØ ØØ     ldy #$ØØ
. 98Ø2  2Ø bØ Ø4  jsr $Ø4bØ     Holt Länge des Strings (LDA($22),Y)
. 98Ø5  fØ 24     beq $982b     Null ?, dann $982b
. 98Ø7  85 78     sta $78       Zwischenspeichern
. 98Ø9  c8        iny
. 98Øa  2Ø bØ Ø4  jsr $Ø4bØ     Holt Stringadresse (Low) (LDA($22),Y)
. 98Ød  18        clc
. 98Øe  65 78     adc $78       Addiert Stringlänge
. 981Ø  85 5a     sta $5a       Gleich Adresse des R-Zeigers (Low)
. 9812  c8        iny
. 9813  2Ø bØ Ø4  jsr $Ø4bØ     Holt Stringadresse (High) (LDA($22),Y)
. 9816  69 ØØ     adc #$ØØ      Addiert Übertrag
. 9818  85 5b     sta $5b       Gleich Adresse des R-Zeigers (High)
. 981a  aØ ØØ     ldy #$ØØ
. 981c  2Ø 89 81  jsr $8189     Holt R-Zeiger (Low) (LDA($5a),Y)
. 981f  69 Ø7     adc #$Ø7      Addiert 7
. 9821  91 5a     sta ($5a),y   Gleich neuer R-Zeiger (Low)
. 9823  c8        iny
. 9824  2Ø 89 81  jsr $8189     Holt R-Zeiger (High) (LDA($5a),Y)
. 9827  69 ØØ     adc #$ØØ      Addiert Übertrag
. 9829  91 5a     sta ($5a),y   Gleich neuer R-Zeiger (High)
. 982b  a9 Ø3     lda #$Ø3      Zeiger in
. 982d  18        clc           ... Stringfeld
. 982e  65 22     adc $22       ... auf nächsten
. 983Ø  85 22     sta $22       ... Stringdeskriptor
. 9832  9Ø c2     bcc $97f6     ... erhöhen
. 9834  e6 23     inc $23
. 9836  dØ be     bne $97f6     Springt immer zurück

. 9838  aØ ØØ     ldy #$ØØ
. 983a  a5 45     lda $45       Holt 1. Namensbyte
. 983c  91 5f     sta ($5f),y   Schreibt in Variablenkopf
. 983e  c8        iny
```

```
. 983f  a5 46     lda $46       Holt 2. Namensbyte
. 9841  91 5f     sta ($5f),y   Schreibt in Variablenkopf
. 9843  a9 00     lda #$00
. 9845  c8        iny           Alle
. 9846  91 5f     sta ($5f),y   ... Variablen-Bytes
. 9848  c0 06     cpy #$06      ... auf 0
. 984a  d0 f9     bne $9845     ... setzen
. 984c  a5 5f     lda $5f       Setzt
. 984e  18        clc           ... Akku, Y-Reg.
. 984f  69 02     adc #$02      ... auf
. 9851  a4 60     ldy $60       ... Variablen-
. 9853  90 01     bcc $9856     ... Byte
. 9855  c8        iny
. 9856  85 47     sta $47       Schreibt Variablenzeiger (Low)
. 9858  84 48     sty $48       Schreibt Variablenzeiger (High)
. 985a  60        rts           Zurück

. 985b  a5 0b     lda $0b       Holt DIM-Zahl
. 985d  0a        asl           Multipliziert mit 2
. 985e  69 05     adc #$05      Addiert 5; gleich Länge des Variablenkopfes
. 9860  65 5f     adc $5f       Addiert
. 9862  a4 60     ldy $60       ... zum
. 9864  90 01     bcc $9867     ... Zeiger des
. 9866  c8        iny           ... Feldkopfes
. 9867  85 58     sta $58       Ergibt Zeiger
. 9869  84 59     sty $59       ... auf 1. Feldelement
. 986b  60        rts           ENDE

. 986c  .byte $90 $80 $00 $00 $00   Konstante 32768

. 9871  20 86 98  jsr $9886     Gleitkomma-Integer-Wandlung
. 9874  a5 64     lda $64       Holt Integer-Zahl (High)
. 9876  a4 65     ldy $65       Holt Integer-Zahl (Low)
. 9878  60        rts           ENDE

. 9879  20 73 04  jsr $0473     CHRGET-Routine
. 987c  20 2c 93  jsr $932c     (FRMEVL) Ausdruck auswerten
. 987f  20 17 93  jsr $9317     Fehler, wenn nicht numerisch
. 9882  a5 66     lda $66       Holt Vorzeichen von FAC
. 9884  30 0d     bmi $9893     Kleiner 0 ?, dann Fehler
. 9886  a5 61     lda $61       Holt Exponent von FAC
. 9888  c9 90     cmp #$90      Kleiner als $10 (16) ?
. 988a  90 0c     bcc $9898     Ja, dann $9898
. 988c  a9 6c     lda #$6c      Holt Konstante
. 988e  a0 98     ldy #$98      ... 32768 (Adr. $986c)
. 9890  20 e0 a2  jsr $a2e0     Vergleich mit FAC
. 9893  d0 03     bne $9898     Ungleich ?, dann Fehler
. 9895  4c 1c 99  jmp $991c     Gleitkomma-Integer-Wandlung

. 9898  4c 27 a3  jmp $a327     'ILLEGAL QUANTITY ERROR' ausgeben

. 989b  :
```

```
. 989b  *** ARRAY-VERWALTUNG ***
. 989b  :
. 989b  a5 0c     lda $0c       Holt DIM-Flag
. 989d  05 0e     ora $0e       Verknüpft mit INTEGER-Flag
. 989f  48        pha
. 98a0  a5 0d     lda $0d       Holt STRING-Flag
. 98a2  48        pha
. 98a3  a0 00     ldy #$00
. 98a5  98        tya
. 98a6  48        pha
. 98a7  a5 46     lda $46       Holt 2. Byte des Variablennamens
. 98a9  48        pha           Auf Stack
. 98aa  a5 45     lda $45       Holt 1. Byte des Variablennamens
. 98ac  48        pha           Auf Stack
. 98ad  20 79 98  jsr $9879     Index auswerten
. 98b0  68        pla           Variablennamen
. 98b1  85 45     sta $45       ... wieder-
. 98b3  68        pla           ... herstellen
. 98b4  85 46     sta $46
. 98b6  68        pla
. 98b7  a8        tay
. 98b8  ba        tsx
. 98b9  bd 02 01  lda $0102,x   Holt Flags
. 98bc  48        pha           ... vom Stack
. 98bd  bd 01 01  lda $0101,x   ... und legt diese
. 98c0  48        pha           ... wieder auf den Stack
. 98c1  a5 64     lda $64       Indexwert
. 98c3  9d 02 01  sta $0102,x   ... wird auf
. 98c6  a5 65     lda $65       ... alten Platz
. 98c8  9d 01 01  sta $0101,x   ... der Flags gelegt
. 98cb  c8        iny
. 98cc  84 0b     sty $0b
. 98ce  20 79 04  jsr $0479     CHRGOT-Routine
. 98d1  a4 0b     ldy $0b
. 98d3  c9 2c     cmp #$2c      Folgt Komma (,) ?
. 98d5  f0 ce     beq $98a5     Ja, dann nächsten Index auswerten
. 98d7  20 8b 94  jsr $948b     Prüft ob ')' folgt
. 98da  68        pla           Flags
. 98db  85 0d     sta $0d       ... wiederherstellen
. 98dd  68        pla
. 98de  85 0e     sta $0e
. 98e0  29 7f     and #$7f
. 98e2  85 0c     sta $0c
. 98e4  a6 2f     ldx $2f       Holt
. 98e6  a5 30     lda $30       ... Variablenanfang
. 98e8  86 5f     stx $5f       Schreibt diesen
. 98ea  85 60     sta $60       ... in Suchzeiger
. 98ec  c5 32     cmp $32       Variablen-
. 98ee  d0 04     bne $98f4     ... Ende
. 98f0  e4 31     cpx $31       ... erreicht ?
. 98f2  f0 46     beq $993a     Ja, dann Variable nicht gefunden
. 98f4  a0 00     ldy #$00
```

```
. 98f6  20 d1 04 jsr $04d1    Holt 1. Namensbyte aus Tabelle (LDA($5f),Y)
. 98f9  c8       iny
. 98fa  c5 45    cmp $45      Vergleicht mit gesuchtem Namen
. 98fc  d0 0b    bne $9909    Byte 1 ungleich ?, dann $9909
. 98fe  20 d1 04 jsr $04d1    Holt 2. Namensbyte aus Tabelle (LDA($5f),Y)
. 9901  85 78    sta $78
. 9903  a5 46    lda $46
. 9905  c5 78    cmp $78      Vergleicht mit gesuchtem Namen
. 9907  f0 18    beq $9921    Gleich ?, dann Variable gefunden
. 9909  c8       iny
. 990a  20 d1 04 jsr $04d1    Holt Feldlänge aus Header (LDA($5f),Y)
. 990d  18       clc          ... und
. 990e  65 5f    adc $5f      ... addiert
. 9910  aa       tax          ... Länge
. 9911  c8       iny          ... zu
. 9912  20 d1 04 jsr $04d1    ... Such-
. 9915  65 60    adc $60      ... String
. 9917  90 cf    bcc $98e8    Weitersuchen (immer)

. 9919  a2 12    ldx #$12     Holt Fehlernummer 18 ('BAD SUBSCRIPT')
. 991b  2c       .byte $2c
. 991c  a2 0e    ldx #$0e     Holt Fehlernummer 14 ('ILLEGAL QUANTITY')
. 991e  4c 83 86 jmp $8683    Fehlerausgabe

. 9921  a2 13    ldx #$13     Holt Fehlernummer 19 ('REDIM'D ARRAY')
. 9923  a5 0c    lda $0c      DIM-Flag gleich 0 ?
. 9925  d0 f7    bne $991e    Nein, dann Fehler
. 9927  20 5b 98 jsr $985b    Zeiger auf 1. Element setzen
. 992a  a0 04    ldy #$04
. 992c  20 d1 04 jsr $04d1    Holt DIM aus Header (LDA($5f),Y)
. 992f  85 78    sta $78      Zwischenspeichern
. 9931  a5 0b    lda $0b      Holt Zahl der Indizes
. 9933  c5 78    cmp $78      Gleich DIM ?
. 9935  d0 e2    bne $9919    Nein, dann Fehler
. 9937  4c c3 99 jmp $99c3    Feldelement aufsuchen

. 993a  20 5b 98 jsr $985b    Setzt Suchzeiger hinter Header
. 993d  20 23 89 jsr $8923    Prüft auf Platz im Speicher
. 9940  a0 00    ldy #$00
. 9942  84 72    sty $72
. 9944  a2 05    ldx #$05     5 Byte für Variablenlänge reservieren
. 9946  a5 45    lda $45      Holt 1. Byte des Variablennamens
. 9948  91 5f    sta ($5f),y  Schreibt in Header
. 994a  10 01    bpl $994d    Keine Integer-Variable ?, dann $994d
. 994c  ca       dex          Variablenlänge um 1 erniedrigen
. 994d  c8       iny
. 994e  a5 46    lda $46      Holt 2. Byte des Variablennamens
. 9950  91 5f    sta ($5f),y  Schreibt in Header
. 9952  10 02    bpl $9956    Keine String-Variable ?, dann $9956
. 9954  ca       dex
. 9955  ca       dex
. 9956  86 71    stx $71      Var.-Länge (Fließkomma=5, String=3, Integer=2)
```

```
. 9958  a5 0b     lda $0b      Holt Zahl der Dimensionen
. 995a  c8        iny
. 995b  c8        iny
. 995c  c8        iny
. 995d  91 5f     sta ($5f),y  Schreibt in Header
. 995f  a2 0b     ldx #$0b     Vorbelegung des DIM-Wertes
. 9961  a9 00     lda #$00
. 9963  24 0c     bit $0c      Aufruf durch DIM-Befehl ?
. 9965  50 08     bvc $996f    Nein, dann $996f
. 9967  68        pla          Holt DIM-Wert vom Stack
. 9968  18        clc
. 9969  69 01     adc #$01     Addiert 1
. 996b  aa        tax          ... wenn
. 996c  68        pla          ... DIM-Wert
. 996d  69 00     adc #$00     ... gleich 0
. 996f  c8        iny
. 9970  91 5f     sta ($5f),y  Schreibt
. 9972  c8        iny          ... DIM-
. 9973  8a        txa          ... Wert
. 9974  91 5f     sta ($5f),y  ... in Header
. 9976  20 2f 9a  jsr $9a2f    Berechnet Platz für Dimensionierung
. 9979  86 71     stx $71      Speichert
. 997b  85 72     sta $72      ... Platzbedarf
. 997d  a4 22     ldy $22      Holt Zeiger in Header
. 997f  c6 0b     dec $0b      Noch eine Dimension ?
. 9981  d0 dc     bne $995f    Ja, dann $995f
. 9983  65 59     adc $59      Addiert
. 9985  b0 67     bcs $99ee    ... Feldlänge
. 9987  85 59     sta $59      ... zur
. 9989  a8        tay          ... Feld-
. 998a  8a        txa          ... anfangs-
. 998b  65 58     adc $58      ... Adresse
. 998d  90 03     bcc $9992
. 998f  c8        iny
. 9990  f0 5c     beq $99ee    Wenn Überlauf, dann 'OUT OF MEMORY ERROR' ausg.
. 9992  20 23 89  jsr $8923    Prüft auf Platz im Speicher
. 9995  85 31     sta $31      Schreibt
. 9997  84 32     sty $32      ... Variablen-Ende
. 9999  a9 00     lda #$00     Alle
. 999b  e6 72     inc $72      ... Feldbytes
. 999d  a4 71     ldy $71      ... werden
. 999f  f0 05     beq $99a6    ... mit
. 99a1  88        dey          ... $00
. 99a2  91 58     sta ($58),y  ... gefüllt
. 99a4  d0 fb     bne $99a1
. 99a6  c6 59     dec $59
. 99a8  c6 72     dec $72
. 99aa  d0 f5     bne $99a1
. 99ac  e6 59     inc $59
. 99ae  38        sec
. 99af  a5 31     lda $31      Holt Variablen-
. 99b1  e5 5f     sbc $5f      ... Endadresse
```

```
. 99b3  a0 02    ldy #$02      ... und subtrahiert
. 99b5  91 5f    sta ($5f),y   ... Header-Anfangsadresse.
. 99b7  a5 32    lda $32       ... - Ergebnis gleich
. 99b9  c8       iny           ... Feldlänge.
. 99ba  e5 60    sbc $60       ... - Feldlänge in
. 99bc  91 5f    sta ($5f),y   ... Header schreiben
. 99be  a5 0c    lda $0c       Aufruf durch DIM-Befehl ?
. 99c0  d0 6c    bne $9a2e     Ja, dann fertig
. 99c2  c8       iny
. 99c3  20 d1 04 jsr $04d1     Holt DIM-Wert (LDA($5f),Y)
. 99c6  85 0b    sta $0b       An Header übergeben
. 99c8  a9 00    lda #$00      Multiplizier-
. 99ca  85 71    sta $71       ... Register
. 99cc  85 72    sta $72       ... initialisieren
. 99ce  c8       iny
. 99cf  68       pla           Holt Index von Stack
. 99d0  aa       tax
. 99d1  85 64    sta $64
. 99d3  20 d1 04 jsr $04d1     Holt DIM-Wert aus Header (LDA(5f),Y)
. 99d6  85 78    sta $78
. 99d8  68       pla
. 99d9  85 65    sta $65
. 99db  c5 78    cmp $78       Vergleichen
. 99dd  90 12    bcc $99f1     Kleiner ?, dann OK.
. 99df  d0 0a    bne $99eb     Größer ?, dann Fehler
. 99e1  c8       iny
. 99e2  20 d1 04 jsr $04d1     Indirekter Speicherzugriff (LDA($5f),Y)
. 99e5  85 78    sta $78
. 99e7  e4 78    cpx $78
. 99e9  90 07    bcc $99f2
. 99eb  4c 19 99 jmp $9919     'BAD SUBSCRIPT ERROR' ausgeben

. 99ee  4c 81 86 jmp $8681     'OUT OF MEMORY ERROR' ausgeben

. 99f1  :
. 99f1  *** FELDELEMENT SUCHEN ***
. 99f1  :
. 99f1  c8       iny
. 99f2  a5 72    lda $72       Holt
. 99f4  05 71    ora $71       ... Multiplizierregister
. 99f6  18       clc
. 99f7  f0 0a    beq $9a03     Wenn leer ?, dann $9a03
. 99f9  20 2f 9a jsr $9a2f     Multiplizierroutine
. 99fc  8a       txa
. 99fd  65 64    adc $64
. 99ff  aa       tax
. 9a00  98       tya
. 9a01  a4 22    ldy $22
. 9a03  65 65    adc $65
. 9a05  86 71    stx $71
. 9a07  c6 0b    dec $0b
. 9a09  d0 c1    bne $99cc
```

```
. 9aØb  85 72     sta $72
. 9aØd  a2 Ø5     ldx #$Ø5      5 Byte für Elementlänge reservieren
. 9aØf  a5 45     lda $45       Holt 1. Byte des Variablennamens
. 9a11  1Ø Ø1     bpl $9a14     Keine Integervariable ?, dann $9a14
. 9a13  ca        dex
. 9a14  a5 46     lda $46       Holt 2. Byte des Variablennamens
. 9a16  1Ø Ø2     bpl $9a1a     Keine Stringvariable ?, dann $9a1a
. 9a18  ca        dex
. 9a19  ca        dex
. 9a1a  86 28     stx $28       Variablenlänge (Fließkomma=5, String=3, Integer=2)
. 9a1c  a9 ØØ     lda #$ØØ
. 9a1e  2Ø 3a 9a  jsr $9a3a     Berechnet Offset des Feldelements
. 9a21  8a        txa
. 9a22  65 58     adc $58       Addiert Adresse des 1. Feldelements
. 9a24  85 47     sta $47       Gleich
. 9a26  98        tya           ... Adresse
. 9a27  65 59     adc $59       ... des
. 9a29  85 48     sta $48       ... gesuchten
. 9a2b  a8        tay           ... Elements
. 9a2c  a5 47     lda $47
. 9a2e  6Ø        rts           ENDE
. 9a2f  :
. 9a2f  *** MULTIPLIZIERROUTINE ***
. 9a2f  :
. 9a2f  84 22     sty $22       Y zwischenspeichern
. 9a31  2Ø d1 Ø4  jsr $Ø4d1     Holt Zeichen (LDA($5f),Y)
. 9a34  85 28     sta $28       Im Bereich für Multiplikation ablegen
. 9a36  88        dey
. 9a37  2Ø d1 Ø4  jsr $Ø4d1     Holt nächstes Zeichen (LDA($5f),Y)
. 9a3a  85 29     sta $29       Im Bereich für Multiplikation ablegen
. 9a3c  a9 1Ø     lda #$1Ø      Holt Verschiebe-Zähler
. 9a3e  85 5d     sta $5d       Im Bereich für numerische Operation ablegen
. 9a4Ø  a2 ØØ     ldx #$ØØ      X-Reg. initialisieren
. 9a42  aØ ØØ     ldy #$ØØ      Y-Reg. initialisieren
. 9a44  8a        txa           Aus Y-Reg. und X-Reg.
. 9a45  Øa        asl           ... und $71 und $72
. 9a46  aa        tax           ... wird je ein
. 9a47  98        tya           ... 16-Bit-Schieberegister gebildet
. 9a48  2a        rol           ... Anschließend wird $28, $29
. 9a49  a8        tay           ... mit $71, $72 multipliziert
. 9a4a  bØ a2     bcs $99ee     ... und in Y-Reg., X-Reg. abgelegt
. 9a4c  Ø6 71     asl $71       ... Bei
. 9a4e  26 72     rol $72       ... Überlauf
. 9a5Ø  9Ø Øb     bcc $9a5d     ... des
. 9a52  18        clc           ... Ergebnisregisters
. 9a53  8a        txa           ... wird
. 9a54  65 28     adc $28
. 9a56  aa        tax
. 9a57  98        tya
. 9a58  65 29     adc $29
. 9a5a  a8        tay
. 9a5b  bØ 91     bcs $99ee     ... 'OUT OF MEMORY ERROR' ausgegeben
```

```
. 9a5d  c6 5d     dec $5d
. 9a5f  dØ e3     bne $9a44
. 9a61  6Ø        rts           ENDE
. 9a62  :
. 9a62  *** F R E ***
. 9a62  :
. 9a62  a5 Ød     lda $Ød       Holt String-Flag
. 9a64  fØ Ø3     beq $9a69     Keine Stringvariable ?, dann $9a69
. 9a66  2Ø 4e 9c  jsr $9c4e     (FRESTR)
. 9a69  2Ø 54 a9  jsr $a954     GARBAGE COLLECT
. 9a6c  38        sec
. 9a6d  a5 33     lda $33       Holt Stringbereich-Anfang
. 9a6f  e5 31     sbc $31       ... Subtrahiert
. 9a71  a8        tay           ... Variablen-Ende
. 9a72  a5 34     lda $34       ... gleich
. 9a74  e5 32     sbc $32       ... freier Speicher
. 9a76  2Ø 92 9a  jsr $9a92     String-Flag initialisieren und FAC vorbereiten
. 9a79  38        sec
. 9a7a  4c ce a2  jmp $a2ce     Erzeugt Gleitkomma-Zahl in FAC

. 9a7d  :
. 9a7d  *** P O S ***
. 9a7d  :
. 9a7d  38        sec
. 9a7e  2Ø fØ ff  jsr $fffØ     (PLOT) Holt Cursor-Spalte ins Y-Reg.
. 9a81  a9 ØØ     lda #$ØØ     High-Byte gleich Ø
. 9a83  4c 71 94  jmp $9471     Akku, Y-Reg. als Gleitkomma-Zahl nach FAC

. 9a86  :
. 9a86  *** Prüft auf Direktmodus ***
. 9a86  :
. 9a86  24 81     bit $81       RUN-Modus ?
. 9a88  3Ø a4     bmi $9a2e     Ja, dann ENDE
. 9a8a  a2 15     ldx #$15      Holt Fehlernummer 21 ('ILLEGAL DIRECT')

. 9a8c  2c        .byte $2c

. 9a8d  a2 1b     ldx #$1b      Holt Fehlernummer 27 ('UNDEFINED FUNCTION')
. 9a8f  4c 83 86  jmp $8683     Fehlerausgabe

. 9a92  :
. 9a92  *** Akku, Y-Reg. nach FAC ***
. 9a92  :
. 9a92  a2 ØØ     ldx #$ØØ     String-Flag
. 9a94  86 Ød     stx $Ød       ... initialisieren
. 9a96  85 62     sta $62       Akku nach FAC
. 9a98  84 63     sty $63       Y-Reg. nach FAC
. 9a9a  a2 9Ø     ldx #$9Ø      Exponent
. 9a9c  6Ø        rts           ENDE

. 9a9d  :
. 9a9d  *** D E F ***
```

```
. 9a9d  :
. 9a9d  20 cb 9a jsr $9acb    Prüft FN-Namen und setzt Zeiger
. 9aa0  20 86 9a jsr $9a86    Wenn Direktmodus, dann Fehler
. 9aa3  20 8e 94 jsr $948e    Prüft ob Klammer auf '(' folgt
. 9aa6  a9 80    lda #$80     Sperrt
. 9aa8  85 10    sta $10      ... Integer-Variable
. 9aaa  20 a5 96 jsr $96a5    Variable suchen oder anlegen
. 9aad  20 17 93 jsr $9317    Numerischen Ausdruck auswerten
. 9ab0  20 8b 94 jsr $948b    Prüft ob Klammer zu ')' folgt
. 9ab3  a9 b2    lda #$b2     Holt "="-Token
. 9ab5  20 93 94 jsr $9493    Auswertung
. 9ab8  48       pha          1. Zeichen des FN-Ausdrucks auf Stack
. 9ab9  a5 48    lda $48      Holt FN-Variablenzeiger
. 9abb  48       pha          ... und
. 9abc  a5 47    lda $47      ... legt
. 9abe  48       pha          ... auf Stack
. 9abf  a5 3c    lda $3c      Holt Textpointer
. 9ac1  48       pha          ... und
. 9ac2  a5 3b    lda $3b      ... legt
. 9ac4  48       pha          ... auf Stack
. 9ac5  20 b0 8d jsr $8db0    Setzt Textpointer auf Trennzeichen
. 9ac8  4c 3e 9b jmp $9b3e    Alten Wert der FN-Variable wiederherstellen

. 9acb  :
. 9acb  *** F N ***
. 9acb  :
. 9acb  a9 a5    lda #$a5     Holt FN-Token
. 9acd  20 93 94 jsr $9493    Auswertung
. 9ad0  09 80    ora #$80     Sperrt
. 9ad2  85 10    sta $10      ... Integer-Variable
. 9ad4  20 ac 96 jsr $96ac    Variable suchen oder anlegen
. 9ad7  85 4e    sta $4e      Schreibt
. 9ad9  84 4f    sty $4f      ... Variablenzeiger
. 9adb  4c 17 93 jmp $9317    (CHKNUM) Prüft ob numerisch

. 9ade  20 cb 9a jsr $9acb    Prüft Namen und setzt Zeiger
. 9ae1  a5 4f    lda $4f      Zeiger der
. 9ae3  48       pha          ... FN-Namensvariable
. 9ae4  a5 4e    lda $4e      ... auf Stack
. 9ae6  48       pha          ... legen
. 9ae7  20 85 94 jsr $9485    Klammerterm auswerten
. 9aea  20 17 93 jsr $9317    (CHKNUM) Prüft ob numerisch
. 9aed  68       pla          Zeiger der
. 9aee  85 4e    sta $4e      ... FN-Namensvariable
. 9af0  68       pla          ... wieder-
. 9af1  85 4f    sta $4f      ... herstellen
. 9af3  a0 02    ldy #$02
. 9af5  20 59 81 jsr $8159    Holt FN-Variablenzeiger (Low) (LDA($4e),Y)
. 9af8  85 47    sta $47      Schreibt in aktuellen Variablenzeiger
. 9afa  aa       tax
. 9afb  c8       iny
. 9afc  20 59 81 jsr $8159    Holt FN-Variablenzeiger (High) (LDA($4e),Y)
```

```
. 9aff  f0 8c     beq $9a8d     Gleich 0 ?, dann 'UNDEFINED FUNCTION ERROR'
. 9b01  85 48     sta $48       Schreibt in aktuellen Variablenzeiger
. 9b03  c8        iny
. 9b04  20 61 81  jsr $8161     Holt Wert der FN-Variable (LDA($47),Y)
. 9b07  48        pha           Legt auf Stack
. 9b08  88        dey
. 9b09  10 f9     bpl $9b04
. 9b0b  a4 48     ldy $48
. 9b0d  20 59 a2  jsr $a259     Rundet FAC und schreibt in FN-Variable
. 9b10  a5 3c     lda $3c       Legt
. 9b12  48        pha           ... Textpointer
. 9b13  a5 3b     lda $3b       ... auf
. 9b15  48        pha           ... Stack
. 9b16  20 59 81  jsr $8159     Holt Zeiger auf
. 9b19  85 3b     sta $3b       ... FN-Ausdruck
. 9b1b  c8        iny           ... und
. 9b1c  20 59 81  jsr $8159     ... schreibt
. 9b1f  85 3c     sta $3c       ... in Textpointer
. 9b21  a5 48     lda $48       Legt
. 9b23  48        pha           ... FN-Variablenzeiger
. 9b24  a5 47     lda $47       ... auf
. 9b26  48        pha           ... Stack
. 9b27  20 14 93  jsr $9314     Wertet FN-Ausdruck aus
. 9b2a  68        pla           FN-Variablenzeiger
. 9b2b  85 4e     sta $4e       ... wieder-
. 9b2d  68        pla           ... herstellen
. 9b2e  85 4f     sta $4f
. 9b30  20 79 04  jsr $0479     CHRGOT-Routine
. 9b33  f0 03     beq $9b38     Folgt Trennzeichen ?, dann $9b38
. 9b35  4c a1 94  jmp $94a1     'SYNTAX ERROR' ausgeben

. 9b38  68        pla           Textpointer
. 9b39  85 3b     sta $3b       ... wiederherstellen
. 9b3b  68        pla
. 9b3c  85 3c     sta $3c
. 9b3e  a0 00     ldy #$00      Alten
. 9b40  68        pla           ... Wert
. 9b41  91 4e     sta ($4e),y   ... der
. 9b43  68        pla           ... FN-Variable
. 9b44  c8        iny           ... wiederherstellen
. 9b45  91 4e     sta ($4e),y
. 9b47  68        pla
. 9b48  c8        iny
. 9b49  91 4e     sta ($4e),y
. 9b4b  68        pla
. 9b4c  c8        iny
. 9b4d  91 4e     sta ($4e),y
. 9b4f  68        pla
. 9b50  c8        iny
. 9b51  91 4e     sta ($4e),y
. 9b53  60        rts           ENDE
```

```
. 9b54  :
. 9b54  *** STRING-PLATZRESERVIERUNG ***
. 9b54  :
. 9b54  a6 64     ldx $64       Zeiger
. 9b56  a4 65     ldy $65       ... auf
. 9b58  86 50     stx $50       ... Stringdeskriptor
. 9b5a  84 51     sty $51       ... zwischenspeichern
. 9b5c  20 06 a9  jsr $a906     Stringbereich reservieren
. 9b5f  86 62     stx $62       Schreibt
. 9b61  84 63     sty $63       ... Stringadresse
. 9b63  85 61     sta $61       Schreibt Stringlänge
. 9b65  60        rts           ENDE

. 9b66  :
. 9b66  *** S T R $ ***
. 9b66  :
. 9b66  20 17 93  jsr $9317     (CHKNUM) Prüft ob numerisch
. 9b69  a0 00     ldy #$00
. 9b6b  20 71 a4  jsr $a471     Gleitkomma in Zahlenstring wandeln
. 9b6e  68        pla
. 9b6f  68        pla
. 9b70  a9 ff     lda #$ff      Holt Adresse
. 9b72  a0 00     ldy #$00      ... von Zahlenstring-Puffer
. 9b74  :
. 9b74  *** STRING-ÜBERNAHME ***
. 9b74  :
. 9b74  a2 22     ldx #$22      Holt Anführungszeichen (")
. 9b76  86 07     stx $07       Zwischen-
. 9b78  86 08     stx $08       ... speichern
. 9b7a  85 6f     sta $6f       Schreibt Stringadresse (Low)
. 9b7c  84 70     sty $70       Schreibt Stringadresse (High)
. 9b7e  85 62     sta $62       Schreibt Stringadresse (Low) in FAC
. 9b80  84 63     sty $63       Schreibt Stringadresse (High) in FAC
. 9b82  a0 ff     ldy #$ff
. 9b84  c8        iny
. 9b85  20 c6 04  jsr $04c6     Holt nächstes Zeichen (LDA($6f),Y)
. 9b88  f0 0c     beq $9b96     Endmarke ?, dann $9b96
. 9b8a  c5 07     cmp $07       Auf
. 9b8c  f0 04     beq $9b92     ... Delimiter
. 9b8e  c5 08     cmp $08       ... prüfen
. 9b90  d0 f2     bne $9b84
. 9b92  c9 22     cmp #$22      Auf Anführungszeichen (") prüfen
. 9b94  f0 01     beq $9b97     Wenn Anführungszeichen, dann $9b97
. 9b96  18        clc
. 9b97  84 61     sty $61       Schreibt Stringlänge in FAC
. 9b99  98        tya           Addiert Stringlänge
. 9b9a  65 6f     adc $6f       ... mit String-Anfangsadresse
. 9b9c  85 71     sta $71       Ergebnis gleich String-Endadresse + 1
. 9b9e  a6 70     ldx $70
. 9ba0  90 01     bcc $9ba3
. 9ba2  e8        inx
. 9ba3  86 72     stx $72
```

```
. 9ba5  98       tya
. 9ba6  20 54 9b jsr $9b54    String-Platzreservierung
. 9ba9  a6 6f    ldx $6f      Holt
. 9bab  a4 70    ldy $70      ... Stringadresse
. 9bad  20 2c 9c jsr $9c2c    String in Bereich kopieren
. 9bb0  :
. 9bb0  *** DESKRIPTOR IN STRING-STACK ***
. 9bb0  :
. 9bb0  a6 16    ldx $16      Holt Zeiger auf temporären String-Stack
. 9bb2  e0 22    cpx #$22     String-Stack voll ?
. 9bb4  d0 05    bne $9bbb    Nein, dann $9bbb
. 9bb6  a2 19    ldx #$19     Holt Fehlernummer 25 ('FORMULA TOO COMPLEX')
. 9bb8  4c 83 86 jmp $8683    Fehlerausgabe

. 9bbb  a5 61    lda $61      Holt Stringlänge
. 9bbd  95 00    sta $00,x    Schreibt in String-Stack
. 9bbf  a5 62    lda $62      Holt String-Adresse (Low)
. 9bc1  95 01    sta $01,x    Schreibt in String-Stack
. 9bc3  a5 63    lda $63      Holt String-Adresse (High)
. 9bc5  95 02    sta $02,x    Schreibt in String-Stack
. 9bc7  a0 00    ldy #$00
. 9bc9  86 64    stx $64      Schreibt Adresse des Deskriptors im Stack (Low)
. 9bcb  84 65    sty $65      Schreibt Adresse des Deskriptors im Stack (High)
. 9bcd  84 70    sty $70      Schreibt Adresse des Deskriptors im Stack (High)
. 9bcf  88       dey
. 9bd0  84 0d    sty $0d      Setzt String-Flag
. 9bd2  86 17    stx $17      Schreibt Zeiger auf letzten Stack-Eintrag
. 9bd4  e8       inx
. 9bd5  e8       inx
. 9bd6  e8       inx
. 9bd7  86 16    stx $16      Schreibt Zeiger auf neuen Stack-Eintrag
. 9bd9  60       rts          ENDE

. 9bda  :
. 9bda  *** STRING-VERKETTUNG ***
. 9bda  :
. 9bda  a5 65    lda $65      Legt Zeiger auf
. 9bdc  48       pha          ... Deskriptor des 1. Strings
. 9bdd  a5 64    lda $64      ... auf
. 9bdf  48       pha          ... Stack
. 9be0  20 14 94 jsr $9414    Holt Adresse des 2. Deskriptors
. 9be3  20 1a 93 jsr $931a    (CHKSTR) Prüft auf String
. 9be6  68       pla          Zeiger auf
. 9be7  85 6f    sta $6f      ... Deskriptor
. 9be9  68       pla          ... des 1. Strings
. 9bea  85 70    sta $70      ... wiederherstellen
. 9bec  a0 00    ldy #$00
. 9bee  20 c6 04 jsr $04c6    Holt Länge des 1. Strings (LDA($6f),Y)
. 9bf1  85 78    sta $78      Zwischenspeichern
. 9bf3  20 dc 04 jsr $04dc    Holt Länge des 2. Strings (LDA($64),Y)
. 9bf6  18       clc          Zur Länge
. 9bf7  65 78    adc $78      ... des 1. Strings addieren
```

```
. 9bf9  90 03    bcc $9bfe    String nicht zu lange ?, dann $9bfe
. 9bfb  4c 4c cc jmp $cc4c    Sonst 'STRING TOO LONG ERROR' ausgeben

. 9bfe  20 54 9b jsr $9b54    String-Platzreservierung
. 9c01  20 1b 9c jsr $9c1b    String-Kopierroutine
. 9c04  a5 50    lda $50      Holt Zeiger
. 9c06  a4 51    ldy $51      ... auf 2. Deskriptor
. 9c08  20 52 9c jsr $9c52    (FRESTR)
. 9c0b  20 30 9c jsr $9c30    Kopiert 2. String hinter 1. String
. 9c0e  a5 6f    lda $6f      Holt Zeiger
. 9c10  a4 70    ldy $70      ... auf 1. Deskriptor
. 9c12  20 52 9c jsr $9c52    (FRESTR)
. 9c15  20 b0 9b jsr $9bb0    Deskriptor in String-Stack
. 9c18  4c 46 93 jmp $9346    Zurück zur Auswertung

. 9c1b  :
. 9c1b  *** STRING-KOPIERROUTINE ***
. 9c1b  :
. 9c1b  a0 00    ldy #$00
. 9c1d  20 c6 04 jsr $04c6    Holt Stringlänge (LDA($6f),Y)
. 9c20  48       pha          Legt auf Stack
. 9c21  c8       iny
. 9c22  20 c6 04 jsr $04c6    Holt Stringadresse (Low) (LDA($6f),Y)
. 9c25  aa       tax
. 9c26  c8       iny
. 9c27  20 c6 04 jsr $04c6    Holt Stringadresse (High) (LDA($6f),Y)
. 9c2a  a8       tay
. 9c2b  68       pla          Holt Stringlänge vom Stack
. 9c2c  86 22    stx $22      Schreibt Stringadresse (Low)
. 9c2e  84 23    sty $23      Schreibt Stringadresse (High)
. 9c30  a8       tay          Stringlänge ins Y-Reg.
. 9c31  f0 0b    beq $9c3e    Stringlänge gleich 0, dann fertig
. 9c33  48       pha
. 9c34  88       dey
. 9c35  20 b0 04 jsr $04b0    Holt Zeichen aus String (LDA($22),Y)
. 9c38  91 35    sta ($35),y  In Stringbereich kopieren
. 9c3a  98       tya          Stringende erreicht ?
. 9c3b  d0 f7    bne $9c34    Nein, dann $9c34
. 9c3d  68       pla          Stringlänge
. 9c3e  18       clc          ... addiert mit
. 9c3f  65 35    adc $35      ... Stringbereich-
. 9c41  85 35    sta $35      ... Anfang
. 9c43  90 02    bcc $9c47    ... gleich
. 9c45  e6 36    inc $36      ... Stringende
. 9c47  60       rts          ENDE
```

```
. 9c48  :
. 9c48  *** FRESTR ***
. 9c48  :
. 9c48  20 2c 93 jsr $932c    (FRMEVL) Ausdruck auswerten
. 9c4b  20 1a 93 jsr $931a    (CHKSTR) Prüft auf String
. 9c4e  a5 64    lda $64      Holt
. 9c50  a4 65    ldy $65      ... Deskriptor-Zeiger
. 9c52  85 22    sta $22      In Hilfszeiger
. 9c54  84 23    sty $23      ... ablegen
. 9c56  20 aa 9c jsr $9caa    Entfernt Deskriptor aus Stack
. 9c59  d0 39    bne $9c94    War Deskriptor nicht im Stack ?, dann $9c94
. 9c5b  20 9c 8f jsr $8f9c    String auf R-Zeiger prüfen
. 9c5e  90 34    bcc $9c94    Kein R-Zeiger ?, dann $9c94
. 9c60  88       dey
. 9c61  a9 ff    lda #$ff     String löschen (mit $FF)
. 9c63  91 22    sta ($22),y
. 9c65  88       dey
. 9c66  8a       txa
. 9c67  91 22    sta ($22),y
. 9c69  48       pha          Speichert Stringlänge
. 9c6a  49 ff    eor #$ff     Invertiert Stringlänge
. 9c6c  38       sec          Addiert mit 1 ergibt negative Stringlänge
. 9c6d  65 22    adc $22      Stringlänge von Adresse
. 9c6f  a4 23    ldy $23      des R-Zeigers subtrahieren
. 9c71  b0 01    bcs $9c74
. 9c73  88       dey
. 9c74  85 22    sta $22      Schreibt
. 9c76  84 23    sty $23      ... Stringadresse
. 9c78  aa       tax
. 9c79  68       pla
. 9c7a  c4 34    cpy $34      Liegt String
. 9c7c  d0 3c    bne $9cba    ... am unteren Ende
. 9c7e  e4 33    cpx $33      ... des Stringspeichers ?
. 9c80  d0 38    bne $9cba    Nein, dann $9cba
. 9c82  48       pha
. 9c83  38       sec
. 9c84  65 33    adc $33      Zeiger auf
. 9c86  85 33    sta $33      ... Stringspeicher
. 9c88  90 02    bcc $9c8c    ... um Stringlänge + 1
. 9c8a  e6 34    inc $34      ... hinaufsetzen
. 9c8c  e6 33    inc $33      Zeiger auf Stringspeicher
. 9c8e  d0 02    bne $9c92    ... für R-Zeiger
. 9c90  e6 34    inc $34      ... um 1 hinaufsetzen
. 9c92  68       pla          Holt Stringlänge vom Stack
. 9c93  60       rts          Zurück

. 9c94  a0 00    ldy #$00
. 9c96  20 b0 04 jsr $04b0    Holt Stringlänge (LDA($22),Y)
. 9c99  48       pha          Auf Stack
. 9c9a  c8       iny
. 9c9b  20 b0 04 jsr $04b0    Holt Stringadresse (Low) (LDA($22),Y)
. 9c9e  aa       tax
```

```
. 9c9f  c8       iny
. 9ca0  20 b0 04 jsr $04b0    Holt Stringadresse (High) (LDA($22),Y)
. 9ca3  a8       tay
. 9ca4  86 22    stx $22      In Hilfszeiger
. 9ca6  84 23    sty $23      ... ablegen
. 9ca8  68       pla          Holt Stringlänge vom Stack
. 9ca9  60       rts          Zurück

. 9caa  c4 18    cpy $18      Deskriptor
. 9cac  d0 0c    bne $9cba    ... im
. 9cae  c5 17    cmp $17      ... String-Stack ?
. 9cb0  d0 08    bne $9cba    Nein, dann $9cba
. 9cb2  85 16    sta $16      Stack-Zeiger
. 9cb4  e9 03    sbc #$03     Minus 3
. 9cb6  85 17    sta $17      Gleich Zeiger auf letzten String
. 9cb8  a0 00    ldy #$00     Setzt Null-Flag
. 9cba  60       rts          ENDE

. 9cbb  :
. 9cbb  *** C H R $ ***
. 9cbb  :
. 9cbb  20 87 9d jsr $9d87    (GETBYT) Byte übernehmen
. 9cbe  8a       txa          ... und auf
. 9cbf  48       pha          ... Stack legen
. 9cc0  a9 01    lda #$01     Stringlänge gleich 1
. 9cc2  20 5c 9b jsr $9b5c    Stringbereich reservieren
. 9cc5  68       pla          Holt Byte von Stack
. 9cc6  a0 00    ldy #$00
. 9cc8  91 62    sta ($62),y  Byte im Stringbereich ablegen
. 9cca  68       pla          Rücksprungadresse
. 9ccb  68       pla          ... vom Stack holen
. 9ccc  4c b0 9b jmp $9bb0    Deskriptor in String-Stack

. 9ccf  :
. 9ccf  *** L E F T $ ***
. 9ccf  :
. 9ccf  20 46 9d jsr $9d46    Holt Stringadresse und Parameter vom Stack
. 9cd2  48       pha          Left$-Parameter merken
. 9cd3  20 81 81 jsr $8181    Holt Stringlänge (LDA($50),Y)
. 9cd6  85 78    sta $78      Stringlänge zwischenspeichern
. 9cd8  68       pla          Holt Left$-Parameter
. 9cd9  c5 78    cmp $78      Parameter kleiner als Stringlänge ?
. 9cdb  98       tya          Akku gleich 0
. 9cdc  90 05    bcc $9ce3    Ja, dann $9ce3
. 9cde  20 81 81 jsr $8181    Holt Stringlänge (LDA($50),Y)
. 9ce1  aa       tax          Als Left$-Parameter nehmen
. 9ce2  98       tya          Akku gleich 0
. 9ce3  48       pha          Als Position des 1. Elements
. 9ce4  8a       txa          Akku gleich Left$-Parameter
. 9ce5  48       pha
. 9ce6  20 5c 9b jsr $9b5c    Stringbereich reservieren
. 9ce9  a5 50    lda $50      Holt
```

```
. 9ceb  a4 51     ldy $51       Deskriptor-Zeiger
. 9ced  20 52 9c  jsr $9c52     (FRESTR)
. 9cf0  68        pla           Länge des neuen Strings
. 9cf1  a8        tay
. 9cf2  68        pla           Nummer des 1. Elements (bei left$: 0)
. 9cf3  18        clc           ... addiert mit Adresse
. 9cf4  65 22     adc $22       ... des gegebenen
. 9cf6  85 22     sta $22       ... Strings
. 9cf8  90 02     bcc $9cfc     ... gleich
. 9cfa  e6 23     inc $23       ... Adresse des neuen Strings
. 9cfc  98        tya
. 9cfd  20 30 9c  jsr $9c30     String in Stringbereich kopieren
. 9d00  4c b0 9b  jmp $9bb0     Deskriptor in String-Stack

. 9d03  :
. 9d03  *** R I G H T $ ***
. 9d03  :
. 9d03  20 46 9d  jsr $9d46     Holt Stringadresse und Parameter vom Stack
. 9d06  48        pha           Right$-Parameter merken
. 9d07  20 81 81  jsr $8181     Holt Stringlänge (LDA($50),Y)
. 9d0a  85 78     sta $78       Stringlänge zwischenspeichern
. 9d0c  68        pla           Holt Right$-Parameter
. 9d0d  18        clc           Minus 1
. 9d0e  e5 78     sbc $78       Minus Stringlänge
. 9d10  49 ff     eor #$ff      Ergebnis invertieren gleich Nr. des 1. Elements
. 9d12  4c dc 9c  jmp $9cdc     Weiter bei Left$

. 9d15  :
. 9d15  *** M I D $ ***
. 9d15  :
. 9d15  a9 ff     lda #$ff
. 9d17  85 65     sta $65
. 9d19  20 79 04  jsr $0479     CHRGET-Routine
. 9d1c  c9 29     cmp #$29      2. Parameter vorhanden ?
. 9d1e  f0 06     beq $9d26     Nein, dann $9d26
. 9d20  20 91 94  jsr $9491     (CHKCOM) Prüft ob Komma folgt
. 9d23  20 84 9d  jsr $9d84     (GETBYT) 2. Parameter holen
. 9d26  20 46 9d  jsr $9d46     Stringadresse und 1. Parameter holen
. 9d29  f0 53     beq $9d7e     Wenn 1. Parameter gleich 0, dann Fehler
. 9d2b  ca        dex
. 9d2c  8a        txa
. 9d2d  48        pha           Nummer des 1. Elements im gegebenen String
. 9d2e  a2 00     ldx #$00
. 9d30  48        pha
. 9d31  20 81 81  jsr $8181     Holt Länge des Strings (LDA($50),Y)
. 9d34  85 78     sta $78       Länge zwischenspeichern
. 9d36  68        pla           Holt Nr. des 1. Elements vom MID$(...)
. 9d37  18        clc           Subtrahiert Länge
. 9d38  e5 78     sbc $78       ... des gegebenen String
. 9d3a  b0 a8     bcs $9ce4     Wenn nicht negativ, dann $9ce4
. 9d3c  49 ff     eor #$ff      Berechnet neue Stringlänge
. 9d3e  c5 65     cmp $65
```

```
. 9d40  90 a3    bcc $9ce5    Stringlänge kleiner 255, dann $9ce5
. 9d42  a5 65    lda $65
. 9d44  b0 9f    bcs $9ce5    Springt immer nach $9ce5

. 9d46  20 8b 94 jsr $948b    Prüft ob 'Klammer zu' folgt
. 9d49  68       pla          Holt letzte
. 9d4a  a8       tay          ... Rücksprungadresse
. 9d4b  68       pla          ... vom Stack
. 9d4c  85 55    sta $55
. 9d4e  68       pla          Holt Rücksprungadresse
. 9d4f  68       pla          ... von $95ec (Funktionsaufruf) für JSR $54
. 9d50  68       pla          Holt 1. Parameter
. 9d51  aa       tax
. 9d52  68       pla          Holt Adresse
. 9d53  85 50    sta $50      ... des String-
. 9d55  68       pla          ... Deskriptors und
. 9d56  85 51    sta $51      ... schreibt in Arbeitsbereich
. 9d58  a5 55    lda $55      Legt letzte
. 9d5a  48       pha          ... Rücksprungadresse
. 9d5b  98       tya          ... wieder
. 9d5c  48       pha          ... auf Stack
. 9d5d  a0 00    ldy #$00
. 9d5f  8a       txa
. 9d60  60       rts          ENDE

. 9d61  :
. 9d61  *** L E N ***
. 9d61  :
. 9d61  20 67 9d jsr $9d67    (FRESTR) und String-Flag zurücksetzen
. 9d64  4c 81 9a jmp $9a81    Schreibt Byte nach FAC

. 9d67  20 4b 9c jsr $9c4b    (FRESTR)
. 9d6a  a2 00    ldx #$00
. 9d6c  86 0d    stx $0d      String-Flag initialisieren
. 9d6e  a8       tay          Stringlänge ins Y-Reg.
. 9d6f  60       rts          ENDE

. 9d70  :
. 9d70  *** A S C ***
. 9d70  :
. 9d70  20 67 9d jsr $9d67    (FRESTR) und String-Flag zurücksetzen
. 9d73  f0 06    beq $9d7b    Stringlänge gleich 0 ?, dann $9d7b
. 9d75  a0 00    ldy #$00
. 9d77  20 b0 04 jsr $04b0    Holt 1. Zeichen aus String (LDA($22),Y)
. 9d7a  a8       tay
. 9d7b  4c 81 9a jmp $9a81    Schreibt Byte nach FAC

. 9d7e  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. 9d81  :
. 9d81  *** GETBYT ***
. 9d81  :
```

```
. 9d81  20 73 04 jsr $0473    CHRGET-Routine
. 9d84  20 14 93 jsr $9314    (FRMNUM) Numerischen Ausdruck auswerten
. 9d87  20 7f 98 jsr $987f    Gleitkomma-Integer-Wandlung; Fehler wenn neg.
. 9d8a  a6 64    ldx $64      Holt High-Byte des Ausdrucks
. 9d8c  d0 f0    bne $9d7e    Ungleich 0 ?, dann Fehler
. 9d8e  a6 65    ldx $65      Holt Low-Byte des Ausdrucks
. 9d90  4c 79 04 jmp $0479    CHRGET-Routine

. 9d93  :
. 9d93  *** V A L ***
. 9d93  :
. 9d93  20 67 9d jsr $9d67    (FRESTR) und String-Flag zurücksetzen
. 9d96  f0 37    beq $9dcf    Stringlänge gleich 0 ?, dann $9dcf
. 9d98  a6 3b    ldx $3b      Holt
. 9d9a  a4 3c    ldy $3c      ... Textpointer
. 9d9c  86 71    stx $71      Textpointer
. 9d9e  84 72    sty $72      ... zwischenspeichern
. 9da0  a6 22    ldx $22      Schreibt
. 9da2  86 3b    stx $3b      ... String-
. 9da4  18       clc          ... Anfangsadresse
. 9da5  65 22    adc $22      ... in
. 9da7  85 24    sta $24      ... Textpointer.
. 9da9  a6 23    ldx $23      ... Neue
. 9dab  86 3c    stx $3c      ... String-Endadresse
. 9dad  90 01    bcc $9db0    ... gleich
. 9daf  e8       inx          ... Stringlänge plus
. 9db0  86 25    stx $25      ... String-Anfangsadresse.
. 9db2  a0 00    ldy #$00
. 9db4  20 bb 04 jsr $04bb    Holt Zeichen nach String (LDA($24),Y)
. 9db7  48       pha          Zeichen zwischenspeichern
. 9db8  98       tya          ... und
. 9db9  91 24    sta ($24),y  ... durch $00 ersetzen
. 9dbb  20 79 04 jsr $0479    CHRGOT-Routine
. 9dbe  20 7f a3 jsr $a37f    String-Gleitkomma-Wandlung und Gleitkomma nach FAC
. 9dc1  68       pla          Holt Zeichen
. 9dc2  a0 00    ldy #$00
. 9dc4  91 24    sta ($24),y  Zeichen wieder zurückschreiben
. 9dc6  a6 71    ldx $71      Textpointer
. 9dc8  a4 72    ldy $72      ... wiederherstellen
. 9dca  86 3b    stx $3b
. 9dcc  84 3c    sty $3c
. 9dce  60       rts          ENDE

. 9dcf  4c 2b 9f jmp $9f2b    FAC löschen

. 9dd2  :
. 9dd2  *** FRMNUM, GETADR, CHKCOM und GETBYT ***
. 9dd2  :
. 9dd2  20 14 93 jsr $9314    (FRMNUM) Numerischen Ausdruck auswerten
. 9dd5  20 e4 9d jsr $9de4    (GETADR)
. 9dd8  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. 9ddb  4c 84 9d jmp $9d84    (GETBYT)
```

```
. 9dde  :
. 9dde  *** GETADR ***
. 9dde  :
. 9dde  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. 9de1  20 14 93 jsr $9314    (FRMNUM) Numerischen Ausdruck auswerten
. 9de4  a5 66    lda $66      Holt Vorzeichen von FAC
. 9de6  30 96    bmi $9d7e    Wenn negativ, dann Fehler
. 9de8  a5 61    lda $61      Holt Exponent von FAC
. 9dea  c9 91    cmp #$91     Exponent größer 16 ?
. 9dec  b0 90    bcs $9d7e    Ja, dann Fehler
. 9dee  20 27 a3 jsr $a327    Gleitkomma-Integer-Wandlung
. 9df1  a5 64    lda $64      Holt
. 9df3  a4 65    ldy $65      ... Integer
. 9df5  84 14    sty $14      Schreibt
. 9df7  85 15    sta $15      ... Adresse
. 9df9  60       rts          ENDE

. 9dfa  :
. 9dfa  *** P E E K ***
. 9dfa  :
. 9dfa  a5 15    lda $15      Rettet
. 9dfc  48       pha          ... Adresse
. 9dfd  a5 14    lda $14      ... auf
. 9dff  48       pha          ... Stack
. 9e00  20 e4 9d jsr $9de4    (GETADR)
. 9e03  a0 00    ldy #$00
. 9e05  20 5d 81 jsr $815d    Holt PEEK-Wert (LDA($14),Y)
. 9e08  a8       tay          PEEK-Wert ins Y-Reg.
. 9e09  68       pla          Adresse
. 9e0a  85 14    sta $14      ... vom
. 9e0c  68       pla          ... Stack
. 9e0d  85 15    sta $15      ... holen
. 9e0f  4c 81 9a jmp $9a81    Schreibt Byte nach FAC

. 9e12  :
. 9e12  *** P O K E ***
. 9e12  :
. 9e12  20 d2 9d jsr $9dd2    FRMNUM, GETADR, CHKCOM und GETBYT
. 9e15  8a       txa          Byte
. 9e16  a0 00    ldy #$00     ... an POKE-Adresse
. 9e18  91 14    sta ($14),y  ... speichern
. 9e1a  60       rts          ENDE

. 9e1b  :
. 9e1b  *** D E C ***
. 9e1b  :
. 9e1b  20 67 9d jsr $9d67    (FRESTR) und String-Flag initialisieren
. 9e1e  85 24    sta $24      Schreibt Stringlänge
. 9e20  a0 00    ldy #$00     Zeiger in String initialisieren
. 9e22  84 25    sty $25      Stellenzähler initialisieren
. 9e24  84 71    sty $71
```

```
. 9e26  84 72     sty $72
. 9e28  c4 24     cpy $24       String-Ende erreicht ?
. 9e2a  f0 34     beq $9e60     Ja, dann $9e60
. 9e2c  20 b0 04  jsr $04b0     Holt Zeichen aus String (LDA($22),Y)
. 9e2f  c8        iny
. 9e30  c9 20     cmp #$20      Leerzeichen ?
. 9e32  f0 f4     beq $9e28     Ja, dann überlesen
. 9e34  e6 25     inc $25
. 9e36  a6 25     ldx $25       Holt Stellenzähler
. 9e38  e0 05     cpx #$05      Schon 5 Stellen erreicht ?
. 9e3a  f0 2b     beq $9e67     Ja, dann Fehler
. 9e3c  c9 30     cmp #$30      Prüft,
. 9e3e  90 27     bcc $9e67     ... ob
. 9e40  c9 3a     cmp #$3a      ... HEX-
. 9e42  90 0a     bcc $9e4e     ... Ziffern
. 9e44  c9 41     cmp #$41      ... zulässig
. 9e46  90 1f     bcc $9e67
. 9e48  c9 47     cmp #$47
. 9e4a  b0 1b     bcs $9e67     Nein, dann Fehler
. 9e4c  e9 07     sbc #$07
. 9e4e  e9 2f     sbc #$2f      Hex-Ziffer
. 9e50  0a        asl           ... um 4 Bit
. 9e51  0a        asl           ... nach
. 9e52  0a        asl           ... links
. 9e53  0a        asl           ... verschieben
. 9e54  a2 04     ldx #$04
. 9e56  0a        asl           ... und
. 9e57  26 71     rol $71       ... in das Ergebnis
. 9e59  26 72     rol $72       ... einrotieren
. 9e5b  ca        dex           Schon 4 Bit einrotiert ?
. 9e5c  d0 f8     bne $9e56     Nein, dann $9e56
. 9e5e  f0 c8     beq $9e28     Sonst nach $9e28

. 9e60  a4 71     ldy $71       Holt Integer-Wert
. 9e62  a5 72     lda $72       ... aus Ergebnis-Register
. 9e64  4c 76 9a  jmp $9a76     Erzeugt Gleitkommazahl im FAC

. 9e67  4c 1c 99  jmp $991c     'ILLEGAL QUANTITY ERROR' ausgeben

. 9e6a  :
. 9e6a  *** W A I T ***
. 9e6a  :
. 9e6a  20 d2 9d  jsr $9dd2     FRMNUM, GETADR, CHKCOM und GETBYT
. 9e6d  86 49     stx $49       Speichert 1. Byte
. 9e6f  a2 00     ldx #$00
. 9e71  20 79 04  jsr $0479     CHRGOT-Routine
. 9e74  f0 03     beq $9e79     Folgt Trennzeichen ?, dann $9e79
. 9e76  20 d8 9d  jsr $9dd8     CHKCOM und GETBYT
. 9e79  86 4a     stx $4a       Speichert 2. Byte
. 9e7b  a0 00     ldy #$00
. 9e7d  20 5d 81  jsr $815d     Holt Byte unter Adresse (LDA($14),Y)
. 9e80  45 4a     eor $4a       Exclusiv-Oder-Verknüpfung mit 2. Byte
```

```
. 9e82  25 49     and $49       UND-Verknüpfung mit 1. Byte
. 9e84  f0 f7     beq $9e7d     Wenn Ergebnis gleich 0, dann warten...
. 9e86  60        rts           ENDE

. 9e87  :
. 9e87  *** FAC := ARG - FAC ***
. 9e87  :
. 9e87  a5 66     lda $66       Holt Vorzeichen von FAC
. 9e89  49 ff     eor #$ff      Vorzeichen invertieren
. 9e8b  85 66     sta $66       Vorzeichen zurückschreiben
. 9e8d  45 6e     eor $6e       Mit Vorzeichen
. 9e8f  85 6f     sta $6f       ... von ARG Exclusic-Oder verknüpfen
. 9e91  a5 61     lda $61       Holt Exponent von FAC
. 9e93  4c 9e 9e  jmp $9e9e     FAC := ARG + FAC

. 9e96  :
. 9e96  *** EXPONENT VON FAC := EXPONENT VON ARG ***
. 9e96  :
. 9e96  20 cd 9f  jsr $9fcd     Vergleicht Exponenten
. 9e99  90 3c     bcc $9ed7     Springt immer nach $9ed7

. 9e9b  :
. 9e9b  *** FAC := VARIABLE + FAC ***
. 9e9b  :
. 9e9b  20 07 a1  jsr $a107     Variable in Adresse (Akku, Y-Reg) nach ARG

. 9e9e  :
. 9e9e  *** FAC := ARG + FAC ***
. 9e9e  :
. 9e9e  d0 03     bne $9ea3     Wenn FAC ungleich 0, dann $9ea3
. 9ea0  4c 81 a2  jmp $a281     FAC := ARG

. 9ea3  a6 70     ldx $70       Holt niederwertige Stelle von FAC
. 9ea5  86 56     stx $56
. 9ea7  a2 69     ldx #$69      Holt Offset-Zeiger auf ARG
. 9ea9  a5 69     lda $69       Holt Exponent von ARG
. 9eab  a8        tay
. 9eac  f0 d8     beq $9e86     Wenn ARG gleich 0, dann Ende
. 9eae  38        sec
. 9eaf  e5 61     sbc $61       Subtrahiert Exponent von FAC
. 9eb1  f0 24     beq $9ed7     Ergebnis gleich 0 ?, dann $9ed7
. 9eb3  90 12     bcc $9ec7     Exponent von FAC größer ?, dann 9ec7
. 9eb5  84 61     sty $61       Exponent von FAC := Exponent von ARG
. 9eb7  a4 6e     ldy $6e       Holt Vorzeichen von ARG
. 9eb9  84 66     sty $66       In Vorzeichen von ARG übernehmen
. 9ebb  49 ff     eor #$ff      Differenz der Exponenten invertieren
. 9ebd  69 00     adc #$00      Carry addieren
. 9ebf  a0 00     ldy #$00
. 9ec1  84 56     sty $56
. 9ec3  a2 61     ldx #$61      Holt Offset-Zeiger für FAC
. 9ec5  d0 04     bne $9ecb     Springt immer nach $9ecb
```

```
. 9ec7  aØ ØØ     ldy #$ØØ
. 9ec9  84 7Ø     sty $7Ø       Niederwertige Stelle von FAC initialisieren
. 9ecb  c9 f9     cmp #$f9      Differenz der Exponenten größer 7 ?
. 9ecd  3Ø c7     bmi $9e96     Exponent von FAC gleich Exponent von ARG
. 9ecf  a8        tay
. 9edØ  a5 7Ø     lda $7Ø       Holt niederwertige Stelle von FAC
. 9ed2  56 Ø1     lsr $Ø1,x
. 9ed4  2Ø e4 9f  jsr $9fe4     Bits egalisieren
. 9ed7  24 6f     bit $6f       Vorzeichen vergleichen
. 9ed9  1Ø 57     bpl $9f32     Gleich ?, dann Mantissen addieren
. 9edb  aØ 61     ldy #$61      Holt Offset-Zeiger für FAC
. 9edd  eØ 69     cpx #$69      Ist X-Reg. gleich Offset-Zeiger für ARG ?
. 9edf  fØ Ø2     beq $9ee3     Ja, dann $9ee3
. 9ee1  aØ 69     ldy #$69      Sonst Y-Reg gleich Offset-Zeiger für ARG
. 9ee3  38        sec           Subtrahieren der Mantissen...
. 9ee4  49 ff     eor #$ff
. 9ee6  65 56     adc $56
. 9ee8  85 7Ø     sta $7Ø       (niederwertige Stelle)
. 9eea  b9 Ø4 ØØ  lda $ØØØ4,y
. 9eed  f5 Ø4     sbc $Ø4,x
. 9eef  85 65     sta $65       (4. Stelle)
. 9ef1  b9 Ø3 ØØ  lda $ØØØ3,y
. 9ef4  f5 Ø3     sbc $Ø3,x
. 9ef6  85 64     sta $64       (3. Stelle)
. 9ef8  b9 Ø2 ØØ  lda $ØØØ2,y
. 9efb  f5 Ø2     sbc $Ø2,x
. 9efd  85 63     sta $63       (2. Stelle)
. 9eff  b9 Ø1 ØØ  lda $ØØØ1,y
. 9fØ2  f5 Ø1     sbc $Ø1,x
. 9fØ4  85 62     sta $62       (1. Stelle)
. 9fØ6  bØ Ø3     bcs $9fØb     Kein Unterlauf ?, dann $9fØb
. 9fØ8  2Ø 7b 9f  jsr $9f7b     Mantisse von FAC invertieren
. 9fØb  aØ ØØ     ldy #$ØØ      FAC
. 9fØd  98        tya           ... linksbündig
. 9fØe  18        clc           ... normalisieren
. 9fØf  a6 62     ldx $62       1. Stelle gleich Ø ?
. 9f11  dØ 4a     bne $9f5d     Nein, dann $9f5d
. 9f13  a6 63     ldx $63       Sonst Mantisse um 1 Byte nach links schieben...
. 9f15  86 62     stx $62       (1. Stelle gleich 2. Stelle)
. 9f17  a6 64     ldx $64
. 9f19  86 63     stx $63       (2. Stelle gleich 3. Stelle)
. 9f1b  a6 65     ldx $65
. 9f1d  86 64     stx $64       (3. Stelle gleich 4. Stelle)
. 9f1f  a6 7Ø     ldx $7Ø
. 9f21  86 65     stx $65       (4. Stelle gleich niederwertige Stelle)
. 9f23  84 7Ø     sty $7Ø       Niederwertige Stelle gleich Ø
. 9f25  69 Ø8     adc #$Ø8      Verschiebung um 8 Bit
. 9f27  c9 2Ø     cmp #$2Ø      Schon 4 Byte verschoben ?
. 9f29  dØ e4     bne $9fØf     Nein, dann $9fØf
. 9f2b  a9 ØØ     lda #$ØØ      Sonst: Alle Mantissen-Stellen sind Ø
. 9f2d  85 61     sta $61       Also auch Exponent gleich Ø setzen
. 9f2f  85 66     sta $66       Auch Vorzeichenbyte gleich Ø setzen
```

```
. 9f31  60       rts ENDE

. 9f32  65 56    adc $56      Addieren der Mantissen
. 9f34  85 70    sta $70      (niederwertige Stelle)
. 9f36  a5 65    lda $65
. 9f38  65 6d    adc $6d
. 9f3a  85 65    sta $65      (4. Stelle)
. 9f3c  a5 64    lda $64
. 9f3e  65 6c    adc $6c
. 9f40  85 64    sta $64      (3. Stelle)
. 9f42  a5 63    lda $63
. 9f44  65 6b    adc $6b
. 9f46  85 63    sta $63      (2. Stelle)
. 9f48  a5 62    lda $62
. 9f4a  65 6a    adc $6a
. 9f4c  85 62    sta $62      (1. Stelle)
. 9f4e  4c 6a 9f jmp $9f6a    Wenn nötig Überlauf bearbeiten

. 9f51  69 01    adc #$01     FAC so oft
. 9f53  06 70    asl $70      ... um 1 Bit
. 9f55  26 65    rol $65      ... nach links verschieben
. 9f57  26 64    rol $64      ... bis das höchste Bit
. 9f59  26 63    rol $63      ... der 1. Stelle
. 9f5b  26 62    rol $62      ... gesetzt
. 9f5d  10 f2    bpl $9f51    ... ist
. 9f5f  38       sec          Wenn Zahl der
. 9f60  e5 61    sbc $61      ... Verschiebungen minus Exponent
. 9f62  b0 c7    bcs $9f2b    ... größer 0, dann FAC := 0
. 9f64  49 ff    eor #$ff     Exponent minus
. 9f66  69 01    adc #$01     ... Zahl der Verschiebungen
. 9f68  85 61    sta $61      ... gleich neuer Exponent
. 9f6a  90 0e    bcc $9f7a    Wenn kein Überlauf, dann $9f7a
. 9f6c  e6 61    inc $61      Exponent um 1 erhöhen
. 9f6e  f0 42    beq $9fb2    Exponent gleich 0 ?, dann Fehler
. 9f70  66 62    ror $62      Mantisse um 1 Bit nach rechts schieben...
. 9f72  66 63    ror $63
. 9f74  66 64    ror $64
. 9f76  66 65    ror $65
. 9f78  66 70    ror $70
. 9f7a  60       rts          ENDE

. 9f7b  a5 66    lda $66      FAC invertieren ...
. 9f7d  49 ff    eor #$ff
. 9f7f  85 66    sta $66
. 9f81  a5 62    lda $62
. 9f83  49 ff    eor #$ff
. 9f85  85 62    sta $62
. 9f87  a5 63    lda $63
. 9f89  49 ff    eor #$ff
. 9f8b  85 63    sta $63
. 9f8d  a5 64    lda $64
. 9f8f  49 ff    eor #$ff
```

```
. 9f91  85 64     sta $64
. 9f93  a5 65     lda $65
. 9f95  49 ff     eor #$ff
. 9f97  85 65     sta $65
. 9f99  a5 70     lda $70
. 9f9b  49 ff     eor #$ff
. 9f9d  85 70     sta $70
. 9f9f  e6 70     inc $70       ... und jedes Byte um 1 erhöhen...
. 9fa1  d0 0e     bne $9fb1
. 9fa3  e6 65     inc $65
. 9fa5  d0 0a     bne $9fb1
. 9fa7  e6 64     inc $64
. 9fa9  d0 06     bne $9fb1
. 9fab  e6 63     inc $63
. 9fad  d0 02     bne $9fb1
. 9faf  e6 62     inc $62
. 9fb1  60        rts           ENDE

. 9fb2  a2 0f     ldx #$0f      Holt Fehlernummer 15 ('OVERFLOW')
. 9fb4  4c 83 86  jmp $8683     Fehlermeldung ausgeben

. 9fb7  :
. 9fb7  *** REGISTER NACH RECHTS VERSCHIEBEN ***
. 9fb7  :
. 9fb7  a2 25     ldx #$25      Holt Offset-Zeiger für Register
. 9fb9  b4 04     ldy $04,x     Register um 1 Byte nach rechts verschieben...
. 9fbb  84 70     sty $70
. 9fbd  b4 03     ldy $03,x
. 9fbf  94 04     sty $04,x
. 9fc1  b4 02     ldy $02,x
. 9fc3  94 03     sty $03,x
. 9fc5  b4 01     ldy $01,x
. 9fc7  94 02     sty $02,x
. 9fc9  a4 68     ldy $68
. 9fcb  94 01     sty $01,x
. 9fcd  69 08     adc #$08      Bit-Zähler um 8 (1 Byte) erhöhen
. 9fcf  30 e8     bmi $9fb9     Bit-Zähler negativ ?, dann $9fb9
. 9fd1  f0 e6     beq $9fb9     Bit-Zähler gleich 0 ?, dann $9fb9
. 9fd3  e9 08     sbc #$08      Bit-Zähler wieder um 8 zurücksetzen
. 9fd5  a8        tay
. 9fd6  a5 70     lda $70       Holt niederwertige Stelle von FAC
. 9fd8  b0 14     bcs $9fee     Gleich 0 ?, dann fertig
. 9fda  16 01     asl $01,x     Höchstes Byte der Mantisse
. 9fdc  90 02     bcc $9fe0     ... nicht gesetzt ?, dann 9fe0
. 9fde  f6 01     inc $01,x     1. Stelle der Mantisse um 1 erhöhen
. 9fe0  76 01     ror $01,x     Mantisse um 1 Bit nach rechts verschieben...
. 9fe2  76 01     ror $01,x
. 9fe4  76 02     ror $02,x
. 9fe6  76 03     ror $03,x
. 9fe8  76 04     ror $04,x
. 9fea  6a        ror
. 9feb  c8        iny           Bit-Zähler um 1 erhöhen
```

```
. 9fec  dØ ec     bne $9fda    Bit-Zähler ungleich Ø ?, dann $9fda
. 9fee  18        clc
. 9fef  6Ø        rts          Sonst ENDE

. 9ffØ  :
. 9ffØ  *** KONSTANTEN ***
. 9ffØ  :
. 9ffØ  .byte $81 $ØØ $ØØ $ØØ $ØØ    1
. 9ff5  .byte $Ø3                    Polynomgrad
. 9ff6  .byte $7f $5e $56 $cb $79    Ø.434255942
. 9ffb  .byte $8Ø $13 $9b $Øb $64    Ø.576584541
. aØØØ  .byte $8Ø $76 $38 $93 $16    Ø.961ØØ0759
. aØØ5  .byte $82 $38 $aa $2b $2Ø    2.88539ØØ7
. aØØa  .byte $8Ø $35 $Ø4 $f3 $34    Ø.7Ø7106781
. aØØf  .byte $81 $35 $Ø4 $f3 $34    1.41421356  (Wurzel 2)
. aØ14  .byte $8Ø $8Ø $ØØ $ØØ $ØØ   -Ø.5
. aØ19  .byte $8Ø $31 $72 $17 $f8    Ø.693147181 (ln 2)

. aØ1e  :
. aØ1e  *** L O G ***
. aØ1e  :
. aØ1e  2Ø bØ a2 jsr $a2bØ    Ermittelt Vorzeichen
. aØ21  fØ Ø2    beq $aØ25    Wenn FAC gleich Ø, dann Fehler
. aØ23  1Ø Ø3    bpl $aØ28    Sonst keine Fehlermeldung
. aØ25  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. aØ28  a5 61    lda $61      Holt Exponent von FAC
. aØ2a  e9 7f    sbc #$7f     Subtrahiert $8Ø (Carry=1)
. aØ2c  48       pha          Exponent auf Stack legen
. aØ2d  a9 8Ø    lda #$8Ø     Setzt Exponent
. aØ2f  85 61    sta $61      ... gleich $8Ø
. aØ31  a9 Øa    lda #$Øa     Setzt Zeiger auf
. aØ33  aØ aØ    ldy #$aØ     ... Konstante Ø.7Ø7106781
. aØ35  2Ø 66 aØ jsr $aØ66    FAC := FAC + Konstante (Ø.7Ø7106781)
. aØ38  a9 Øf    lda #$Øf     Setzt Zeiger auf
. aØ3a  aØ aØ    ldy #$aØ     ... Konstante 1.41421356
. aØ3c  2Ø 72 aØ jsr $aØ72    FAC := Konstante (1.41421356) / FAC
. aØ3f  a9 fØ    lda #$fØ     Setzt Zeiger auf
. aØ41  aØ 9f    ldy #$9f     ... Konstante 1
. aØ43  2Ø 6c aØ jsr $aØ6c    FAC := Konstante - FAC
. aØ46  a9 f5    lda #$f5     Setzt Zeiger auf
. aØ48  aØ 9f    ldy #$9f     ... Polynomgrad
. aØ4a  2Ø b3 a6 jsr $a6b3    Polynom auswerten
. aØ4d  a9 14    lda #$14     Setzt Zeiger auf
. aØ4f  aØ aØ    ldy #$aØ     ... Konstante -Ø.5
. aØ51  2Ø 66 aØ jsr $aØ66    FAC := FAC + Konstante (-Ø.5)
. aØ54  68       pla          Holt Exponent vom Stack
. aØ55  2Ø Øa a4 jsr $a4Øa    Berechnet und addiert Kennzahl
. aØ58  a9 19    lda #$19     Setzt Zeiger auf
. aØ5a  aØ aØ    ldy #$aØ     ... Konstante Ø.693147181
. aØ5c  2Ø dc aØ jsr $aØdc    Schreibt Konstante (Ø.693147181) nach ARG
. aØ5f  4c 7b aØ jmp $aØ7b    FAC := FAC * ARG
```

```
. aØ62  :
. aØ62  *** FAC := FAC + Ø.5 ***
. aØ62  :
. aØ62  a9 a3    lda #$a3      Setzt Zeiger auf
. aØ64  aØ a5    ldy #$a5      ... Konstante Ø.5
. aØ66  2Ø dc aØ jsr $aØdc     Schreibt Konstante (Ø.5) nach ARG
. aØ69  4c 9e 9e jmp $9e9e     FAC := ARG + FAC

. aØ6c  :
. aØ6c  *** FAC := KONSTANTE - FAC ***
. aØ6c  :
. aØ6c  2Ø dc aØ jsr $aØdc     Holt Konstante (Akku, Y-Reg) aus ROM nach ARG
. aØ6f  4c 87 9e jmp $9e87     FAC := ARG - FAC

. aØ72  :
. aØ72  *** FAC := KONSTANTE / FAC ***
. aØ72  :
. aØ72  2Ø dc aØ jsr $aØdc     Holt Konstante (Akku, Y-Reg) aus ROM nach ARG
. aØ75  4c 97 a1 jmp $a197     FAC := ARG / FAC

. aØ78  :
. aØ78  *** FAC := VARIABLE * FAC ***
. aØ78  :
. aØ78  2Ø Ø7 a1 jsr $a1Ø7     Variable von (Akku, Y-Reg) in RAM nach ARG
. aØ7b  :
. aØ7b  *** FAC := ARG * FAC ***
. aØ7b  :
. aØ7b  dØ Ø3    bne $aØ8Ø     FAC ungleich Ø ?, dann $aØ8Ø
. aØ7d  4c db aØ jmp $aØdb     Sonst ENDE

. aØ8Ø  2Ø 37 a1 jsr $a137     Berechnet Exponent
. aØ83  a9 ØØ    lda #$ØØ     Funktionsregister
. aØ85  85 26    sta $26      ... initialisieren...
. aØ87  85 27    sta $27
. aØ89  85 28    sta $28
. aØ8b  85 29    sta $29
. aØ8d  a5 7Ø    lda $7Ø      Holt niederwertige Stelle
. aØ8f  2Ø a9 aØ jsr $aØa9    Multiplizieren
. aØ92  a5 65    lda $65      Holt 4. Stelle
. aØ94  2Ø a9 aØ jsr $aØa9    Multiplizieren
. aØ97  a5 64    lda $64      Holt 3. Stelle
. aØ99  2Ø a9 aØ jsr $aØa9    Multiplizieren
. aØ9c  a5 63    lda $63      Holt 2. Stelle
. aØ9e  2Ø a9 aØ jsr $aØa9    Multiplizieren
. aØa1  a5 62    lda $62      Holt 1. Stelle
. aØa3  2Ø ae aØ jsr $aØae    Multiplizieren
. aØa6  4c Øc a2 jmp $a2Øc    Ergebnis nach FAC und normalisieren

. aØa9  dØ Ø3    bne $aØae
. aØab  4c b7 9f jmp $9fb7    Funktionsergebnis nach rechts schieben
```

```
. aØae  :
. aØae  *** MULTIPLIKATION ***
. aØae  :
. aØae  4a       lsr          Für jedes im Akku
. aØaf  Ø9 8Ø    ora #$8Ø     ... gesetzte Bit
. aØb1  a8       tay          ... wird ARG zum
. aØb2  9Ø 19    bcc $aØcd    ... Funktionsregister
. aØb4  18       clc          ... addiert;
. aØb5  a5 29    lda $29      ... dann wird
. aØb7  65 6d    adc $6d      ... dieses um
. aØb9  85 29    sta $29      ... 1 Bit
. aØbb  a5 28    lda $28      ... nach rechts
. aØbd  65 6c    adc $6c      ... verschoben.
. aØbf  85 28    sta $28      ... Ist
. aØc1  a5 27    lda $27      ... ein
. aØc3  65 6b    adc $6b      ... Bit
. aØc5  85 27    sta $27      ... nicht
. aØc7  a5 26    lda $26      ... gesetzt,
. aØc9  65 6a    adc $6a      ... so
. aØcb  85 26    sta $26      ... wird
. aØcd  66 26    ror $26      ... nur
. aØcf  66 27    ror $27      ... verschoben ...
. aØd1  66 28    ror $28
. aØd3  66 29    ror $29
. aØd5  66 7Ø    ror $7Ø
. aØd7  98       tya
. aØd8  4a       lsr
. aØd9  dØ d6    bne $aØb1
. aØdb  6Ø       rts          ENDE

. aØdc  :
. aØdc  *** (AKKU, Y-REG) AUS ROM NACH ARG ***
. aØdc  :
. aØdc  85 22    sta $22      Speichert Quelladresse
. aØde  84 23    sty $23      ... in Hilfszeiger
. aØeØ  aØ Ø4    ldy #$Ø4     Kopiert Konstante nach ARG...
. aØe2  b1 22    lda ($22),y
. aØe4  85 6d    sta $6d
. aØe6  88       dey
. aØe7  b1 22    lda ($22),y
. aØe9  85 6c    sta $6c
. aØeb  88       dey
. aØec  b1 22    lda ($22),y
. aØee  85 6b    sta $6b
. aØfØ  88       dey
. aØf1  b1 22    lda ($22),y
. aØf3  85 6e    sta $6e
. aØf5  45 66    eor $66
. aØf7  85 6f    sta $6f      Schreibt Vorzeichen von FAC * ARG
. aØf9  a5 6e    lda $6e
. aØfb  Ø9 8Ø    ora #$8Ø
. aØfd  85 6a    sta $6a
```

```
. aØff  88       dey
. a1ØØ  b1 22    lda ($22),y
. a1Ø2  85 69    sta $69
. a1Ø4  a5 61    lda $61       Holt Exponent von FAC
. a1Ø6  6Ø       rts           ENDE

. a1Ø7  :
. a1Ø7  *** (AKKU, Y-REG) AUS RAM NACH ARG ***
. a1Ø7  :
. a1Ø7  85 22    sta $22       Speichert Quelladresse
. a1Ø9  84 23    sty $23       ... in Hilfszeiger
. a1Øb  aØ Ø4    ldy #$Ø4      Kopiert Variable nach ARG...
. a1Ød  2Ø bØ Ø4 jsr $Ø4bØ     Indirekter RAM-Zugriff (LAD($22),Y)
. a11Ø  85 6d    sta $6d
. a112  88       dey
. a113  2Ø bØ Ø4 jsr $Ø4bØ     Indirekter RAM-Zugriff (LAD($22),Y)
. a116  85 6c    sta $6c
. a118  88       dey
. a119  2Ø bØ Ø4 jsr $Ø4bØ     Indirekter RAM-Zugriff (LAD($22),Y)
. a11c  85 6b    sta $6b
. a11e  88       dey
. a11f  2Ø bØ Ø4 jsr $Ø4bØ     Indirekter RAM-Zugriff (LAD($22),Y)
. a122  85 6e    sta $6e
. a124  45 66    eor $66
. a126  85 6f    sta $6f       Schreibt Vorzeichen von FAC * ARG
. a128  a5 6e    lda $6e
. a12a  Ø9 8Ø    ora #$8Ø
. a12c  85 6a    sta $6a
. a12e  88       dey
. a12f  2Ø bØ Ø4 jsr $Ø4bØ     Indirekter RAM-Zugriff (LAD($22),Y)
. a132  85 69    sta $69
. a134  a5 61    lda $61       Holt Exponent von FAC
. a136  6Ø       rts           ENDE

. a137  :
. a137  *** EXPONENTEN ADDIEREN ***
. a137  :
. a137  a5 69    lda $69       Exponent von ARG gleich Ø ?
. a139  fØ 1f    beq $a15a     Ja, dann $a15a
. a13b  18       clc
. a13c  65 61    adc $61       Addiert Exponent von FAC
. a13e  9Ø Ø4    bcc $a144     Kein Überlauf, dann $a144
. a14Ø  3Ø 1d    bmi $a15f     Wenn Überlauf und Erg. größer 127, dann Fehler
. a142  18       clc

. a143  2c       .byte $2c

. a144  1Ø 14    bpl $a15a     Wenn kein Überlauf und Erg. kleiner 128, dann O.K.
. a146  69 8Ø    adc #$8Ø
. a148  85 61    sta $61       Schreibt neuen Exponenten
. a14a  dØ Ø3    bne $a14f     Exponent ungleich Ø, dann $a14f
. a14c  4c 2f 9f jmp $9f2f     FAC := Ø
```

```
. a14f  a5 6f     lda $6f       Holt Vorzeichen von ARG * FAC
. a151  85 66     sta $66       Schreibt Vorzeichen von FAC
. a153  60        rts           ENDE

. a154  a5 66     lda $66       Holt Vorzeichen von FAC
. a156  49 ff     eor #$ff      Invertiert Vorzeichen
. a158  30 05     bmi $a15f     Wenn Ergebnis kleiner 0, dann $a15f
. a15a  68        pla
. a15b  68        pla
. a15c  4c 2b 9f  jmp $9f2b     FAC := 0

. a15f  4c b2 9f  jmp $9fb2     'OVERFLOW ERROR' ausgeben

. a162  :
. a162  *** FAC := FAC * 10 ***
. a162  :
. a162  20 91 a2  jsr $a291     ARG := FAC
. a165  aa        tax           Exponent von FAC gleich 0 ?
. a166  f0 10     beq $a178     Ja, dann FAC := 0
. a168  18        clc
. a169  69 02     adc #$02      Exponent + 2 entspricht FAC * 4
. a16b  b0 f2     bcs $a15f     Überlauf ?, dann Fehler
. a16d  a2 00     ldx #$00
. a16f  86 6f     stx $6f
. a171  20 ab 9e  jsr $9eab     FAC := FAC + ARG
. a174  e6 61     inc $61       Exponent + 1 entspricht FAC * 2
. a176  f0 e7     beq $a15f     Überlauf ?, dann Fehler
. a178  60        rts           ENDE

. a179  .byte $84 $20 $00 $00 $00   Gleitkomma-Konstante 10

. a17e  a2 14     ldx #$14      Holt Fehlernummer 20 ('DIVISION BY ZERO')
. a180  4c 83 86  jmp $8683     Fehlerausgabe

. a183  :
. a183  *** FAC := FAC / 10 ***
. a183  :
. a183  20 91 a2  jsr $a291     FAC runden und ARG := FAC
. a186  a9 79     lda #$79      Setzt Zeiger auf
. a188  a0 a1     ldy #$a1      ... Gleitkomma-Konstante 10
. a18a  a2 00     ldx #$00      Initialisiert
. a18c  86 6f     stx $6f       ... Vorzeichen von FAC
. a18e  20 21 a2  jsr $a221     Gleitkomma-Konstante 10 nach FAC
. a191  4c 97 a1  jmp $a197     FAC := ARG / FAC

. a194  :
. a194  *** FAC := ARG / FAC ***
. a194  :
. a194  20 07 a1  jsr $a107     (Akku, Y-Reg. aus RAM nach ARG
. a197  f0 e5     beq $a17e     Wenn FAC gleich 0, dann $a17e
. a199  20 a0 a2  jsr $a2a0     Rundet FAC
. a19c  a9 00     lda #$00      Wechselt
```

```
. a19e  38       sec          ... Vorzeichen
. a19f  e5 61    sbc $61      ... des Exponenten
. a1a1  85 61    sta $61      ... von FAC
. a1a3  20 37 a1 jsr $a137    Ermittelt Exponent und Vorzeichen
. a1a6  e6 61    inc $61      Erhöht Exponent
. a1a8  f0 b5    beq $a15f    Bei Überlauf Fehler
. a1aa  a2 fc    ldx #$fc     Holt
. a1ac  a9 01    lda #$01     ... Offset-Zeiger
. a1ae  a4 6a    ldy $6a      Vergleicht
. a1b0  c4 62    cpy $62      ... ARG
. a1b2  d0 10    bne $a1c4    ... mit
. a1b4  a4 6b    ldy $6b      ... FAC...
. a1b6  c4 63    cpy $63
. a1b8  d0 0a    bne $a1c4
. a1ba  a4 6c    ldy $6c
. a1bc  c4 64    cpy $64
. a1be  d0 04    bne $a1c4
. a1c0  a4 6d    ldy $6d
. a1c2  c4 65    cpy $65
. a1c4  08       php          Legt Vergleichsergebnis auf Stack
. a1c5  2a       rol
. a1c6  90 09    bcc $a1d1
. a1c8  e8       inx
. a1c9  95 29    sta $29,x    Schreibt Ergebnis in Funktionsregister
. a1cb  f0 32    beq $a1ff    Wenn Offset von X gleich 0, dann $a1ff
. a1cd  10 34    bpl $a203    Wenn Offset von X größer 0, dann $a203
. a1cf  a9 01    lda #$01
. a1d1  28       plp          Holt Vergleichsergebnis vom Stack
. a1d2  b0 0e    bcs $a1e2    Wenn ARG größer/gleich FAC, dann $a1e2
. a1d4  06 6d    asl $6d      ARG um
. a1d6  26 6c    rol $6c      ... 1 Bit
. a1d8  26 6b    rol $6b      ... nach links
. a1da  26 6a    rol $6a      ... verschieben
. a1dc  b0 e6    bcs $a1c4    Bei Überlauf nach $a1c4
. a1de  30 ce    bmi $a1ae    Wenn höchstes Bit gesetzt, dann $a1ae
. a1e0  10 e2    bpl $a1c4    Springt immer nach $a1c4

. a1e2  a8       tay          Subtrahiert
. a1e3  a5 6d    lda $6d      ... Mantisse
. a1e5  e5 65    sbc $65      ... von
. a1e7  85 6d    sta $6d      ... FAC
. a1e9  a5 6c    lda $6c      ... von
. a1eb  e5 64    sbc $64      ... Mantisse
. a1ed  85 6c    sta $6cc     ... von
. a1ef  a5 6b    lda $6b      ... ARG...
. a1f1  e5 63    sbc $63
. a1f3  85 6b    sta $6b
. a1f5  a5 6a    lda $6a
. a1f7  e5 62    sbc $62
. a1f9  85 6a    sta $6a
. a1fb  98       tya
. a1fc  4c d4 a1 jmp $a1d4    Zurück zum Verschieben
```

```
. a1ff  a9 40     lda #$40
. a201  d0 ce     bne $a1d1     Springt immer nach $a1d1

. a203  0a        asl
. a204  0a        asl
. a205  0a        asl
. a206  0a        asl
. a207  0a        asl
. a208  0a        asl
. a209  85 70     sta $70       Schreibt niederwertige Stelle von FAC
. a20b  28        plp
. a20c  a5 26     lda $26       Ergebnis
. a20e  85 62     sta $62       ... nach
. a210  a5 27     lda $27       ... FAC
. a212  85 63     sta $63       ... kopieren...
. a214  a5 28     lda $28
. a216  85 64     sta $64
. a218  a5 29     lda $29
. a21a  85 65     sta $65
. a21c  4c 0b 9f  jmp $9f0b     FAC linksbündig normalisieren

. a21f  :
. a21f  *** FAC := (Akku, Y-Reg) ***
. a21f  :
. a21f  18        clc           Carry=0 ist Flag für "Laden aus RAM"

. a220  24        .byte $24

. a221  38        sec           Carry=1 ist Flag für "Laden aus ROM"
. a222  85 22     sta $22       Schreibt Quelladresse
. a224  84 23     sty $23       ... in Hilfszeiger
. a226  a0 04     ldy #$04
. a228  20 20 a3  jsr $a320     Holt 4. Stelle (LDA($22),Y)
. a22b  85 65     sta $65       Schreibt in FAC
. a22d  88        dey
. a22e  20 20 a3  jsr $a320     Holt 3. Stelle (LDA($22),Y)
. a231  85 64     sta $64       Schreibt in FAC
. a233  88        dey
. a234  20 20 a3  jsr $a320     Holt 2. Stelle (LDA($22),Y)
. a237  85 63     sta $63       Schreibt in FAC
. a239  88        dey
. a23a  20 20 a3  jsr $a320     Holt Vorzeichen und 1. Stelle (LDA($22),Y)
. a23d  85 66     sta $66       Schreibt Vorzeichenbyte
. a23f  09 80     ora #$80
. a241  85 62     sta $62       Schreibt 1. Stelle in FAC
. a243  88        dey
. a244  20 20 a3  jsr $a320     Holt Exponent (LDA($22),Y)
. a247  85 61     sta $61       Schreibt Vorzeichenbyte
. a249  84 70     sty $70       Setzt niederwertige Stelle gleich 0
. a24b  60        rts           ENDE
```

```
. a24c  :
. a24c  *** FAC IN SPEICHER KOPIEREN ***
. a24c  :
. a24c  a2 5c     ldx #$5c      Zielbereich $5c-$60

. a24e  2c        .byte $2c

. a24f  a2 57     ldx #$57      Zielbereich $57-$5b
. a251  a0 00     ldy #$00
. a253  f0 04     beq $a259

. a255  a6 49     ldx $49       Holt Startadresse
. a257  a4 4a     ldy $4a       ... des Zielbereichs
. a259  20 a0 a2  jsr $a2a0     FAC runden
. a25c  86 22     stx $22       In Hilfszeiger
. a25e  84 23     sty $23       ... schreiben
. a260  a0 04     ldy #$04
. a262  a5 65     lda $65       Holt 4. Stelle
. a264  91 22     sta ($22),y   Schreibt 4. Stelle in Zielbereich
. a266  88        dey
. a267  a5 64     lda $64       Holt 3. Stelle
. a269  91 22     sta ($22),y   Schreibt 3. Stelle in Zielbereich
. a26b  88        dey
. a26c  a5 63     lda $63       Holt 2. Stelle
. a26e  91 22     sta ($22),y   Schreibt 2. Stelle in Zielbereich
. a270  88        dey
. a271  a5 66     lda $66       Holt Vorzeichenbyte
. a273  09 7f     ora #$7f
. a275  25 62     and $62       Und-Verknüpfung mit 1. Stelle
. a277  91 22     sta ($22),y   Schreibt Vorzeichen und 1. Stelle in Zielber.
. a279  88        dey
. a27a  a5 61     lda $61       Holt Exponent
. a27c  91 22     sta ($22),y   Schreibt Exponent in Zielbereich
. a27e  84 70     sty $70       Setzt niederwertige Stelle gleich 0
. a280  60        rts           ENDE

. a281  :
. a281  *** FAC := ARG ***
. a281  :
. a281  a5 6e     lda $6e       Kopiert
. a283  85 66     sta $66       ... Vorzeichen
. a285  a2 05     ldx #$05      Kopiert
. a287  b5 68     lda $68,x     ... Mantisse...
. a289  95 60     sta $60,x
. a28b  ca        dex
. a28c  d0 f9     bne $a287
. a28e  86 70     stx $70       Setzt niederwertige Stelle gleich 0
. a290  60        rts ENDE

. a291  :
. a291  *** ARG := FAC ***
. a291  :
```

```
. a291  20 a0 a2 jsr $a2a0     FAC runden
. a294  a2 06    ldx #$06      FAC
. a296  b5 60    lda $60,x     ... nach
. a298  95 68    sta $68,x     ... ARG
. a29a  ca       dex           ... kopieren...
. a29b  d0 f9    bne $a296
. a29d  86 70    stx $70       Setzt niederwertige Stelle gleich 0
. a29f  60       rts           ENDE

. a2a0  :
. a2a0  *** FAC RUNDEN ***
. a2a0  :
. a2a0  a5 61    lda $61       FAC gleich 0 ?
. a2a2  f0 fb    beq $a29f     Ja, dann fertig
. a2a4  06 70    asl $70       Niederwertige Stelle um 1 Bit nach links versch.
. a2a6  90 f7    bcc $a29f     Kleiner 128 ?, dann fertig
. a2a8  20 a3 9f jsr $9fa3     Mantisse um 1 erhöhen
. a2ab  d0 f2    bne $a29f     Ergebnis ungleich 0, dann fertig
. a2ad  4c 6c 9f jmp $9f6c     FAC normalisieren

. a2b0  :
. a2b0  *** VORZEICHEN ERMITTELN ***
. a2b0  :
. a2b0  a5 61    lda $61       FAC gleich 0 ?
. a2b2  f0 09    beq $a2bd     Ja, dann fertig
. a2b4  a5 66    lda $66       Holt Vorzeichenbyte
. a2b6  2a       rol           Wenn negativ, dann Carry = 1
. a2b7  a9 ff    lda #$ff
. a2b9  b0 02    bcs $a2bd     Wenn negativ, dann Akku gleich 255
. a2bb  a9 01    lda #$01      Wenn positiv, dann Akku gleich 1
. a2bd  60       rts           ENDE

. a2be  :
. a2be  *** S G N ***
. a2be  :
. a2be  20 b0 a2 jsr $a2b0     Vorzeichen ermitteln
. a2c1  85 62    sta $62       1. Stelle gleich 0,1 oder 255 (je nach Vorzeichen)
. a2c3  a9 00    lda #$00
. a2c5  85 63    sta $63       2. Stelle gleich 0
. a2c7  a2 88    ldx #$88      Holt Exponent für Byte-Wert
. a2c9  a5 62    lda $62       Holt 1. Stelle
. a2cb  49 ff    eor #$ff      1. Stelle invertieren
. a2cd  2a       rol           0 und positiv setzt Carry
. a2ce  a9 00    lda #$00
. a2d0  85 65    sta $65       4. Stelle gleich 0
. a2d2  85 64    sta $64       3. Stelle gleich 0
. a2d4  86 61    stx $61       Setzt Exponent für Byte-Wert
. a2d6  85 70    sta $70       Schreibt niederwertige Stelle
. a2d8  85 66    sta $66       Schreibt Vorzeichenbyte
. a2da  4c 06 9f jmp $9f06     Mantisse negieren, wenn Carry=0; sonst normalis.
. a2dd  :
. a2dd  *** A B S ***
```

```
. a2dd  :
. a2dd  46 66     lsr $66       Bit 7 von Vorzeichen löschen
. a2df  60        rts           ENDE

. a2e0  :
. a2e0  *** ZAHLENVERGLEICH ***
. a2e0  :
. a2e0  85 24     sta $24       Holt Adresse
. a2e2  84 25     sty $25       ... der Vergleichszahl
. a2e4  a0 00     ldy #$00
. a2e6  b1 24     lda ($24),y   Holt Exponent der Vergleichszahl (LDA($24),Y)
. a2e8  c8        iny
. a2e9  aa        tax
. a2ea  f0 c4     beq $a2b0     Wenn Exponent gleich 0, dann auch Zahl gleich 0
. a2ec  b1 24     lda ($24),y   Holt 1. Stelle der Vergleichszahl
. a2ee  45 66     eor $66       Exclusiv-Oder-Verknüpfung mit Vorzeichen von FAC
. a2f0  30 c2     bmi $a2b4     Wenn Vorzeichen verschieden, dann $a2b4
. a2f2  e4 61     cpx $61       Exponenten gleich ?
. a2f4  d0 21     bne $a317     Nein, dann $a317
. a2f6  b1 24     lda ($24),y   Holt 1. Stelle der Vergleichszahl
. a2f8  09 80     ora #$80      Setzt Vorzeichenbit
. a2fa  c5 62     cmp $62       Gleich 1. Stelle von FAC ?
. a2fc  d0 19     bne $a317     Nein, dann $a317
. a2fe  c8        iny
. a2ff  b1 24     lda ($24),y   Ist 2. Stelle der Vergleichszahl
. a301  c5 63     cmp $63       ... gleich 2. Stelle von FAC ?
. a303  d0 12     bne $a317     Nein, dann $a317
. a305  c8        iny
. a306  b1 24     lda ($24),y   Ist 3. Stelle der Vergleichszahl
. a308  c5 64     cmp $64       ... gleich 3. Stelle von FAC ?
. a30a  d0 0b     bne $a317     Nein, dann $a317
. a30c  c8        iny
. a30d  a9 7f     lda #$7f      Wenn niederwertige Stelle
. a30f  c5 70     cmp $70       ... kleiner 128, dann Carry = 1
. a311  b1 24     lda ($24),y   Holt 4. Stelle von Vergleichszahl
. a313  e5 65     sbc $65       ... und subtrahiert 4. Stelle von FAC
. a315  f0 2f     beq $a346     Wenn Erg. gleich 0, dann beide Zahlen gleich
. a317  a5 66     lda $66       Holt Vorzeichen von FAC
. a319  90 02     bcc $a31d     Wenn Vergleichszahl kleiner, dann $a31d
. a31b  49 ff     eor #$ff
. a31d  4c b6 a2  jmp $a2b6     Ergebnis nach Akku

. a320  b1 22     lda ($22),y   Aus ROM laden
. a322  b0 22     bcs $a346     Wenn Carry = 1, dann fertig
. a324  4c b0 04  jmp $04b0     Aus RAM laden (LDA($22),Y)

. a327  :
. a327  *** GLEITKOMMA - INTEGER - WANDLUNG ***
. a327  :
. a327  a5 61     lda $61       FAC gleich 0 ?
. a329  f0 4a     beq $a375     Ja, dann $a375
. a32b  38        sec
```

```
. a32c  e9 a0     sbc #$a0      Exponent transformieren
. a32e  24 66     bit $66       FAC negativ ?
. a330  10 09     bpl $a33b     Nein, dann $a33b
. a332  aa        tax           Exponent - $a0
. a333  a9 ff     lda #$ff      Setzt Überlauf
. a335  85 68     sta $68       ... auf $ff
. a337  20 81 9f  jsr $9f81     FAC negieren
. a33a  8a        txa
. a33b  a2 61     ldx #$61      Lädt Offset-Zeiger für FAC
. a33d  c9 f9     cmp #$f9      Exponent - $a0 größer -8 ?
. a33f  10 06     bpl $a347     Ja, dann $a347
. a341  20 cd 9f  jsr $9fcd     Schiebt FAC nach rechts
. a344  84 68     sty $68       Setzt Überlauf gleich 0
. a346  60        rts           ENDE

. a347  a8        tay           Exponent - $a0
. a348  a5 66     lda $66       Holt Vorzeichenbyte von FAC
. a34a  29 80     and #$80      Isoliert Bit 7
. a34c  46 62     lsr $62       Schiebt 1. Stelle 1 Bit nach links
. a34e  05 62     ora $62       Bit 7 einordnen
. a350  85 62     sta $62       Ergebnis gleich 1. Stelle
. a352  20 e4 9f  jsr $9fe4     Verschiebt FAC bitweise nach rechts
. a355  84 68     sty $68       Setzt Überlauf gleich 0
. a357  60        rts           ENDE

. a358  :
. a358  *** I N T ***
. a358  :
. a358  a5 61     lda $61       Holt Exponent von FAC
. a35a  c9 a0     cmp #$a0      Exponent kleiner 160 ?
. a35c  b0 20     bcs $a37e     Nein, dann ist FAC ganzzahlig
. a35e  20 27 a3  jsr $a327     Gleitkomma - Integer - Wandlung
. a361  84 70     sty $70       Setzt niederwertige Stelle gleich 0
. a363  a5 66     lda $66       Holt Vorzeichenbyte
. a365  84 66     sty $66       Setzt Vorzeichenbyte gleich 0
. a367  49 80     eor #$80      Nicht negativ
. a369  2a        rol           Carry = 1
. a36a  a9 a0     lda #$a0      Setzt Exponent
. a36c  85 61     sta $61       ... für Integer ($a0)
. a36e  a5 65     lda $65       Holt 4. Stelle von FAC
. a370  85 07     sta $07       Für Potenzier-Routine merken
. a372  4c 06 9f  jmp $9f06     FAC normalisieren

. a375  85 62     sta $62       1. Stelle von FAC gleich 0
. a377  85 63     sta $63       2. Stelle von FAC gleich 0
. a379  85 64     sta $64       3. Stelle von FAC gleich 0
. a37b  85 65     sta $65       4. Stelle von FAC gleich 0
. a37d  a8        tay           Y-Reg. gleich 0
. a37e  60        rts           ENDE

. a37f  :
. a37f  *** STRING - GLEITKOMMA - WANDLUNG ***
```

```
. a37f  :
. a37f  a0 00     ldy #$00      Bereich $5d-$67
. a381  a2 0a     ldx #$0a      ... löschen...
. a383  94 5d     sty $5d,x
. a385  ca        dex
. a386  10 fb     bpl $a383
. a388  90 0f     bcc $a399     Holt Ziffer in Akku
. a38a  c9 2d     cmp #$2d      Zeichen gleich Minuszeichen (-) ?
. a38c  d0 04     bne $a392     Nein, dann $a392
. a38e  86 67     stx $67       Zeiger für Polynom-Auswertung gleich $ff
. a390  f0 04     beq $a396     Springt immer nach $a396

. a392  c9 2b     cmp #$2b      Zeichen gleich Pluszeichen (+) ?
. a394  d0 05     bne $a39b     Nein, dann $a39b
. a396  20 73 04  jsr $0473     CHRGET-Routine
. a399  90 5b     bcc $a3f6     Zeichen gleich Ziffer ?, dann $a3f6
. a39b  c9 2e     cmp #$2e      Zeichen gleich Komma (.) ?
. a39d  f0 2e     beq $a3cd     Ja, dann $a3cd
. a39f  c9 45     cmp #$45      Zeichen gleich e (Exponential-Darstellung) ?
. a3a1  d0 30     bne $a3d3     Nein, dann $a3d3
. a3a3  20 73 04  jsr $0473     CHRGET-Routine
. a3a6  90 17     bcc $a3bf     Zeichen gleich Ziffer ?, dann $a3bf
. a3a8  c9 ab     cmp #$ab      Zeichen gleich Minus-Token (-) ?
. a3aa  f0 0e     beq $a3ba     Ja, dann $a3ba
. a3ac  c9 2d     cmp #$2d      Zeichen gleich Minuszeichen (-) ?
. a3ae  f0 0a     beq $a3ba     Ja, dann $a3ba
. a3b0  c9 aa     cmp #$aa      Zeichen gleich Plus-Token (+) ?
. a3b2  f0 08     beq $a3bc     Ja, dann $a3bc
. a3b4  c9 2b     cmp #$2b      Zeichen gleich Pluszeichen (+) ?
. a3b6  f0 04     beq $a3bc     Ja, dann $a3bc
. a3b8  d0 07     bne $a3c1     Nein, dann $a3c1

. a3ba  66 60     ror $60       Setzt Bit 7
. a3bc  20 73 04  jsr $0473     CHRGET-Routine
. a3bf  90 5c     bcc $a41d     Zeichen gleich Ziffer ?, dann $a41d
. a3c1  24 60     bit $60       Prüft, ob Bit 7 gesetzt
. a3c3  10 0e     bpl $a3d3     Nein, dann $a3d3
. a3c5  a9 00     lda #$00
. a3c7  38        sec
. a3c8  e5 5e     sbc $5e
. a3ca  4c d5 a3  jmp $a3d5     Sprung nach $a3d5

. a3cd  66 5f     ror $5f       Aufruf durch Dezimalpunkt...
. a3cf  24 5f     bit $5f
. a3d1  50 c3     bvc $a396
. a3d3  a5 5e     lda $5e
. a3d5  38        sec
. a3d6  e5 5d     sbc $5d
. a3d8  85 5e     sta $5e
. a3da  f0 12     beq $a3ee
. a3dc  10 09     bpl $a3e7
. a3de  20 83 a1  jsr $a183     FAC := FAC / 10
```

```
. a3e1  e6 5e     inc $5e
. a3e3  d0 f9     bne $a3de
. a3e5  f0 07     beq $a3ee
. a3e7  20 62 a1  jsr $a162    FAC := FAC * 10
. a3ea  c6 5e     dec $5e
. a3ec  d0 f9     bne $a3e7
. a3ee  a5 67     lda $67
. a3f0  30 01     bmi $a3f3
. a3f2  60        rts          Zurück

. a3f3  4c 27 a6  jmp $a627    Vorzeichenwechsel von FAC

. a3f6  48        pha          Aufruf durch Mantissen-Ziffer...
. a3f7  24 5f     bit $5f
. a3f9  10 02     bpl $a3fd
. a3fb  e6 5d     inc $5d
. a3fd  20 62 a1  jsr $a162    FAC := FAC * 10
. a400  68        pla
. a401  38        sec
. a402  e9 30     sbc #$30
. a404  20 0a a4  jsr $a40a    Addiert Stelle zu FAC
. a407  4c 96 a3  jmp $a396    Nächstes Zeichen

. a40a  48        pha
. a40b  20 91 a2  jsr $a291    ARG := FAC
. a40e  68        pla
. a40f  20 c1 a2  jsr $a2c1    1. Stelle von FAC gleich Akku
. a412  a5 6e     lda $6e      Holt Vorzeichen von ARG
. a414  45 66     eor $66      Verknüpfen mit Vorzeichen von FAC
. a416  85 6f     sta $6f      Schreibt Vorzeichen von ARG * FAC
. a418  a6 61     ldx $61      Holt Exponent von FAC
. a41a  4c 9e 9e  jmp $9e9e    FAC := FAC + ARG

. a41d  a5 5e     lda $5e      Aufruf durch Exponent-Ziffer...
. a41f  c9 0a     cmp #$0a
. a421  90 09     bcc $a42c
. a423  a9 64     lda #$64
. a425  24 60     bit $60
. a427  30 16     bmi $a43f
. a429  4c b2 9f  jmp $9fb2    'OVERFLOW ERROR' ausgeben

. a42c  0a        asl
. a42d  0a        asl
. a42e  18        clc
. a42f  65 5e     adc $5e
. a431  0a        asl
. a432  18        clc
. a433  a0 00     ldy #$00
. a435  85 78     sta $78
. a437  20 a5 04  jsr $04a5    Indirekter Speicherzugriff (LDA($3b),Y)
. a43a  65 78     adc $78
. a43c  38        sec
```

```
. a43d  e9 30     sbc #$30
. a43f  85 5e     sta $5e
. a441  4c bc a3  jmp $a3bc    Nächstes Zeichen

. a444  :
. a444  *** KONSTANTEN ***
. a444  :
. a444  .byte $9b $3e $bc $1f $fd     99999999,9
. a449  .byte $9e $6e $6d $27 $fd    999999999
. a44e  .byte $9e $6e $6b $28 $00   1000000000

. a453  :
. a453  *** ZEILENNUMMER AUSGEBEN ***
. a453  :
. a453  20 4f ff  jsr $ff4f    " IN " ausgeben

. a456  .byte ' IN '
. a45a  .byte $00

. a45b  a5 3a     lda $3a      Holt laufende
. a45d  a6 39     ldx $39      ... Basic-Zeilennummer nach Akku und X-Reg.
. a45f  :
. a45f  *** INTEGERWERT AUF (AKKU,Y-REG.) AUSGEBEN ***
. a45f  :
. a45f  85 62     sta $62      Schreibt Akku in 1. Stelle von FAC
. a461  86 63     stx $63      Schreibt X-Reg. in 2. Stelle von FAC
. a463  a2 90     ldx #$90     Holt Exponent für INTEGER
. a465  38        sec          Keine Negierung der Mantisse
. a466  20 ce a2  jsr $a2ce    Füllt FAC mit Nullen auf
. a469  20 71 a4  jsr $a471    Gleitkomma-String-Wandlung
. a46c  4c 88 90  jmp $9088    String ausgeben

. a46f  :
. a46f  *** GLEITKOMMA - STRING - WANDLUNG ***
. a46f  :
. a46f  a0 01     ldy #$01     Offset-Zeiger im Stringpuffer auf 1. Zeichen
. a471  a9 20     lda #$20     Holt Leerzeichen
. a473  24 66     bit $66      Ist Vorzeichen von FAC negativ ?
. a475  10 02     bpl $a479    Nein, dann $a479
. a477  a9 2d     lda #$2d     Holt Bindestrich (-)
. a479  99 ff 00  sta $00ff,y  Schreibt Zeichen in Stringpuffer
. a47c  85 66     sta $66      Schreibt Vorzeichen von FAC
. a47e  84 71     sty $71      Schreibt Offset-Zeiger
. a480  c8        iny          Offset-Zeiger erhöhen
. a481  a9 30     lda #$30     Holt Ziffer Null (0)
. a483  a6 61     ldx $61      Holt Exponent von FAC
. a485  d0 03     bne $a48a    Exponent ungleich Null ?, dann $a48a
. a487  4c 96 a5  jmp $a596    Schreibt '0' in Puffer und ENDE

. a48a  a9 00     lda #$00
. a48c  e0 80     cpx #$80     Holt Exponent von FAC
. a48e  f0 02     beq $a492    Wenn FAC zwischen 0.5 und 1, dann $a492
```

```
. a490  b0 09    bcs $a49b    Wenn FAC größer-gleich 1. dann $a49b
. a492  a9 4e    lda #$4e     Setzt Zeiger auf
. a494  a0 a4    ldy #$a4     ... Konstante 1000000000
. a496  20 5c a0 jsr $a05c    FAC = FAC * Konstante (* 1000000000)
. a499  a9 f7    lda #$f7     Akku gleich -9
. a49b  85 5d    sta $5d      -9 für FAC kleiner 1, sonst 0
. a49d  a9 49    lda #$49     Setzt Zeiger auf
. a49f  a0 a4    ldy #$a4     ... Konstante 999999999
. a4a1  20 e0 a2 jsr $a2e0    Vergleicht FAC mit Konstante
. a4a4  f0 1e    beq $a4c4    Wenn FAC gleich 999999999, dann $a4c4
. a4a6  10 12    bpl $a4ba    Wenn FAC größer 999999999, dann $a4ba
. a4a8  a9 44    lda #$44     Setzt Zeiger auf
. a4aa  a0 a4    ldy #$a4     ... Konstante 99999999,9
. a4ac  20 e0 a2 jsr $a2e0    Vergleicht FAC mit Konstante
. a4af  f0 02    beq $a4b3    Wenn FAC gleich 99999999.9, dann $a4b3
. a4b1  10 0e    bpl $a4c1    Wenn FAC größer 99999999.9, dann $a4c1
. a4b3  20 62 a1 jsr $a162    FAC := FAC * 10
. a4b6  c6 5d    dec $5d
. a4b8  d0 ee    bne $a4a8    Springt immer nach $a4a8

. a4ba  20 83 a1 jsr $a183    FAC := FAC * 10
. a4bd  e6 5d    inc $5d
. a4bf  d0 dc    bne $a49d    Springt immer nach $a49d

. a4c1  20 62 a0 jsr $a062    FAC := FAC + 0,5
. a4c4  20 27 a3 jsr $a327    Gleitkomma - Integer - Wandlung
. a4c7  a2 01    ldx #$01
. a4c9  a5 5d    lda $5d      FAC wurde in den Bereich 1e+08,...,
. a4cb  18       clc          ... 1e+09 transformiert; Adresse $5d enthält
. a4cc  69 0a    adc #$0a     ... den korrigierten Exponenten
. a4ce  30 09    bmi $a4d9    Wenn Betrag der Zahl kleiner 0.01, dann $a4d9
. a4d0  c9 0b    cmp #$0b
. a4d2  b0 06    bcs $a4da    Wenn Betrag der Zahl größer 1e+09, dann $a4da
. a4d4  69 ff    adc #$ff
. a4d6  aa       tax
. a4d7  a9 02    lda #$02
. a4d9  38       sec
. a4da  e9 02    sbc #$02
. a4dc  85 5e    sta $5e      Exponent, wenn Betrag zwischen 0.01 und 1e+09
  a4de  86 5d    stx $5d
. a4e0  8a       txa
. a4e1  f0 02    beq $a4e5    Wenn Betrag zwischen 0.1 und 1, dann $a4e5
. a4e3  10 13    bpl $a4f8    Wenn Betrag nicht zwischen 0.01 und 0.1, dann $a4f8
. a4e5  a4 71    ldy $71
. a4e7  a9 2e    lda #$2e     Holt Punkt (.)
. a4e9  c8       iny
. a4ea  99 ff 00 sta $00ff,y  Schreibt Punkt in Stringpuffer
. a4ed  8a       txa
. a4ee  f0 06    beq $a4f6    Wenn Betrag zwischen 0.1 und 1, dann $a4f6
. a4f0  a9 30    lda #$30     Holt Ziffer Null (0)
. a4f2  c8       iny
. a4f3  99 ff 00 sta $00ff,y  Schreibt 0 in Stringpuffer
```

```
. a4f6  84 71     sty $71
. a4f8  a0 00     ldy #$00      Holt Zeiger in Konstanten-Tabelle ($a5a8)
. a4fa  a2 80     ldx #$80
. a4fc  a5 65     lda $65       Durch abwechselndes
. a4fe  18        clc           ... Addieren
. a4ff  79 ab a5  adc $a5ab,y   ... und
. a502  85 65     sta $65       ... Subtrahieren
. a504  a5 64     lda $64       ... der entsprechenden
. a506  79 aa a5  adc $a5aa,y   ... Stellenwerte
. a509  85 64     sta $64       ... aus der
. a50b  a5 63     lda $63       ... Tabelle ($a5a8)
. a50d  79 a9 a5  adc $a5a9,y   ... werden die
. a510  85 63     sta $63       ... einzelnen
. a512  a5 62     lda $62       ... Ziffern
. a514  79 a8 a5  adc $a5a8,y   ... des
. a517  85 62     sta $62       ... Zahlenstrings
. a519  e8        inx           ... nacheinander
. a51a  b0 04     bcs $a520     ... berechnet
. a51c  10 de     bpl $a4fc
. a51e  30 02     bmi $a522

. a520  30 da     bmi $a4fc
. a522  8a        txa
. a523  90 04     bcc $a529     Komplement noch nicht addiert ?, dann $a529
. a525  49 ff     eor #$ff      Ergebnis bezüglich
. a527  69 0a     adc #$0a      ... 10 komplementieren
. a529  69 2f     adc #$2f      Ergebnis: Zifferncode
. a52b  c8        iny           Tabellenzeiger
. a52c  c8        iny           ... auf
. a52d  c8        iny           ... nächsten
. a52e  c8        iny           ... String
. a52f  84 47     sty $47
. a531  a4 71     ldy $71
. a533  c8        iny
. a534  aa        tax
. a535  29 7f     and #$7f      Schreibt Zifferncode
. a537  99 ff 00  sta $00ff,y   ... in Stringpuffer
. a53a  c6 5d     dec $5d
. a53c  d0 06     bne $a544     Letzte Stelle noch nicht erreicht ?, dann $a544
. a53e  a9 2e     lda #$2e      Holt Punkt-Zeichen (.)
. a540  c8        iny
. a541  99 ff 00  sta $00ff,y   Schreibt Punkt in Stringpuffer
. a544  84 71     sty $71
. a546  a4 47     ldy $47
. a548  8a        txa
. a549  49 ff     eor #$ff
. a54b  29 80     and #$80
. a54d  aa        tax
. a54e  c0 24     cpy #$24      Ende der Gleitkomma-String-Wandlung ?
. a550  f0 04     beq $a556     Ja, dann $a556
. a552  c0 3c     cpy #$3c      Ende der TI$-Erzeugung ?
. a554  d0 a6     bne $a4fc     Nein, dann $a4fc
```

```
. a556  a4 71    ldy $71
. a558  b9 ff ØØ lda $ØØff,y  Holt letzte Stelle aus Stringspeicher
. a55b  88       dey
. a55c  c9 3Ø    cmp #$3Ø     Zeichen gleich Null ?
. a55e  fØ f8    beq $a558    Ja, dann weiter rückwärts suchen
. a56Ø  c9 2e    cmp #$2e     Zeichen gleich Komma-Zeichen (.) ?
. a562  fØ Ø1    beq $a565    Ja, dann $a565
. a564  c8       iny          Offset-Zeiger wieder erhöhen
. a565  a9 2b    lda #$2b     Holt Plus-Zeichen (+)
. a567  a6 5e    ldx $5e      Betrag zwischen Ø.Ø1 und 1e+Ø9 ?
. a569  fØ 2e    beq $a599    Ja, dann $a599
. a56b  1Ø Ø8    bpl $a575    Zehnerexponent positiv
. a56d  a9 ØØ    lda #$ØØ
. a56f  38       sec
. a57Ø  e5 5e    sbc $5e
. a572  aa       tax
. a573  a9 2d    lda #$2d     Holt Minus-Zeichen (-)
. a575  99 Ø1 Ø1 sta $Ø1Ø1,y  Schreibt Zeichen in Stringpuffer
. a578  a9 45    lda #$45     Holt 'E'-Zeichen (für Exponent)
. a57a  99 ØØ Ø1 sta $Ø1ØØ,y  Schreibt 'E' in Stringpuffer
. a57d  8a       txa          Generiert Stelle
. a57e  a2 2f    ldx #$2f     ... des Exponenten
. a58Ø  38       sec
. a581  e8       inx
. a582  e9 Øa    sbc #$Øa
. a584  bØ fb    bcs $a581
. a586  69 3a    adc #$3a     (Einerstelle)
. a588  99 Ø3 Ø1 sta $Ø1Ø3,y
. a58b  8a       txa          (Zehnerstelle)
. a58c  99 Ø2 Ø1 sta $Ø1Ø2,y
. a58f  a9 ØØ    lda #$ØØ     (Endmarke)
. a591  99 Ø4 Ø1 sta $Ø1Ø4,y
. a594  fØ Ø8    beq $a59e    Springt immer nach $a59e

. a596  99 ff ØØ sta $ØØff,y  Schreibt Zeichen in Stringpuffer
. a599  a9 ØØ    lda #$ØØ
. a59b  99 ØØ Ø1 sta $Ø1ØØ,y
. a59e  a9 ØØ    lda #$ØØ     Holt Startadresse des
. a5aØ  aØ Ø1    ldy #$Ø1     ... Stringpuffers nach (Akku, Y-Reg.)
. a5a2  6Ø       rts          ENDE
. a5a3  :
. a5a3  *** KONSTANTEN ***
. a5a3  :
. a5a3  .byte $8Ø $ØØ $ØØ $ØØ $ØØ   Ø.5

. a5a8  :
. a5a8  *** KONSTANTEN FÜR GLEITKOMMA - STRING - WANDLUNG ***
. a5a8  :
. a5a8  .byte $fa $Øa $1f $ØØ -1ØØ ØØØ ØØØ
. a5ac  .byte $ØØ $98 $96 $8Ø   1Ø ØØØ ØØØ
. a5bØ  .byte $ff $fØ $bd $cØ   -1 ØØØ ØØØ
. a5b4  .byte $ØØ $Ø1 $86 $aØ      1ØØ ØØØ
```

```
. a5b8  .byte $ff $ff $d8 $f0      -10 000
. a5bc  .byte $00 $00 $03 $e8        1 000
. a5c0  .byte $ff $ff $ff $9c         -100
. a5c4  .byte $00 $00 $00 $0a           10
. a5c8  .byte $ff $ff $ff $ff           -1

. a5cc  :
. a5cc  *** KONSTANTEN FÜR ERZEUGUNG VON TI$ ***
. a5cc  :
. a5cc  .byte $ff $df $0a $80 -2 160 000
. a5d0  .byte $00 $03 $4b $c0    216 000
. a5d4  .byte $ff $ff $73 $60    -36 000
. a5d8  .byte $00 $00 $0e $10      3 600
. a5dc  .byte $ff $ff $fd $a8       -600
. a5e0  .byte $00 $00 $00 $3c         60

. a5e4  :
. a5e4  *** S Q R ***
. a5e4  :
. a5e4  20 91 a2 jsr $a291   FAC runden und ARG := FAC
. a5e7  a9 a3    lda #$a3    Setzt Zeiger auf
. a5e9  a0 a5    ldy #$a5    ... Konstante (0.5)
. a5eb  20 21 a2 jsr $a221   Konstante (0.5) nach FAC
. a5ee  :
. a5ee  *** FAC := ARG HOCH FAC ***
. a5ee  :
. a5ee  f0 70    beq $a660   Wenn FAC gleich Null, dann $a660
. a5f0  a5 69    lda $69     Ist ARG gleich Null ?
. a5f2  d0 03    bne $a5f7   Nein, dann $a5f7
. a5f4  4c 2d 9f jmp $9f2d   FAC := 0

. a5f7  a2 4e    ldx #$4e    Setzt Zeiger auf
. a5f9  a0 00    ldy #$00    ... Zeropage-Bereich $004e (Arbeitsbereich)
. a5fb  20 59 a2 jsr $a259   Kopiert FAC nach $004e
. a5fe  a5 6e    lda $6e     Holt Vorzeichenbyte von FAC
. a600  10 0f    bpl $a611   Vorzeichen positiv ?, dann $a611
. a602  20 58 a3 jsr $a358   INT (schneidet Nachkommastellen ab)
. a605  a9 4e    lda #$4e    Setzt Zeiger auf
. a607  a0 00    ldy #$00    ... Adresse $004e
. a609  20 e0 a2 jsr $a2e0   Vergleicht FAC mit INT(FAC)
. a60c  d0 03    bne $a611   Ungleich ?, dann $a611
. a60e  98       tya         Akku gleich 4
. a60f  a4 07    ldy $07     Holt letzte Stelle des Exponenten
. a611  20 83 a2 jsr $a283   FAC := Betrag von ARG
. a614  98       tya
. a615  48       pha         Letzte Exponentenstelle auf Stack
. a616  20 1e a0 jsr $a01e   FAC := ln(FAC)
. a619  a9 4e    lda #$4e    Setzt Zeiger auf
. a61b  a0 00    ldy #$00    ... Adresse $004e
. a61d  20 78 a0 jsr $a078   FAC := FAC * Konstante
. a620  20 60 a6 jsr $a660   FAC := exp(FAC)
. a623  68       pla         Holt letzte Exponentenstelle vom Stack
```

```
. a624  4a       lsr          Exponent geradzahlig ?
. a625  90 0a    bcc $a631    Ja, dann $a631
. a627  :
. a627  *** VORZEICHENWECHSEL VON FAC ***
. a627  :
. a627  a5 61    lda $61      FAC gleich Null ?
. a629  f0 06    beq $a631    Ja, dann ENDE
. a62b  a5 66    lda $66      Holt Vorzeichen
. a62d  49 ff    eor #$ff     Invertiert Vorzeichen
. a62f  85 66    sta $66      Schreibt Vorzeichen zurück
. a631  60       rts          ENDE

. a632  :
. a632  *** KONSTANTEN FÜR EXP ***
. a632  :
. a632  .byte $81 $38 $aa $3b $29  1.44269504
. a637  .byte $07                  Polynomgrad
. a638  .byte $71 $34 $58 $3e $56  2.14987637 e-05
. a63d  .byte $74 $16 $7e $b3 $1b  1.43523140 e-04
. a642  .byte $77 $2f $ee $e3 $85  1.34226348 e-03
. a647  .byte $7a $1d $84 $1c $2a  9.61401701 e-03
. a64c  .byte $7c $63 $59 $58 $0a  0.05550051269
. a651  .byte $7e $75 $fd $e7 $c6  0.240226385
. a656  .byte $80 $31 $72 $18 $10  0.693147186 (ln 2)
. a65b  .byte $81 $00 $00 $00 $00  1

. a660  :
. a660  *** E X P ***
. a660  :
. a660  a9 32    lda #$32     Setzt Zeiger auf
. a662  a0 a6    ldy #$a6     ... Konstanten-Tabelle
. a664  20 5c a0 jsr $a05c    FAC := FAC * Konstante
. a667  a5 70    lda $70      Holt niederwertige Stelle von FAC
. a669  69 50    adc #$50     Addiert $50
. a66b  90 03    bcc $a670    Wenn Ergebnis kleiner gleich $FF, dann $a670
. a66d  20 a8 a2 jsr $a2a8    FAC := FAC + 1
. a670  85 56    sta $56      Schreibt Rundungsstelle + $50
. a672  20 94 a2 jsr $a294    ARG := FAC
. a675  a5 61    lda $61      Holt Exponent von FAC
. a677  c9 88    cmp #$88     Exponent kleiner $88 ?
. a679  90 03    bcc $a67e    Ja, dann $a67e
. a67b  20 54 a1 jsr $a154    Pos. Vorzeichen: Fehler; Neg. Vorzeichen: FAC := 0
. a67e  20 58 a3 jsr $a358    INT (schneidet Nachkommastellen ab)
. a681  a5 07    lda $07      Holt ganzzahligen Anteil
. a683  18       clc
. a684  69 81    adc #$81     FAC gleich $7f ?
. a686  f0 f3    beq $a67b    Ja, dann $a67b
. a688  38       sec
. a689  e9 01    sbc #$01
. a68b  48       pha          Vertauschung
. a68c  a2 05    ldx #$05     ... von
. a68e  b5 69    lda $69,x    ... ARG
```

```
. a690  b4 61     ldy $61,x     ... und
. a692  95 61     sta $61,x     ... FAC...
. a694  94 69     sty $69,x
. a696  ca        dex
. a697  10 f5     bpl $a68e
. a699  a5 56     lda $56       Niederwertige Stellen
. a69b  85 70     sta $70       ... von ARG und FAC vertauschen
. a69d  20 87 9e  jsr $9e87     FAC := ARG - FAC
. a6a0  20 27 a6  jsr $a627     FAC := - FAC
. a6a3  a9 37     lda #$37      setzt Zeiger auf
. a6a5  a0 a6     ldy #$a6      ... Polynomgrad
. a6a7  20 c9 a6  jsr $a6c9     Polynom auswerten
. a6aa  a9 00     lda #$00      Setzt Vorzeichen von
. a6ac  85 6f     sta $6f       ... FAC * ARG auf Null
. a6ae  68        pla
. a6af  20 39 a1  jsr $a139     Exponenten addieren
. a6b2  60        rts           ENDE

. a6b3  :
. a6b3  *** POLYNOMAUSWERTUNG ***
. a6b3  :
. a6b3  85 71     sta $71       Speichert Adresse des
. a6b5  84 72     sty $72       ... Polynomgrades
. a6b7  20 4f a2  jsr $a24f     FAC runden und nach Adresse $57 - $5b kopieren
. a6ba  a9 57     lda #$57
. a6bc  20 78 a0  jsr $a078     FAC := FAC * Adresse $57 - $5b
. a6bf  20 cd a6  jsr $a6cd     Horner- Algorithmus
. a6c2  a9 57     lda #$57      Setzt Zeiger auf
. a6c4  a0 00     ldy #$00      ... Adresse $0057
. a6c6  4c 78 a0  jmp $a078     FAC := FAC * Adresse $57 - $5b

. a6c9  :
. a6c9  *** HORNER - ALGORITHMUS ***
. a6c9  :
. a6c9  85 71     sta $71       Speichert Adresse des
. a6cb  84 72     sty $72       ... Polynomgrades
. a6cd  20 4c a2  jsr $a24c     FAC nach Adresse $5c - $60
. a6d0  b1 71     lda ($71),y   Polynomgrad
. a6d2  85 67     sta $67       ... als Zähler
. a6d4  a4 71     ldy $71       ... einrichten
. a6d6  c8        iny           ... und
. a6d7  98        tya           ... Zeiger
. a6d8  d0 02     bne $a6dc     ... auf
. a6da  e6 72     inc $72       ... Anfang der
. a6dc  85 71     sta $71       ... Koeffiziententabelle
. a6de  a4 72     ldy $72       ... setzen
. a6e0  20 5c a0  jsr $a05c     FAC := FAC * Koeffizient
. a6e3  a5 71     lda $71       Zeiger
. a6e5  a4 72     ldy $72       ... auf
. a6e7  18        clc           ... nächsten
. a6e8  69 05     adc #$05      ... Koeffizienten
. a6ea  90 01     bcc $a6ed     ... setzen...
```

```
. a6ec  c8       iny
. a6ed  85 71    sta $71
. a6ef  84 72    sty $72
. a6f1  20 66 a0 jsr $a066    FAC := FAC + Koeffizient
. a6f4  a9 5c    lda #$5c     Setzt Zeiger auf
. a6f6  a0 00    ldy #$00     ... Adresse $005c
. a6f8  c6 67    dec $67      Zähler um 1 erniedrigen
. a6fa  d0 e4    bne $a6e0    Ungleich Null ?, dann $a6e0
. a6fc  60       rts          ENDE

. a6fd  :
. a6fd  *** ZUFALLSZAHLEN FÜR RND ***
. a6fd  :
. a6fd  .byte $98 $35 $44 $7a $00   1. Zufallszahl
. a702  .byte $68 $28 $b1 $46 $00   2. Zufallszahl

. a707  :
. a707  *** R N D ***
. a707  :
. a707  20 b0 a2 jsr $a2b0    Vorzeichen von FAC ermitteln
. a70a  30 2e    bmi $a73a    Wenn negativ, dann $a73a
. a70c  d0 17    bne $a725    Wenn positiv, dann $a725
. a70e  ad 00 ff lda $ff00    Liest
. a711  85 62    sta $62      ... Bytes
. a713  ad 01 ff lda $ff01    ... aus
. a716  85 64    sta $64      ... Timer
. a718  ad 02 ff lda $ff02    ... und
. a71b  85 63    sta $63      ... schreibt
. a71d  ad 03 ff lda $ff03    ... diese vermischt
. a720  85 65    sta $65      ... nach FAC
. a722  4c 4a a7 jmp $a74a    Springt nach $a74a

. a725  a9 03    lda #$03     Holt Adresse der
. a727  a0 05    ldy #$05     ... letzten Zufallszahl
. a729  20 21 a2 jsr $a221    Schreibt Zufallszahl nach FAC
. a72c  a9 fd    lda #$fd     Setzt Zeiger auf
. a72e  a0 a6    ldy #$a6     ... 1. Zufallszahl
. a730  20 5c a0 jsr $a05c    FAC := FAC * Konstante
. a733  a9 02    lda #$02     Setzt Zeiger auf
. a735  a0 a7    ldy #$a7     ... 1. Zufallszahl
. a737  20 66 a0 jsr $a066    FAC := FAC + Konstante
. a73a  a6 65    ldx $65      Vertauscht
. a73c  a5 62    lda $62      ... Stellen
. a73e  85 65    sta $65      ... von
. a740  86 62    stx $62      ... FAC...
. a742  a6 63    ldx $63
. a744  a5 64    lda $64
. a746  85 63    sta $63
. a748  86 64    stx $64
. a74a  a9 00    lda #$00     Setzt Vorzeichenbyte
. a74c  85 66    sta $66      ... gleich Null
. a74e  a5 61    lda $61      Holt Exponent von FAC
```

```
. a75Ø  85 7Ø     sta $7Ø       Schreibt niederwertige Stelle
. a752  a9 8Ø     lda #$8Ø      Setzt Exponent
. a754  85 61     sta $61       ... auf $8Ø
. a756  2Ø Øb 9f  jsr $9fØb     FAC normalisieren
. a759  a2 Ø3     ldx #$Ø3
. a75b  aØ Ø5     ldy #$Ø5
. a75d  4c 59 a2  jmp $a259     Rundet FAC und FAC abspeichern

. a76Ø  :
. a76Ø  *** HILFSROUTINEN FÜR BASIC-STACK ***
. a76Ø  :
. a76Ø  a5 7c     lda $7c       Umspeichern
. a762  85 3d     sta $3d       ... des
. a764  a5 7d     lda $7d       ... Stack-
. a766  85 3e     sta $3e       ... Zeigers
. a768  6Ø        rts           ENDE

. a769  a5 3d     lda $3d       Umspeichern
. a76b  85 7c     sta $7c       ... des
. a76d  a5 3e     lda $3e       ... Stack-
. a76f  85 7d     sta $7d       ... Zeigers
. a771  6Ø        rts           ENDE

. a772  98        tya           Erhöht
. a773  18        clc           ... Stack-
. a774  65 7c     adc $7c       ... Zeiger
. a776  85 7c     sta $7c       ... um
. a778  9Ø Ø2     bcc $a77c     ... Y-Register...
. a77a  e6 7d     inc $7d
. a77c  6Ø        rts           ENDE

. a77d  :
. a77d  *** KERNEL - AUFRUFE ***
. a77d  :
. a77d  aa        tax           Holt Fehlernummer nach X-Register
. a77e  dØ Ø2     bne $a782     Fehlernummer vorhanden ?, dann $a78e
. a78Ø  a2 1e     ldx #$1e      Sonst Fehlernummer 3Ø ('BREAK ERROR') holen
. a782  4c 83 86  jmp $8683     Fehlerausgabe

. a785  2Ø cØ ff  jsr $ffcØ     OPEN
. a788  bØ f3     bcs $a77d     Fehler ?, dann $a77d
. a78a  6Ø        rts           ENDE

. a78b  2Ø d2 ff  jsr $ffd2     (BSOUT)
. a78e  bØ ed     bcs $a77d     Fehler ?, dann $a77d
. a79Ø  6Ø        rts           ENDE

. a791  2Ø cf ff  jsr $ffcf     (BASIN)
. a794  bØ e7     bcs $a77d     Fehler ?, dann $a77d
. a796  6Ø        rts           ENDE

. a797  48        pha
```

```
. a798  20 c9 ff jsr $ffc9    (CKOUT)
. a79b  20 f8 a8 jsr $a8f8    Wenn Bus, dann DS$ löschen
. a79e  aa       tax
. a79f  68       pla
. a7a0  90 03    bcc $a7a5    Kein Fehler ?, dann $a7a5
. a7a2  8a       txa
. a7a3  b0 d8    bcs $a77d    Fehler ?, dann $a77d
. a7a5  60       rts          ENDE

. a7a6  20 c6 ff jsr $ffc6    (CHKIN) Eingabe-Umlenkung
. a7a9  20 f8 a8 jsr $a8f8    Wenn Bus, dann DS$ löschen
. a7ac  b0 cf    bcs $a77d    Fehler ?, dann $a77d
. a7ae  60       rts          ENDE

. a7af  20 e4 ff jsr $ffe4    (GETIN) Ein-/Ausgabe-Status lesen
. a7b2  b0 c9    bcs $a77d    Fehler ?, dann $a77d
. a7b4  60       rts          ENDE

. a7b5  :
. a7b5  *** S Y S ***
. a7b5  :
. a7b5  20 e1 9d jsr $9de1    Holt Adresse
. a7b8  a9 a7    lda #$a7     Legt
. a7ba  48       pha          ... Rücksprungadresse -1 ($a7ce)
. a7bb  a9 ce    lda #$ce     ... auf
. a7bd  48       pha          ... Stack
. a7be  ad f5 07 lda $07f5    Rettet CPU-Status
. a7c1  48       pha          ... auf Stack
. a7c2  ad f2 07 lda $07f2    Setzt Akku
. a7c5  ae f3 07 ldx $07f3    Setzt X-Reg
. a7c8  ac f4 07 ldy $07f4    Setzt Y-Reg
. a7cb  28       plp
. a7cc  6c 14 00 jmp ($0014)  Aufruf des Maschinenspracheprogramms

. a7cf  08       php
. a7d0  8d f2 07 sta $07f2    Akku retten
. a7d3  8e f3 07 stx $07f3    X-Reg. retten
. a7d6  8c f4 07 sty $07f4    Y-Reg. retten
. a7d9  68       pla
. a7da  8d f5 07 sta $07f5    CPU-Status retten
. a7dd  60       rts          ENDE

. a7de  :
. a7de  *** S A V E ***
. a7de  :
. a7de  20 6b a8 jsr $a86b    Übernimmt Parameter
. a7e1  a6 2d    ldx $2d      Holt
. a7e3  a4 2e    ldy $2e      ... Programmende
. a7e5  a9 2b    lda #$2b     Zeiger auf Adresse von Programmanfang
. a7e7  20 d8 ff jsr $ffd8    SAVE
. a7ea  20 f8 a8 jsr $a8f8    Wenn Bus, dann DS$ löschen
. a7ed  b0 8e    bcs $a77d    Fehler ?, dann $a77d
```

```
. a7ef  60       rts          ENDE

. a7f0  :
. a7f0  *** V E R I F Y ***
. a7f0  :
. a7f0  a9 01    lda #$01     Flag für VERIFY

. a7f2  2c       .byte $2c

. a7f3  :
. a7f3  *** L O A D ***
. a7f3  :
. a7f3  a9 00    lda #$00     Flag für LOAD
. a7f5  85 0a    sta $0a      Setzt Verify/Load-Flag
. a7f7  20 6b a8 jsr $a86b    Übernimmt Parameter
. a7fa  a5 0a    lda $0a      Holt Load/Verify-Flag
. a7fc  a6 2b    ldx $2b      Holt
. a7fe  a4 2c    ldy $2c      ... Programmanfang
. a800  20 d5 ff jsr $ffd5    LOAD
. a803  08       php
. a804  20 f8 a8 jsr $a8f8    Wenn Bus, dann DS$ löschen
. a807  28       plp
. a808  b0 5e    bcs $a868    Fehler ?, dann $a868
. a80a  a5 0a    lda $0a      LOAD-Befehl ?
. a80c  f0 16    beq $a824    Ja, dann $a824
. a80e  a2 1c    ldx #$1c     Holt Fehlermeldung 28 ('FERIFY')
. a810  20 b7 ff jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. a813  29 10    and #$10     Verify-Error-Bit gesetzt ?
. a815  d0 16    bne $a82d    Ja, dann $a82d
. a817  24 81    bit $81      Direktmodus ?
. a819  30 08    bmi $a823    Nein, dann $a823
. a81b  20 4f ff jsr $ff4f    '(CR)OK(CR)' ausgeben

. a81e  0d       .byte '(CR)'
. a81f  4f 4b    .byte 'OK'
. a821  0d       .byte '(CR)'
. a822  00       .byte $00
. a823  60       rts          ENDE

. a824  20 b7 ff jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. a827  29 bf    and #$bf     EOI löschen
. a829  f0 05    beq $a830    Wenn kein weiteres Bit gesetzt, dann $a830
. a82b  a2 1d    ldx #$1d     Holt Fehlermeldung 29 ('LOAD')
. a82d  4c 83 86 jmp $8683    Fehlerausgabe

. a830  24 81    bit $81      Direktmodus ?
. a832  30 10    bmi $a844    Nein, dann $a844
. a834  86 2d    stx $2d      Setzt LOAD-Endadresse
. a836  84 2e    sty $2e      ... gleich Variablenanfang
. a838  20 6f 86 jsr $866f    'READY.' ausgeben
. a83b  20 18 88 jsr $8818    Linkadresse erzeugen
. a83e  20 93 8a jsr $8a93    Textpointer zurücksetzen und CLR
```

```
. a841  4c 0f 87 jmp $870f    READY.

. a844  20 f1 8a jsr $8af1    Textpointer zurücksetzen
. a847  20 18 88 jsr $8818    Linkadresse erzeugen
. a84a  4c d5 8a jmp $8ad5    RESTORE, Stack initialisieren und PRG starten

. a84d  :
. a84d  *** O P E N ***
. a84d  :
. a84d  20 b0 a8 jsr $a8b0    Übernimmt Parameter
. a850  18       clc
. a851  20 85 a7 jsr $a785    OPEN
. a854  20 f8 a8 jsr $a8f8    Wenn Bus, dann DS$ löschen
. a857  b0 0f    bcs $a868    Fehler ?, dann $a868
. a859  60       rts          ENDE

. a85a  :
. a85a  *** C L O S E ***
. a85a  :
. a85a  20 b0 a8 jsr $a8b0    Übernimmt Parameter
. a85d  a5 49    lda $49
. a85f  18       clc
. a860  20 c3 ff jsr $ffc3    CLOSE
. a863  20 f8 a8 jsr $a8f8    Wenn Bus, dann DS$ löschen
. a866  90 bb    bcc $a823    Kein Fehler ?, dann $a823
. a868  4c 7d a7 jmp $a77d    Fehlerausgabe

. a86b  :
. a86b  *** PARAMETER - ÜBERNAHME (LOAD/SAVE) ***
. a86b  :
. a86b  a9 00    lda #$00
. a86d  20 bd ff jsr $ffbd    (SETNAM) Filename löschen
. a870  a2 01    ldx #$01     Geräteadresse (FA) gleich 1 (für Kassette)
. a872  a0 00    ldy #$00     Sekundäradresse (SA) gleich 0 (für 'lesen')
. a874  20 ba ff jsr $ffba    (SETLFS) Setzt log.Adr., Geräte- & Sekundäradr.
. a877  20 9d a8 jsr $a89d    CHRGOT, Abbruch bei Trennzeichen
. a87a  20 ee a8 jsr $a8ee    Übernimmt Filenamen
. a87d  20 9d a8 jsr $a89d    CHRGOT, Abbruch bei Trennzeichen
. a880  20 97 a8 jsr $a897    (CHKCOM), (GETBYT)
. a883  a0 00    ldy #$00     Sekundäradresse (SA) gleich 0 (für 'lesen')
. a885  86 49    stx $49      Schreibt Geräteadresse nach $49
. a887  20 ba ff jsr $ffba    (SETLFS) Setzt log.Adr., Geräte- und Sekundäradr.
. a88a  20 9d a8 jsr $a89d    CHRGOT, Abbruch bei Trennzeichen
. a88d  20 97 a8 jsr $a897    (CHKCOM), (GETBYT)
. a890  8a       txa          Bringt Sekundäradresse
. a891  a8       tay          ... nach Y-Reg.
. a892  a6 49    ldx $49      Holt Geräteadresse
. a894  4c ba ff jmp $ffba    (SETLFS) Setzt log.Adr., Geräte- und Sekundäradr.

. a897  20 a5 a8 jsr $a8a5    (CHKCOM), CHRGOT, Fehler, wenn bei Ende
. a89a  4c 84 9d jmp $9d84    (GETBYT) Byte übernehmen
```

```
. a89d  20 79 04 jsr $0479    CHRGOT-Routine
. a8a0  d0 02    bne $a8a4    Kein Trennzeichen ?, dann $a8a4
. a8a2  68       pla
. a8a3  68       pla
. a8a4  60       rts          ENDE

. a8a5  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. a8a8  20 79 04 jsr $0479    CHRGOT-Routine
. a8ab  d0 f7    bne $a8a4    Kein Trennzeichen ?, dann $a8a4
. a8ad  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. a8b0  :
. a8b0  *** PARAMETER - ÜBERNAHME (OPEN/CLOSE) ***
. a8b0  :
. a8b0  a9 00    lda #$00
. a8b2  20 bd ff jsr $ffbd    (SETNAM) Filename löschen
. a8b5  20 a8 a8 jsr $a8a8    CHRGOT, Fehler bei Ende
. a8b8  20 84 9d jsr $9d84    (GETBYT) Byte übernehmen
. a8bb  86 49    stx $49      Speichert logische Adresse
. a8bd  8a       txa          Schreibt logische Adresse nach Akku
. a8be  a2 01    ldx #$01     Geräteadresse (FA) gleich 1 (für Kassette)
. a8c0  a0 00    ldy #$00     Sekundäradresse (SA) gleich 0 (für 'lesen')
. a8c2  20 ba ff jsr $ffba    (SETLFS) Setzt log.Adr., Geräte- und Sekundäradr.
. a8c5  20 9d a8 jsr $a89d    CHRGOT, Abbruch bei Trennzeichen
. a8c8  20 97 a8 jsr $a897    (CHKCOM), (GETBYT)
. a8cb  86 4a    stx $4a      Speichert Geräteadresse
. a8cd  a0 00    ldy #$00     Sekundäradresse (SA) gleich 0 (für 'lesen')
. a8cf  a5 49    lda $49      Holt logische Adresse
. a8d1  e0 03    cpx #$03     Geräteadresse kleiner 3 ?
. a8d3  90 01    bcc $a8d6    Ja, dann $a8d6
. a8d5  88       dey          Sekundäradresse gleich 255 ($ff)
. a8d6  20 ba ff jsr $ffba    (SETLFS) Setzt log.Adr., Geräte- und Sekundäradr.
. a8d9  20 9d a8 jsr $a89d    CHRGOT, Abbruch bei Trennzeichen
. a8dc  20 97 a8 jsr $a897    (CHKCOM), (GETBYT)
. a8df  8a       txa
. a8e0  a8       tay
. a8e1  a6 4a    ldx $4a
. a8e3  a5 49    lda $49
. a8e5  20 ba ff jsr $ffba    (SETLFS) Setzt log.Adr., Geräte- und Sekundäradr.
. a8e8  20 9d a8 jsr $a89d    CHRGOT, Abbruch bei Trennzeichen
. a8eb  20 a5 a8 jsr $a8a5    (CHKCOM), (GETBYT)
. a8ee  20 48 9c jsr $9c48    Übernimmt Filenamen
. a8f1  a6 22    ldx $22      Holt
. a8f3  a4 23    ldy $23      ... Stringadresse
. a8f5  4c bd ff jmp $ffbd    (SETNAM) Filename löschen

. a8f8  :
. a8f8  *** DS$ LÖSCHEN ***
. a8f8  :
. a8f8  08       php
. a8f9  48       pha
. a8fa  a5 ae    lda $ae      Holt aktuelle Geräteadresse (FA)
```

```
. a8fc  c9 Ø4     cmp #$Ø4      Bus ?
. a8fe  9Ø Ø3     bcc $a9Ø3     Nein, dann $a9Ø3
. a9ØØ  2Ø 57 cd  jsr $cd57     DS$ löschen
. a9Ø3  68        pla
. a9Ø4  28        plp
. a9Ø5  6Ø        rts           ENDE

. a9Ø6  :
. a9Ø6  *** STRING - PLATZRESERVIERUNG ***
. a9Ø6  :
. a9Ø6  46 Øf     lsr $Øf       Löscht Flag für GARBAGE COLLECT
. a9Ø8  aa        tax           Stringlänge gleich Null ?
. a9Ø9  fØ 38     beq $a943     Ja, dann $a943
. a9Øb  48        pha           Stringlänge auf Stack legen
. a9Øc  a5 33     lda $33       Holt
. a9Øe  38        sec           ... Stringbereich-Anfang
. a9Øf  e9 Ø2     sbc #$Ø2      ... und
. a911  a4 34     ldy $34       ... subtrahiert
. a913  bØ Ø1     bcs $a916     ... 2 Byte (R-Zeiger)...
. a915  88        dey
. a916  85 22     sta $22       Schreibt Adresse
. a918  84 23     sty $23       ... des R-Zeigers
. a91a  8a        txa           Invertiert
. a91b  49 ff     eor #$ff      ... Stringlänge
. a91d  38        sec           ... und
. a91e  65 22     adc $22       ... addiert
. a92Ø  bØ Ø1     bcs $a923     ... Adresse
. a922  88        dey           ... des R-Zeigers
. a923  c4 32     cpy $32       Ist Ergebnis
. a925  9Ø 1d     bcc $a944     ... kleiner
. a927  dØ Ø4     bne $a92d     ... Variablen-
. a929  c5 31     cmp $31       ... Ende ?
. a92b  9Ø 17     bcc $a944     Ja, dann $a944
. a92d  85 35     sta $35       Schreibt
. a92f  84 36     sty $36       ... String-Transportzeiger
. a931  aØ Ø1     ldy #$Ø1      Setzt
. a933  a9 ff     lda #$ff      ... R-Zeiger
. a935  91 22     sta ($22),y   ... für ungültigen
. a937  88        dey           ... String
. a938  68        pla           Holt Stringlänge vom Stack
. a939  91 22     sta ($22),y   Schreibt Stringlänge
. a93b  a6 35     ldx $35       Setzt
. a93d  a4 36     ldy $36       ... String-Transportzeiger
. a93f  86 33     stx $33       ... gleich neuen
. a941  84 34     sty $34       ... Stringbereichs- Anfang
. a943  6Ø        rts           ENDE

. a944  a5 Øf     lda $Øf       GARBAGE COLLECT schon durchgeführt ?
. a946  3Ø Ø9     bmi $a951     Ja, dann 'OUT OF MEMORY ERROR' ausgeben
. a948  2Ø 54 a9  jsr $a954     GARBAGE COLLECT
. a94b  38        sec           Setzt Flag
. a94c  66 Øf     ror $Øf       ... für GARBAGE COLLECT
```

```
. a94e  68       pla          Holt Stringlänge
. a94f  dØ b7    bne $a9Ø8    Springt immer nach $a9Ø8

. a951  4c 81 86 jmp $8681    'OUT OF MEMORY ERROR' ausgeben

. a954  :
. a954  *** GARBAGE COLLECT ***
. a954  :
. a954  a6 16    ldx $16      Holt Zeiger auf String-Stack
. a956  eØ 19    cpx #$19     Stack leer ?
. a958  fØ 1Ø    beq $a96a    Ja, dann $a96a
. a95a  2Ø 57 aa jsr $aa57    Kopiert Adresse des R-Zeigers nach $5c,$5d
. a95d  fØ f7    beq $a956    Stringlänge gleich Null ?, dann $a956
. a95f  8a       txa          Holt
. a96Ø  aØ ØØ    ldy #$ØØ     ... Deskriptor-Adresse
. a962  91 5c    sta ($5c),y  ... und schreibt
. a964  98       tya          ... diese in
. a965  c8       iny          ... R-Zeiger
. a966  91 5c    sta ($5c),y  ... (Dadurch wird String gültig)
. a968  dØ ec    bne $a956    Springt immer nach $a956

. a96a  aØ ØØ    ldy #$ØØ     Flag
. a96c  84 58    sty $58      ... initialisieren
. a96e  a6 37    ldx $37      Holt
. a97Ø  a4 38    ldy $38      ... String-Obergrenze
. a972  86 5f    stx $5f      Schreibt Zielbereichszeiger (Low)
. a974  86 4e    stx $4e      Schreibt Quellbereichszeiger (Low)
. a976  86 35    stx $35      Schreibt Transportzeiger (Low)
. a978  84 6Ø    sty $6Ø      Schreibt Zielbereichszeiger (High)
. a97a  84 4f    sty $4f      Schreibt Quellbereichszeiger (High)
. a97c  84 36    sty $36      Schreibt Transportzeiger (High)
. a97e  8a       txa
. a97f  2Ø ea a9 jsr $a9ea    Zeiger auf nächsten String setzen
. a982  dØ Øc    bne $a99Ø    String gültig ?, dann $a99Ø
. a984  88       dey
. a985  2Ø 65 81 jsr $8165    Holt Stringlänge (LDA ($4e),Y)
. a988  2Ø 39 aa jsr $aa39    Stringlänge vom Q-Zeiger subtrahieren
. a98b  38       sec
. a98c  66 58    ror $58      Setzt Flag
. a98e  dØ ef    bne $a97f    Springt immer nach $a97f

. a99Ø  24 58    bit $58      Flag gesetzt ?
. a992  1Ø 42    bpl $a9d6    Nein, dann $a9d6
. a994  a2 ØØ    ldx #$ØØ     Flag
. a996  86 58    stx $58      ... initialisieren
. a998  a9 Ø2    lda #$Ø2     ??? Sinnlos ???
. a99a  aØ Ø1    ldy #$Ø1     Verschiebt R-Zeiger
. a99c  2Ø 65 81 jsr $8165    LDA ($4e),Y
. a99f  91 5f    sta ($5f),y
. a9a1  88       dey
. a9a2  2Ø 65 81 jsr $8165    LDA ($4e),Y
. a9a5  91 5f    sta ($5f),y
```

```
. a9a7  20 b0 04 jsr $04b0    Holt Stringlänge (LDA ($22),Y)
. a9aa  aa       tax          Stringlänge nach X-Reg.
. a9ab  20 48 aa jsr $aa48    Stringlänge vom R-Zeiger subtrahieren
. a9ae  85 35    sta $35      Schreibt Ergebnis
. a9b0  84 36    sty $36      ... in Transportzeiger
. a9b2  8a       txa          Stringlänge nach Akku
. a9b3  20 39 aa jsr $aa39    Stringlänge vom Q-Zeiger subtrahieren
. a9b6  8a       txa          Setzt Stringlänge
. a9b7  a8       tay          ... gleich Offset
. a9b8  88       dey          String verschieben...
. a9b9  20 65 81 jsr $8165    LDA ($4e),Y
. a9bc  91 5f    sta ($5f),y
. a9be  ca       dex
. a9bf  d0 f7    bne $a9b8
. a9c1  a0 02    ldy #$02
. a9c3  b9 5e 00 lda $005e,y  Kopiert Z-Zeiger
. a9c6  91 22    sta ($22),y  ... in Deskriptor
. a9c8  88       dey
. a9c9  d0 f8    bne $a9c3
. a9cb  a5 4e    lda $4e      Holt
. a9cd  a4 4f    ldy $4f      ... Q-Zeiger
. a9cf  20 ea a9 jsr $a9ea    Zeiger auf nächsten String setzen
. a9d2  f0 b0    beq $a984        String ungültig ?, dann $a984
. a9d4  d0 c4    bne $a99a    Springt immer nach $a99a

. a9d6  a0 00    ldy #$00
. a9d8  20 b0 04 jsr $04b0    Holt Stringlänge (LDA ($22),Y)
. a9db  aa       tax          Schreibt Stringlänge nach X-Reg.
. a9dc  20 48 aa jsr $aa48    Stringlänge vom Z-Zeiger subtrahieren
. a9df  85 35    sta $35      Schreibt Ergebnis
. a9e1  84 36    sty $36      ... in Transportzeiger
. a9e3  8a       txa          Schreibt Stringlänge nach Akku
. a9e4  20 39 aa jsr $aa39    Stringlänge vom Q-Zeiger subtrahieren
. a9e7  4c 7f a9 jmp $a97f    Springt nach $a97f
```

ZEIGER AUF NÄCHSTEN STRING SETZEN

```
. a9ea  c4 34    cpy $34      Wenn
. a9ec  90 2a    bcc $aa18    ... Q-Zeiger
. a9ee  d0 06    bne $a9f6    ... kleiner
. a9f0  c5 33    cmp $33      ... als
. a9f2  f0 24    beq $aa18    ... String-Untergrenze,
. a9f4  90 22    bcc $aa18    ... dann $aa18
. a9f6  24 58    bit $58      Flag gesetzt ?
. a9f8  30 05    bmi $a9ff    Ja, dann $a9ff
. a9fa  a9 02    lda #$02     2 Byte vom
. a9fc  20 48 aa jsr $aa48    ... Z-Zeiger subtrahieren
. a9ff  a9 02    lda #$02     2 Byte vom
. aa01  20 39 aa jsr $aa39    ... Q-Zeiger subtrahieren
. aa04  a0 01    ldy #$01
. aa06  20 65 81 jsr $8165    Holt R-Zeiger (LDA ($4e),Y)
. aa09  c9 ff    cmp #$ff     String ungültig ?
```

```
. aaØb  dØ Ø1     bne $aaØe     Nein, dann $aaØe
. aaØd  6Ø        rts           ENDE

      R-ZEIGER NACH DESKRIPTOR-ZEIGER KOPIEREN

. aaØe  2Ø 65 81 jsr $8165     Holt R-Zeiger (LDA ($4e),Y)
. aa11  99 22 ØØ sta $ØØ22,y  Schreibt Deskriptor-Zeiger
. aa14  88        dey
. aa15  1Ø f7     bpl $aaØe
. aa17  6Ø        rts           ENDE

      STRINGS IM STACK UNGÜLTIG MACHEN

. aa18  a6 16     ldx $16       Holt Zeiger im String-Stack
. aa1a  eØ 19     cpx #$19      Stack leer ?
. aa1c  fØ 1Ø     beq $aa2e     Ja, dann fertig
. aa1e  2Ø 57 aa jsr $aa57     Kopiert R-Zeiger nach Adresse $5c-$5d
. aa21  fØ f7     beq $aa1a     Stringlänge gleich Null ?, dann $aa1a
. aa23  aØ ØØ     ldy #$ØØ      String ungültig machen
. aa25  91 5c     sta ($5c),y   Schreibt Stringlänge nach R-Zeiger
. aa27  c8        iny
. aa28  a9 ff     lda #$ff      Schreibt $ff
. aa2a  91 5c     sta ($5c),y   ... in R-Zeiger
. aa2c  dØ ec     bne $aa1a     Springt immer nach $aa1a

      ABSCHLUSS

. aa2e  68        pla           Rücksprungadresse
. aa2f  68        pla           ... vom Stack holen
. aaØØ  a5 35     lda $35       Holt
. aa32  a4 36     ldy $36       ... Transportzeiger
. aa34  85 33     sta $33       Transportzeiger gleich
. aa36  84 34     sty $34       ... neue String-Untergrenze
. aa38  6Ø        rts           ENDE

      AKKU VOM Q-ZEIGER SUBTRAHIEREN

. aa39  49 ff     eor #$ff      Akku invertieren
. aa3b  38        sec           Carry = 1
. aa3c  65 4e     adc $4e       Addiert Akku
. aa3e  a4 4f     ldy $4f       ... und Carry
. aa4Ø  bØ Ø1     bcs $aa43     ... zum
. aa42  88        dey           ... Q-Zeiger
. aa43  85 4e     sta $4e       Schreibt neuen
. aa45  84 4f     sty $4f       ... Q-Zeiger
. aa47  6Ø        rts           ENDE

      AKKU VOM Z-ZEIGER SUBTRAHIEREN

. aa48  49 ff     eor #$ff      Akku invertieren
. aa4a  38        sec           Carry = 1
. aa4b  65 5f     adc $5f       Addiert Akku
```

```
. aa4d  a4 60     ldy $60       ... und Carry
. aa4f  b0 01     bcs $aa52     ... zum
. aa51  88        dey           ... Z-Zeiger
. aa52  85 5f     sta $5f       Schreibt neuen
. aa54  84 60     sty $60       ... Z-Zeiger
. aa56  60        rts           ENDE

      $5c, $5d := R-ZEIGER

. aa57  ca        dex
. aa58  b5 00     lda $00,x     Holt Stringadresse (High) aus Stack
. aa5a  85 5d     sta $5d
. aa5c  ca        dex
. aa5d  b5 00     lda $00,x     Holt Stringadresse (Low) aus Stack
. aa5f  85 5c     sta $5c
. aa61  ca        dex
. aa62  b5 00     lda $00,x     Holt Stringlänge
. aa64  48        pha
. aa65  18        clc
. aa66  65 5c     adc $5c       Addiert Stringadresse...
. aa68  85 5c     sta $5c       ... gleich neue
. aa6a  90 02     bcc $aa6e     ... R-Zeiger-Adresse
. aa6c  e6 5d     inc $5d
. aa6e  68        pla
. aa6f  60        rts           ENDE

. aa70  :
. aa70  *** C O S ***
. aa70  :
. aa70  a9 ec     lda #$ec      Setzt Zeiger auf
. aa72  a0 aa     ldy #$aa      ... Konstante pi/2 (1.5700079633)
. aa74  20 66 a0  jsr $a066     FAC := FAC + pi/2
. aa77  :
. aa77  *** S I N ***
. aa77  :
. aa77  20 91 a2  jsr $a291     ARG := FAC
. aa7a  a9 f1     lda #$f1      Setzt Zeiger auf
. aa7c  a0 aa     ldy #$aa      ... Konstante 2*pi (6.28318531)
. aa7e  a6 6e     ldx $6e       Holt Vorzeichenbyte von ARG
. aa80  20 8c a1  jsr $a18c     FAC := ARG / (2*pi)
. aa83  20 91 a2  jsr $a291     ARG := FAC
. aa86  20 58 a3  jsr $a358     INT
. aa89  a9 00     lda #$00      Setzt Vorzeichen
. aa8b  85 6f     sta $6f       ... von ARG * FAC auf Null
. aa8d  20 87 9e  jsr $9e87     FAC := ARG - FAC
. aa90  a9 f6     lda #$f6      Setzt Zeiger auf
. aa92  a0 aa     ldy #$aa      ... Konstante 0.25
. aa94  20 6c a0  jsr $a06c     FAC := 0.25 - FAC
. aa97  a5 66     lda $66       Holt Vorzeichenbyte von FAC
. aa99  48        pha
. aa9a  10 0d     bpl $aaa9     Vorzeichen positiv ?, dann $aaa9
. aa9c  20 62 a0  jsr $a062     FAC := FAC + 0.5
```

```
. aa9f  a5 66     lda $66      Holt Vorzeichenbyte von FAC
. aaa1  30 09     bmi $aaac    Vorzeichen negativ ?, dann $aaac
. aaa3  a5 12     lda $12      Invertiert
. aaa5  49 ff     eor #$ff     ... Vorzeichen-
. aaa7  85 12     sta $12      ... Flag
. aaa9  20 27 a6  jsr $a627    FAC := -FAC
. aaac  a9 f6     lda #$f6     Setzt Zeiger auf
. aaae  a0 aa     ldy #$aa     ... Konstante 0.25
. aab0  20 66 a0  jsr $a066    FAC := FAX + 0.25
. aab3  68        pla
. aab4  10 03     bpl $aab9
. aab6  20 27 a6  jsr $a627    FAC := -FAC
. aab9  a9 fb     lda #$fb     Setzt Zeiger auf
. aabb  a0 aa     ldy #$aa     ... Polynomgrad
. aabd  4c b3 a6  jmp $a6b3    Polynomauswertung

. aac0  :
. aac0  *** T A N ***
. aac0  :
. aac0  20 4f a2  jsr $a24f    FAC nach $57-$5b kopieren
. aac3  a9 00     lda #$00     Setzt Vorzeichen-Flag
. aac5  85 12     sta $12      ... auf '+'
. aac7  20 77 aa  jsr $aa77    SIN
. aaca  a2 4e     ldx #$4e     Setzt Zeiger auf
. aacc  a0 00     ldy #$00     ... Adresse $004e
. aace  20 5d a7  jsr $a75d    FAC runden und nach $004e kopieren
. aad1  a9 57     lda #$57     Setzt Zeiger auf
. aad3  a0 00     ldy #$00     ... Adresse $0057
. aad5  20 21 a2  jsr $a221    Kopiert $57-$5b nach FAC
. aad8  a9 00     lda #$00     Setzt Vorzeichen
. aada  85 66     sta $66      ... von FAC auf Null
. aadc  a5 12     lda $12      Holt Vorzeichen-Flag
. aade  20 e8 aa  jsr $aae8    FAC := COS(FAC)
. aae1  a9 4e     lda #$4e     Setzt Zeiger auf
. aae3  a0 00     ldy #$00     ... Adresse $004e
. aae5  4c 94 a1  jmp $a194    FAC := ($4e-$52) / FAC

. aae8  48        pha
. aae9  4c a9 aa  jmp $aaa9    COS

. aaec  :
. aaec  *** KONSTANTEN FÜR SIN & COS ***
. aaec  :

. aaec  .byte $81 $49 $0f $da $a2   1.570079633 (pi/2)
. aaf1  .byte $83 $49 $0f $da $a2   6.28318531  (pi*2)
. aaf6  .byte $7f $00 $00 $00 $00   0.25
. aafb  .byte $05                   Polynomgrad
. aafc  .byte $84 $e6 $1a $2d $1b  -14.3813907
. ab01  .byte $86 $28 $07 $fb $f8   42.0077971
. ab06  .byte $87 $99 $68 $89 $01  -76.7041703
. ab0b  .byte $87 $23 $35 $df $e1   81.6052237
```

```
. ab10  .byte $86 $a5 $5d $e7 $28  -41.3417021
. ab15  .byte $83 $49 $0f $da $a2   6.28318531  (pi*2)

. ab1a  :
. ab1a  *** A T N ***
. ab1a  :
. ab1a  a5 66     lda $66       Holt Vorzeichenbyte von FAC
. ab1b  48        pha           Auf Stack retten
. ab1d  10 03     bpl $ab22     Vorzeichen positiv ?, dann $ab22
. ab1f  20 27 a6  jsr $a627     FAC := -FAC
. ab22  a5 61     lda $61       Holt Exponent von FAC
. ab24  48        pha           Auf Stack retten
. ab25  c9 81     cmp #$81      FAC kleiner 1 ?
. ab27  90 07     bcc $ab30     Ja, dann $ab30
. ab29  a9 f0     lda #$f0      Setzt Zeiger auf
. ab2b  a0 9f     ldy #$9f      ... Konstante 1
. ab2d  20 72 a0  jsr $a072     FAC := 1 / FAC
. ab30  a9 4a     lda #$4a      Setzt Zeiger auf
. ab32  a0 ab     ldy #$ab      ... Polynomgrad
. ab34  20 b3 a6  jsr $a6b3     Polynomauswertung
. ab37  68        pla           Holt alten Exponenten von FAC
. ab38  c9 81     cmp #$81      War FAC kleiner 1 ?
. ab3a  90 07     bcc $ab43     Ja, dann $ab43
. ab3c  a9 ec     lda #$ec      Setzt Zeiger auf
. ab3e  a0 aa     ldy #$aa      ... Konstante pi/2 (1.570079633)
. ab40  20 6c a0  jsr $a06c     FAC := pi/2 - FAC
. ab43  68        pla           Holt Vorzeichen des Arguments
. ab44  10 03     bpl $ab49     Vorzeichen positiv ?, dann $ab49
. ab46  4c 27 a6  jmp $a627     FAC := -FAC

. ab49  60        rts           ENDE

. ab4a  :
. ab4a  *** KONSTANTEN FÜR ATN ***
. ab4a  :

. ab4a  .byte $0b                    Polynomgrad
. ab4b  .byte $76 $b3 $83 $bd $d3  -6.84793912 e-04
. ab50  .byte $79 $1e $f4 $a6 $f5   4.85094216 e-03
. ab55  .byte $7b $83 $fc $b0 $10  -0.0161117018
. ab5a  .byte $7c $0c $1f $67 $ca   0.034209638
. ab5f  .byte $7c $de $53 $cb $c1  -0.0542791328
. ab64  .byte $7d $14 $64 $70 $4c   0.0724571965
. ab69  .byte $7d $b7 $ea $51 $7a  -0.0898023954
. ab6e  .byte $7d $63 $30 $88 $7e   0.110932413
. ab73  .byte $7e $92 $44 $99 $3a  -0.142839808
. ab78  .byte $7e $4c $cc $91 $c7   0.19999912
. ab7d  .byte $7f $aa $aa $aa $13  -0.333333316
. ab82  .byte $81 $00 $00 $00 $00   1

. ab87  :
. ab87  *** VON RENUMBER BERÜCKSICHTIGTE TOKENS ***
```

```
. ab87  :

. ab87  .byte $89              Token 137: GOTO
. ab88  .byte $8a              Token 138: RUN
. ab89  .byte $8d              Token 141: GOSUB
. ab8a  .byte $a7              Token 167: THEN
. ab8b  .byte $8c              Token 140: RESTORE
. ab8c  .byte $d6              Token 214: RESUME
. ab8d  .byte $d7              Token 215: TRAP
. ab8e  .byte $d5              Token 213: ELSE

. ab8f  :
. ab8f  *** R E N U M B E R ***
. ab8f  :
. ab8f  20 de b6 jsr $b6de     RUN-Modus ?, dann Fehler
. ab92  a9 00    lda #$00      Vorbesetzung der Anfangszeile
. ab94  a2 0a    ldx #$0a      ... und Schrittweite auf 10
. ab96  86 03    stx $03       Setzt neue Anfangszeilennummer
. ab98  85 04    sta $04       ... auf 10
. ab9a  86 05    stx $05       Setzt Schrittweite
. ab9c  85 06    sta $06       ... auf 10
. ab9e  85 5a    sta $5a       Setzt Anfangszeilennummer
. aba0  85 5b    sta $5b       ... auf Null
. aba2  20 79 04 jsr $0479     CHRGOT-Routine
. aba5  f0 66    beq $ac0d     Folgt Trennzeichen ?, dann $ac0d

      ÜBERNAHME DER PARAMETER: NEUNUMMER, SCHRITTWEITE, STARTNUMMER

. aba7  20 3e 8e jsr $8e3e     Holt Zeilennummer
. abaa  a5 08    lda $08       Zeilennummer angegeben ?
. abac  f0 08    beq $abb6     Nein, dann $abb6
. abae  a5 14    lda $14       Setzt
. abb0  a6 15    ldx $15       ... Neunummer
. abb2  85 03    sta $03       ... gleich angegebener
. abb4  86 04    stx $04       ... Zeilennummer
. abb6  20 79 04 jsr $0479     CHRGOT-Routine
. abb9  f0 52    beq $ac0d     Folgt Trennzeichen ?, dann $ac0d
. abbb  20 91 94 jsr $9491     (CHKCOM) Prüft ob Komma folgt
. abbe  20 3e 8e jsr $8e3e     Holt Schrittweite
. abc1  a5 08    lda $08       Schrittweite angegeben ?
. abc3  f0 10    beq $abd5     Nein, dann $abd5
. abc5  a5 14    lda $14       Setzt
. abc7  a6 15    ldx $15       ... Schrittweite
. abc9  85 05    sta $05       ... gleich angegebener
. abcb  86 06    stx $06       ... Schrittweite
. abcd  d0 06    bne $abd5     Schrittweite
. abcf  aa       tax           ... nicht Null ?
. abd0  d0 03    bne $abd5     ... dann $abd5
. abd2  4c 1c 99 jmp $991c     'ILLEGAL QUANTITY ERROR' ausgeben

. abd5  20 79 04 jsr $0479     CHRGOT-Routine
. abd8  f0 33    beq $ac0d     Folgt Trennzeichen ?, dann $ac0d
```

```
.abda  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
.abdd  20 3e 8e jsr $8e3e    Holt Zeilennummer
.abe0  a5 14    lda $14      Setzt
.abe2  a6 15    ldx $15      ... Startnummer
.abe4  85 5a    sta $5a      ... gleich angegebener
.abe6  86 5b    stx $5b      ... Zeilennummer
.abe8  20 3d 8a jsr $8a3d    Zeile suchen
.abeb  a5 5f    lda $5f      Zeilen-Adresse
.abed  a6 60    ldx $60      ... gleich
.abef  85 58    sta $58      ... Adresse der
.abf1  86 59    stx $59      ... 1. RENUMBER-Zeile
.abf3  a5 03    lda $03      Holt
.abf5  a6 04    ldx $04      ... Neunummer
.abf7  85 14    sta $14
.abf9  86 15    stx $15
.abfb  20 3d 8a jsr $8a3d    Zeile suchen
.abfe  a5 60    lda $60      Ist Neuadresse
.ac00  38       sec          ... kleiner
.ac01  e5 59    sbc $59      ... Startadresse ?...
.ac03  90 cd    bcc $abd2
.ac05  d0 06    bne $ac0d
.ac07  a5 5f    lda $5f
.ac09  e5 58    sbc $58
.ac0b  90 c5    bcc $abd2    Ja, dann Fehler

     ZIELZEILENNUMMERN IM PROGRAMM KORRIGIEREN

.ac0d  20 f1 8a jsr $8af1    Textpointer initialisieren
.ac10  20 86 ad jsr $ad86    Holt Linkadresse (Low)
.ac13  20 86 ad jsr $ad86    Holt Linkadresse (High)
.ac16  d0 3d    bne $ac55    Keine Endmarke ?, dann $ac55

     ERSETZT ALTE ZEILENNUMMER DURCH NEUE

.ac18  20 59 ad jsr $ad59    Schreibt Zeilennummer nach FAC und Init.Textpointer
.ac1b  20 86 ad jsr $ad86    Holt Linkadresse (Low)
.ac1e  20 86 ad jsr $ad86    Holt Linkadresse (High)
.ac21  d0 03    bne $ac26    Keine Endmarke ?, dann $ac26
.ac23  4c b3 ae jmp $aeb3    Abschluß und READY.

.ac26  20 86 ad jsr $ad86    Holt Zeilennummer (LOW) (LDA($3b),Y)
.ac29  85 14    sta $14
.ac2b  c8       iny
.ac2c  20 a5 04 jsr $04a5    Holt Zeilennummer (High) (LDA($3b),Y)
.ac2f  38       sec          Ist Zeilennummer
.ac30  e5 5b    sbc $5b      ... kleiner
.ac32  90 19    bcc $ac4d    Startnummer ?...
.ac34  d0 06    bne $ac3c
.ac36  a5 14    lda $14
.ac38  e5 5a    sbc $5a
.ac3a  90 11    bcc $ac4d    Ja, dann keine Änderung
.ac3c  a5 62    lda $62      Schreibt
```

```
 ac3e  91 3b    sta ($3b),y  ... Neunummer
. ac4Ø  88       dey          ... in
. ac41  a5 63    lda $63      ... Zeilenkopf...
. ac43  91 3b    sta ($3b),y
. ac45  2Ø 86 ad jsr $ad86    Holt nächstes Zeichen
. ac48  2Ø 73 ad jsr $ad73    Generiert nächste Neunummer, nächste Zeile
. ac4b  fØ ce    beq $ac1b    Springt immer nach $ac1b

. ac4d  2Ø 86 ad jsr $ad86    Holt nächstes Zeichen
. ac5Ø  2Ø 8Ø ad jsr $ad8Ø    Holt nächste Zeile
. ac53  fØ c6    beq $ac1b    Springt immer nach $ac1b

. ac55  2Ø 86 ad jsr $ad86    Holt nächstes Zeichen
. ac58  2Ø 86 ad jsr $ad86    Holt nächstes Zeichen
. ac5b  2Ø 86 ad jsr $ad86    Holt nächstes Zeichen
. ac5e  c9 22    cmp #$22     Zeichen gleich Anführungszeichen (") ?
. ac6Ø  dØ Øb    bne $ac6d    Nein, dann $ac6d
. ac62  2Ø 86 ad jsr $ad86    Holt nächstes Zeichen
. ac65  fØ a9    beq $ac1Ø    Endmarke ?, dann $ac1Ø
. ac67  c9 22    cmp #$22     Noch immer kein Anführungszeichen (") ?
. ac69  dØ f7    bne $ac62    Nein, dann Text überlesen
. ac6b  fØ ee    beq $ac5b    Ja, dann weiter bei $ac5d

      ZEILENNUMMER IM PROGRAMM SUCHEN UND KORRIGIEREN

. ac6d  aa       tax          Ist Zeichen aus Zeile gleich Endmarke ?
. ac6e  fØ aØ    beq $ac1Ø    Ja, dann $ac1Ø
. ac7Ø  1Ø e9    bpl $ac5b    Ist Zeichen kein Token ?, dann $ac5b
. ac72  a2 Ø8    ldx #$Ø8     Token in
. ac74  dd 86 ab cmp $ab86,x  ... Token-Tabelle ($ab87) vorhanden ?
. ac77  fØ 1Ø    beq $ac89    Ja, dann Zeilennummer korrigieren
. ac79  ca       dex
. ac7a  dØ f8    bne $ac74
. ac7c  c9 cb    cmp #$cb     Zeichen gleich 'GO'-Token ?
. ac7e  dØ db    bne $ac5b    Nein, dann $ac5b
. ac8Ø  2Ø 73 Ø4 jsr $Ø473    CHRGET-Routine
. ac83  fØ 8b    beq $ac1Ø    Folgt Trennzeichen ?, dann $ac1Ø
. ac85  c9 a4    cmp #$a4     Zeichen gleich 'TO'-Token ?
. ac87  dØ d2    bne $ac5b    Nein, dann $ac5b

      ZEILENNUMMERN KORRIGIEREN

. ac89  a5 3b    lda $3b      Textpointer
. ac8b  8d 59 Ø2 sta $Ø259    ... zwischenspeichern...
. ac8e  a5 3c    lda $3c
. ac9Ø  8d 5a Ø2 sta $Ø25a
. ac93  2Ø 73 Ø4 jsr $Ø473    CHRGET-Routine
. ac96  bØ c6    bcs $ac5e    Keine Ziffer ?, dann weitersuchen
. ac98  2Ø 3e 8e jsr $8e3e    Holt Zeilennummer
. ac9b  2Ø ee ac jsr $acee    Neue Zeilennummer erzeugen
. ac9e  ad 59 Ø2 lda $Ø259    Textpointer
. aca1  85 3b    sta $3b      ... wiederherstellen...
```

```
. aca3  ad 5a Ø2 lda $Ø25a
. aca6  85 3c    sta $3c
. aca8  aØ ØØ    ldy #$ØØ
. acaa  a2 ØØ    ldx #$ØØ
. acac  bd Ø1 Ø1 lda $Ø1Ø1,x  Liest Zahlenstring (ohne Vorzeichenbyte)
. acaf  fØ 1c    beq $accd    Endmarke ?, dann $accd
. acb1  48       pha          Ziffer auf Stack
. acb2  2Ø 73 Ø4 jsr $Ø473    CHRGET-Routine
. acb5  9Ø Øe    bcc $acc5    Zeichen gleich Ziffer ?, dann keine Verschiebung
. acb7  2Ø 42 ad jsr $ad42    Verschiebung vorbereiten
. acba  e6 6c    inc $6c      Erhöht Schreibzeiger
. acbc  2Ø bb ad jsr $adbb    Verschiebt Programm ein Byte nach hinten
. acbf  e6 2d    inc $2d      Variablenanfang
. acc1  dØ Ø2    bne $acc5    ... ebenfalls
. acc3  e6 2e    inc $2e      ... um 1 Byte erhöhen
. acc5  68       pla          Holt Ziffer aus neuer Nummer
. acc6  aØ ØØ    ldy #$ØØ     Schreibt Ziffer
. acc8  91 3b    sta ($3b),y  ... ins Programm
. acca  e8       inx          Erhöht Zeiger im Nummernstring
. accb  dØ df    bne $acac    Springt immer nach $acac

. accd  2Ø 73 Ø4 jsr $Ø473    CHRGET-Routine
. acdØ  bØ 15    bcs $ace7    Keine Ziffer ?, dann $ace7
. acd2  2Ø 42 ad jsr $ad42    Verschiebung vorbereiten
. acd5  c6 6c    dec $6c      Erniedrigt Schreibzeiger
. acd7  2Ø a2 ad jsr $ada2    Verschiebt Programm ein Byte nach hinten
. acda  a5 2d    lda $2d      Variablenanfang
. acdc  dØ Ø2    bne $aceØ    ... ebenfalls
. acde  c6 2e    dec $2e      ... um 1 Byte erniedrigen
. aceØ  c6 2d    dec $2d
. ace2  2Ø 79 Ø4 jsr $Ø479    CHRGOT-Routine
. ace5  9Ø eb    bcc $acd2    Zeichen gleich Ziffer ?, nochmal vorziehen
. ace7  c9 2c    cmp #$2c     Zeichen gleich Komma (,) ?
. ace9  fØ 9e    beq $ac89    Ja, dann folgt weitere Zeilennummer
. aceb  4c 5e ac jmp $ac5e    Weiter im Programmtext
```

ERZEUGT NEUE ZEILENNUMMER

```
. acee  2Ø 59 ad jsr $ad59    Schreibt Neunummer nach FAC und Textpointer init.
. acf1  2Ø 86 ad jsr $ad86    Holt Linkadresse (Low)
. acf4  dØ Ød    bne $adØ3    Keine Endmarke ?, dann $adØ3
. acf6  2Ø 86 ad jsr $ad86    Holt Linkadresse (High)
. acf9  dØ Øb    bne $adØ6    Keine Endmarke ?, dann $adØ6
. acfb  a9 ff    lda #$ff     Programmende; Zeile nicht vorhanden
. acfd  85 62    sta $62      Setzt Zeilennummer
. acff  85 63    sta $63      ... gleich 65535
. adØ1  3Ø 2a    bmi $ad2d    Springt immer nach $ad2d

. adØ3  2Ø 86 ad jsr $ad86    Holt Linkadresse (High)
. adØ6  2Ø 86 ad jsr $ad86    Holt Zeilennummer (Low)
. adØ9  85 58    sta $58
```

```
. ad0b  c5 14     cmp $14       Zeilennummer gleich alte Zeilennummer (Low) ?
. ad0d  d0 27     bne $ad36     Nein, dann nächste Zeile
. ad0f  20 86 ad  jsr $ad86     Holt Zeilennummer (High)
. ad12  85 59     sta $59
. ad14  c5 15     cmp $15       Zeilennummer gleich alte Zeilennummer (High) ?
. ad16  d0 23     bne $ad3b     Nein, dann nächste Zeile
. ad18  38        sec
. ad19  e5 5b     sbc $5b       Befindet
. ad1b  90 08     bcc $ad25     ... sich
. ad1d  d0 0e     bne $ad2d     ... Zeile
. ad1f  a5 14     lda $14       ... im
. ad21  e5 5a     sbc $5a       ... RENUMBER-Bereich ?
. ad23  b0 08     bcs $ad2d     Ja, dann neue Nummer schon in FAC
. ad25  a5 14     lda $14       Schreibt alte
. ad27  85 63     sta $63       ... Zielzeilennummer
. ad29  a5 15     lda $15       ... nach
. ad2b  85 62     sta $62       ... FAC
. ad2d  a2 90     ldx #$90      Holt Exponent
. ad2f  38        sec           Carry verhindert Negierung
. ad30  20 ce a2  jsr $a2ce     FAC aufbereiten
. ad33  4c 6f a4  jmp $a46f     Gleitkomma-String-Wandlung

. ad36  20 86 ad  jsr $ad86     Holt Zeilennummer (High)
. ad39  85 59     sta $59
. ad3b  20 64 ad  jsr $ad64     Nächste Neunummer, nächste Zeile
. ad3e  f0 b1     beq $acf1     Springt immer nach $acf1

. ad40  .byte $d9
. ad41  .byte $ea

     VERSCHIEBUNG VORBEREITEN

. ad42  a5 3b     lda $3b       Setzt Textpointer
. ad44  85 22     sta $22       ... auf
. ad46  a5 3c     lda $3c       ... Anfang
. ad48  85 23     sta $23       ... des Verschiebbereichs
. ad4a  a5 2d     lda $2d       Setzt
. ad4c  85 24     sta $24       ... Variablenanfang
. ad4e  a5 2e     lda $2e       ... gleich
. ad50  85 25     sta $25       ... Verschiebbereich-Ende
. ad52  a0 00     ldy #$00      Initialisiert
. ad54  84 0b     sty $0b       ... Lese- und
. ad56  84 6c     sty $6c       ... Schreibzeiger
. ad58  60        rts           ENDE

     FAC := NEUE ANFANGSZEILENNUMMER und TEXTPOINTER INITIALISIEREN

. ad59  a5 03     lda $03       Schreibt
. ad5b  85 63     sta $63       ... neue
. ad5d  a5 04     lda $04       ... Anfangszeilennummer
. ad5f  85 62     sta $62       ... nach FAC
. ad61  4c f1 8a  jmp $8af1     Textpointer initialisieren
```

```
      ERHÖHT ZEILENNUMMER UM SCHRITTWEITE

. ad64  a5 59    lda $59      Befindet
. ad66  38       sec          ... sich
. ad67  e5 5b    sbc $5b      ... Zeilennummer
. ad69  90 15    bcc $ad80    ... im
. ad6b  d0 06    bne $ad73    ... RENUMBER-
. ad6d  a5 58    lda $58      ... Bereich ?...
. ad6f  e5 5a    sbc $5a
. ad71  90 0d    bcc $ad80    Nein, dann $ad80
. ad73  a5 63    lda $63      Addiert
. ad75  18       clc          ... im FAC
. ad76  65 05    adc $05      ... Schrittweite
. ad78  85 63    sta $63      ... zur
. ad7a  a5 62    lda $62      ... Zeilennummer...
. ad7c  65 06    adc $06
. ad7e  85 62    sta $62
. ad80  20 86 ad jsr $ad86    Liest nächstes Zeichen
. ad83  d0 fb    bne $ad80    Keine Endmarke ?, dann $ad80
. ad85  60       rts          ENDE

      NÄCHSTES ZEICHEN AUS PRG LESEN

. ad86  a0 00    ldy #$00
. ad88  e6 3b    inc $3b      Textpointer
. ad8a  d0 02    bne $ad8e    ... um 1
. ad8c  e6 3c    inc $3c      ... erhöhen
. ad8e  4c a5 04 jmp $04a5    Liest nächstes Zeichen (LDA($3b),Y)

      HILFSZEIGER 1 UND 2 VERGLEICHEN

. ad91  a5 22    lda $22      Vergleicht
. ad93  c5 24    cmp $24      ... Hilfszeiger 1
. ad95  d0 04    bne $ad9b    ... mit
. ad97  a5 23    lda $23      ... Hilfszeiger 2...
. ad99  c5 25    cmp $25
. ad9b  60       rts          ENDE

      VERSCHIEBUNG EINES PROGRAMMBLOCKS NACH VORNE

. ad9c  e6 22    inc $22      Erhöht
. ad9e  d0 02    bne $ada2    ... Anfangszeiger
. ada0  e6 23    inc $23      ... um 1
. ada2  a4 0b    ldy $0b      Erhöht
. ada4  c8       iny          ... Lesezeiger um 1
. ada5  20 b0 04 jsr $04b0    Liest Zeichen (LDA($22),Y)
. ada8  a4 6c    ldy $6c      Erhöht
. adaa  c8       iny          ... Schreibzeiger um 1
. adab  91 22    sta ($22),y  Schreibt Zeichen zurück
. adad  20 91 ad jsr $ad91    Ende erreicht ? (Hilfszeiger vergleichen)
. adb0  d0 ea    bne $ad9c    Nein, dann weiter verschieben
```

```
. adb2  60       rts          ENDE

     VERSCHIEBUNG EINES PROGRAMMBLOCKS NACH HINTEN

. adb3  a5 24    lda $24      Erniedrigt
. adb5  d0 02    bne $adb9    ... Endzeiger
. adb7  c6 25    dec $25      ... um 1
. adb9  c6 24    dec $24
. adbb  a4 0b    ldy $0b      Holt Lesezeiger
. adbd  20 bb 04 jsr $04bb    Liest Zeichen (LDA($24),Y)
. adc0  a4 6c    ldy $6c      Holt Schreibzeiger
. adc2  91 24    sta ($24),y  Schreibt Zeichen zurück
. adc4  20 91 ad jsr $ad91    Ende erreicht ? (Hilfszeiger vergleichen)
. adc7  d0 ea    bne $adb3    Nein, dann weiter verschieben
. adc9  60       rts          ENDE

  adca  :
  adca  *** F O R ***
  adca  :
. adca  a9 80    lda #$80     Sperrt
. adcc  85 10    sta $10      ... Integervariable
. adce  20 7c 8e jsr $8e7c    LET (Variable ablegen)
. add1  a9 81    lda #$81     Holt 'FOR'-Token
. add3  85 02    sta $02      Speichert Token für Suche im Stack
. add5  20 71 88 jsr $8871    Sucht 'FOR'-Token im Basic-Stack
. add8  f0 08    beq $ade2    Gefunden ?, dann $ade2
. adda  a0 12    ldy #$12     Ist im Basic-Stack
. addc  20 05 89 jsr $8905    ... Platz für 18 ($12) Byte ?
. addf  20 60 a7 jsr $a760    Hilfszeiger := Stackzeiger
. ade2  20 69 a7 jsr $a769    Stackzeiger := Hilfszeiger
. ade5  20 be 8d jsr $8dbe    Y-Reg. gleich Offset zum Trennzeichen
. ade8  98       tya          Offset-Zeiger nach Akku
. ade9  a0 11    ldy #$11
. adeb  18       clc
. adec  65 3b    adc $3b      Addiert Offset
. adee  91 7c    sta ($7c),y  ... zum Textpointer
. adf0  a5 3c    lda $3c      ... und Ergebnis
. adf2  69 00    adc #$00     ... (gleich Anfang der FOR-NEXT-Schleife)
. adf4  88       dey          ... auf Basic-Stack
. adf5  91 7c    sta ($7c),y  ... legen.
. adf7  a5 3a    lda $3a      Legt
. adf9  88       dey          ... Zeilennummer
. adfa  91 7c    sta ($7c),y  ... auf
. adfc  a5 39    lda $39      ... Basic-
. adfe  88       dey          ... Stack...
. adff  91 7c    sta ($7c),y
. ae01  a9 a4    lda #$a4     Holt 'TO'-Token
. ae03  20 93 94 jsr $9493    Prüft ob 'TO'-Token folgt
. ae06  20 17 93 jsr $9317    (CHKNUM) Prüft ob numerisch
. ae09  20 14 93 jsr $9314    (FRMNUM) TO-Ausdruck auswerten
. ae0c  a5 66    lda $66      Holt Vorzeichenbyte von FAC
. ae0e  09 7f    ora #$7f     Vorzeichen in
```

```
. ae10  25 62     and $62       ... höchstes Bit der
. ae12  85 62     sta $62       ... Mantisse übertragen
. ae14  a2 04     ldx #$04      Holt
. ae16  a0 0d     ldy #$0d      ... TO-Wert
. ae18  b5 61     lda $61,x     ... von FAC
. ae1a  91 7c     sta ($7c),y   ... und
. ae1c  ca        dex           ... schreibt
. ae1d  88        dey           ... diesen in
. ae1e  10 f8     bpl $ae18     ... Basic-Stack
. ae20  a9 f0     lda #$f0      Setzt Zeiger auf Konstante 1
. ae22  a0 9f     ldy #$9f      ... (Vorbelegung des STEP-Wertes)
. ae24  20 21 a2  jsr $a221     Holt Konstante 1 nach FAC
. ae27  20 79 04  jsr $0479     CHRGOT-Routine
. ae2a  c9 a9     cmp #$a9      Zeichen gleich 'STEP'-Token ?
. ae2c  d0 06     bne $ae34     Nein, dann $ae34
. ae2e  20 73 04  jsr $0473     CHRGET-Routine
. ae31  20 14 93  jsr $9314     (FRMNUM) STEP-Ausdruck auswerten
. ae34  20 b0 a2  jsr $a2b0     FAC-Vorzeichen ermitteln
. ae37  48        pha
. ae38  20 a0 a2  jsr $a2a0     FAC runden
. ae3b  68        pla
. ae3c  a0 08     ldy #$08      Kopiert
. ae3e  a2 05     ldx #$05      ... Vorzeichen
. ae40  91 7c     sta ($7c),y   ... und
. ae42  b5 60     lda $60,x     ... FAC
. ae44  88        dey           ... auf
. ae45  ca        dex           ... Basic-
. ae46  10 f8     bpl $ae40     ... Stack
. ae48  a5 4a     lda $4a       Kopiert
. ae4a  91 7c     sta ($7c),y   ... FOR-NEXT-
. ae4c  a5 49     lda $49       ... Variablenzeiger
. ae4e  88        dey           ... auf
. ae4f  91 7c     sta ($7c),y   ... Basic-Stack
. ae51  a9 81     lda #$81      Schreibt
. ae53  88        dey           ... 'FOR'-Token
. ae54  91 7c     sta ($7c),y   ... auf Basic-Stack
. ae56  60        rts           ENDE

. ae57  4c a1 94  jmp $94a1     'SYNTAX ERROR' ausgeben

  ae5a  :
  ae4a  *** D E L E T E ***
  ae5a  :
. ae5a  20 de b6  jsr $b6de     Kein Direktmodus ?, dann Fehler
. ae5d  20 79 04  jsr $0479     CHRGOT-Routine
. ae60  f0 f5     beq $ae57     Kein Zeilenbereich angegeben ?, dann Fehler
. ae62  20 ca ae  jsr $aeca     Zeilenbereichsanalyse
. ae65  a5 5f     lda $5f       Anfangsadresse
. ae67  a6 60     ldx $60       ... des zu löschenden
. ae69  85 24     sta $24       ... Bereichs in
. ae6b  86 25     stx $25       ... Hilfszeiger ablegen
. ae6d  20 3d 8a  jsr $8a3d     Holt Endzeilenadresse
```

```
. ae7Ø  9Ø 15     bcc $ae87    Keine Endzeile ?, dann $ae87
. ae72  aØ Ø1     ldy #$Ø1
. ae74  2Ø d1 Ø4  jsr $Ø4d1    Holt Linkadresse (High) (LDA$(5f),Y)
. ae77  88        dey
. ae78  aa        tax
. ae79  dØ Ø5     bne $ae8Ø
. ae7b  2Ø d1 Ø4  jsr $Ø4d1    Holt Linkadresse (Low) (LDA$(5f),Y)
. ae7e  fØ Ø7     beq $ae87    Programmende ?, dann $ae87
. ae8Ø  2Ø d1 Ø4  jsr $Ø4d1    Indirekter Speicherzugriff (LDA($5f),Y)
. ae83  85 5f     sta $5f      Schreibt Ende
. ae85  86 6Ø     stx $6Ø      ... des Löschbereichs
. ae87  a5 24     lda $24      Ist
. ae89  38        sec          ... Anfangsadresse
. ae8a  e5 5f     sbc $5f      ... kleiner
. ae8c  aa        tax          ... als
. ae8d  a5 25     lda $25      ... Endadresse ?...
. ae8f  e5 6Ø     sbc $6Ø
. ae91  a8        tay
. ae92  bØ 1f     bcs $aeb3    Nein, dann fertig
. ae94  8a        txa          Erniedrigt
. ae95  18        clc          ... Variablenanfangs-
. ae96  65 2d     adc $2d      ... Zeiger
. ae98  85 2d     sta $2d      ... um
. ae9a  98        tya          ... Länge
. ae9b  65 2e     adc $2e      ... des
. ae9d  85 2e     sta $2e      ... Löschbereichs
. ae9f  aØ ØØ     ldy #$ØØ
. aea1  2Ø d1 Ø4  jsr $Ø4d1    Indirekter Speicherzugriff (LDA($5f),Y)
. aea4  91 24     sta ($24),y  Kopiert
. aea6  c8        iny          ... oberen Programmabschnitt
. aea7  dØ f8     bne $aea1    ... nach unten
. aea9  e6 6Ø     inc $6Ø      Erhöht Quellzeiger (High)
. aeab  e6 25     inc $25      Erhöht Zielzeiger (High)
. aead  a5 2e     lda $2e      Holt neues Programmende (High)
. aeaf  c5 25     cmp $25      Programmende (High) kleiner Zielzeiger (High) ?
. aeb1  bØ ee     bcs $aea1    Nein, dann Kopieren fortsetzen
. aeb3  2Ø 18 88  jsr $8818    Linkadresse erzeugen
. aeb6  a5 22     lda $22      Variablenanfang
. aeb8  a6 23     ldx $23      ... gleich
. aeba  18        clc          ... Programmende
. aebb  69 Ø2     adc #$Ø2     ... plus 2...
. aebd  85 2d     sta $2d
. aebf  9Ø Ø1     bcc $aec2
. aec1  e8        inx
. aec2  86 2e     stx $2e
. aec4  2Ø 93 8a  jsr $8a93    RESTORE, CLR
. aec7  4c 7e 86  jmp $867e    READY.

  aeca  :
  aeca  *** ZEILENBEREICHSANALYSE ***
  aeca  :
. aeca  fØ Ø6     beq $aed2    Folgt Trennzeichen ?, dann $aed2
```

```
. aecc  90 04    bcc $aed2     Zeichen gleich Ziffer ?, dann $aed2
. aece  c9 ab    cmp #$ab      Zeichen gleich '-'-Token ?
. aed0  d0 22    bne $aef4     Nein, dann Fehler
. aed2  20 3e 8e jsr $8e3e     Zeilennummer in Adreßformat wandeln
. aed5  20 3d 8a jsr $8a3d     Zeile suchen
. aed8  20 79 04 jsr $0479     CHRGOT-Routine
. aedb  f0 0c    beq $aee9     Folgt Trennzeichen ?, dann $aee9
. aedd  c9 ab    cmp #$ab      Zeichen gleich '-'-Token ?
. aedf  d0 13    bne $aef4     Nein, dann Fehler
. aee1  20 73 04 jsr $0473     CHRGET-Routine
. aee4  20 3e 8e jsr $8e3e     Zeilennummer in Adreßformat wandeln
. aee7  d0 0b    bne $aef4     Folgt kein Trennzeichen ?, dann Fehler
. aee9  a5 08    lda $08       Zeilennummer angegeben ?
. aeeb  d0 06    bne $aef3     Ja, dann $aef3
. aeed  a9 ff    lda #$ff      Sonst
. aeef  85 14    sta $14       ... Zeilennummer
. aef1  85 15    sta $15       ... 65535 übergeben
. aef3  60       rts           ENDE

. aef4  4c a1 94 jmp $94a1     'SYNTAX ERROR' ausgeben

. aef7  :
. aef7  *** U S I N G ***
. aef7  :
. aef7  a2 ff    ldx #$ff      Vorbelegung für
. aef9  8e e0 02 stx $02e0     ... Feldende
. aefc  20 73 04 jsr $0473     CHRGET-Routine
. aeff  20 2c 93 jsr $932c     (FRMEVL) Formatstring auswerten
. af02  20 1a 93 jsr $931a     (CHKSTR) Prüft ob String
. af05  a5 64    lda $64       Deskriptor-Zeiger
. af07  48       pha           ... des
. af08  a5 65    lda $65       ... Formatstrings auf
. af0a  48       pha           ... Stack legen
. af0b  a0 02    ldy #$02
. af0d  20 dc 04 jsr $04dc     Holt Adresse des Formatstrings (LDA($64),Y)
. af10  88       dey
. af11  99 3d 00 sta $003d,y   Schreibt Adresse in Formatstringzeiger...
. af14  d0 f7    bne $af0d
. af16  20 dc 04 jsr $04dc     Holt Länge des Formatstrings (LDA($64),Y)
. af19  8d df 02 sta $02df     Schreibt Länge des Formatstrings
. af1c  a8       tay           Stringlänge gleich Null ?
. af1d  f0 0b    beq $af2a     Ja, dann Fehler
. af1f  88       dey
. af20  20 71 81 jsr $8171     Holt Zeichen aus Formatstring (LDA($3d),Y)
. af23  c9 23    cmp #$23      Zeichen gleich '#' ?
. af25  f0 06    beq $af2d     Ja, dann $af2d
. af27  98       tya
. af28  d0 f5    bne $af1f     Sonst weiterlesen

. af2a  4c a1 94 jmp $94a1     'SYNTAX ERROR' ausgeben

. af2d  a9 3b    lda #$3b      Strichpunkt (;) muß auf Formatstring folgen
```

```
. af2f  20 93 94 jsr $9493    Prüft ob Strichpunkt folgt
. af32  84 76    sty $76      Y-Reg. gleich Null
. af34  8c cd 02 sty $02cd    Offset im Zahlenstring auf Null setzen
. af37  20 2c 93 jsr $932c    (FRMEVL) Ausgabeausdruck auswerten
. af3a  24 0d    bit $0d      Prüft ob String
. af3c  10 39    bpl $af77    Nein, dann $af77
. af3e  20 70 b1 jsr $b170    (FRESTR) und Parameter initialisieren
. af41  20 b7 b2 jsr $b2b7    Formatstring auswerten
. af44  ae d5 02 ldx $02d5    '='-Zeichen oder Exponentsymbol im String ?
. af47  f0 15    beq $af5e    Nein, dann $af5e
. af49  a2 00    ldx #$00
. af4b  38       sec
. af4c  ad db 02 lda $02db    Subtrahiert Länge des
. af4f  e5 77    sbc $77      ... Ausgabestrings vom Formatzähler
. af51  90 0b    bcc $af5e    Wenn Ausgabestring länger, dann $af5e
. af53  a2 3d    ldx #$3d     Justierungs-Flag gleich '='-Zeichen ?
. af55  ec d5 02 cpx $02d5    ... (d.h. zentrieren)
. af58  d0 03    bne $af5d    Nein, dann $af5d
. af5a  4a       lsr          Halbiert Akku
. af5b  69 00    adc #$00     ... und rundet auf
. af5d  aa       tax          Zahl der Vorlauf-Blanks nach X-Reg.
. af5e  a0 00    ldy #$00     Initialisiert Offset im Ausgabestring
. af60  8a       txa          Noch mehr Leerzeichen ausgeben ?
. af61  f0 05    beq $af68    Nein, dann $af68
. af63  ca       dex          Blank-Zähler um 1 erniedrigen
. af64  a9 20    lda #$20     Holt Leerzeichen ' '
. af66  d0 08    bne $af70    Leerzeichen ausgeben

. af68  c4 77    cpy $77      Ende des Ausgabestrings ?
. af6a  b0 f8    bcs $af64    Ja, dann $af64
. af6c  20 b0 04 jsr $04b0    Holt Zeichen aus Ausgabestring (LDA($22),Y)
. af6f  c8       iny          Offset in Ausgabestring um 1 erhöhen
. af70  20 b0 b2 jsr $b2b0    Zeichen ausgeben und Formatzeiger erniedrigen
. af73  d0 eb    bne $af60    Formatzähler ungleich Null ?, dann $af60
. af75  f0 24    beq $af9b    Formatzähler gleich 0, dann $af9b

. af77  20 6f a4 jsr $a46f    Gleitkomma-String-Wandlung
. af7a  a0 ff    ldy #$ff     Überprüft
. af7c  c8       iny          ... Länge
. af7d  b9 00 01 lda $0100,y  ... des
. af80  d0 fa    bne $af7c    ... Zahlenstrings
. af82  98       tya          Länge des Zahlenstrings nach Akku
. af83  20 5c 9b jsr $9b5c    Reserviert Platz für Zahlenstring
. af86  a0 00    ldy #$00     Kopiert
. af88  b9 00 01 lda $0100,y  ... Zahlenstring
. af8b  f0 05    beq $af92    ... in
. af8d  91 62    sta ($62),y  ... Stringbereich...
. af8f  c8       iny
. af90  d0 f6    bne $af88
. af92  20 b0 9b jsr $9bb0    Deskriptor in String-Stack
. af95  20 70 b1 jsr $b170    (FRESTR) und Parameter initialisieren
. af98  20 bb af jsr $afbb    Zahlenstring formatieren und ausgeben
```

```
. af9b  20 79 04 jsr $0479    CHRGOT-Routine
. af9e  c9 2c    cmp #$2c     Weiter ausgeben ? (Wenn Komma folgt)
. afa0  f0 8d    beq $af2f    Ja, dann $af2f
. afa2  38       sec          Setzt Flag
. afa3  66 76    ror $76      ... für Abbruch der Ausgabe
. afa5  20 b7 b2 jsr $b2b7    Gibt restliche Zeichen aus Formatstring aus
. afa8  68       pla          Holt Zeiger auf
. afa9  a8       tay          ... Deskriptor des Formatstrings
. afaa  68       pla          ... vom Stack
. afab  20 52 9c jsr $9c52    (FRESTR)
. afae  20 79 04 jsr $0479    CHRGOT-Routine
. afb1  c9 3b    cmp #$3b     Folgt Strichpunkt (;) ?
. afb3  f0 03    beq $afb8    Ja, dann kein Zeilenvorschub
. afb5  4c 3e 90 jmp $903e    Gibt Zeilenvorschub aus

. afb8  4c 73 04 jmp $0473    CHRGET-Routine

     ZAHLENSTRING FORMATIEREN

. afbb  ad e7 04 lda $04e7    SPACE oder PUDEF-Ersatz
. afbe  8d dd 02 sta $02dd    ... als Füllzeichen speichern
. afc1  a9 ff    lda #$ff     Setzt Vorzeichen-Flag auf $FF
. afc3  8d dc 02 sta $02dc    ... und verhindert somit Vorzeichen-Ausgabe
. afc6  4c cb af jmp $afcb    Springt nach $afcb

. afc9  86 82    stx $82      Offset der 1. Nachkommastelle
. afcb  c4 77    cpy $77      Gleich Ende des Zahlenstrings ?
. afcd  f0 33    beq $b002    Ja, dann $b002
. afcf  b9 00 01 lda $0100,y  Holt Zeichen aus Zahlenstring
. afd2  c8       iny          Erhöht Offset-Zeiger um 1
. afd3  c9 20    cmp #$20     Zeichen gleich Leerzeichen ?
. afd5  f0 f4    beq $afcb    Ja, dann Zeichen überlesen
. afd7  c9 2d    cmp #$2d     Zeichen gleich Minuszeichen (-) ?
. afd9  f0 e8    beq $afc3    Ja, dann Zeichen in Vorzeichen-Flag speichern
. afdb  c9 2e    cmp #$2e     Zeichen gleich Dezimalpunkt (.) ?
. afdd  f0 ea    beq $afc9    Ja, dann Offset der 1. Nachkommastelle merken
. afdf  c9 45    cmp #$45     Zeichen gleich Exponentzeichen ?
. afe1  f0 11    beq $aff4    Ja, dann $aff4
. afe3  9d 00 01 sta $0100,x  Ziffer zurückschreiben
. afe6  8e ce 02 stx $02ce    Schreibt Zeiger auf Ende des Zahlenstrings
. afe9  e8       inx          Erhöht Schreib-Offset um 1
. afea  24 82    bit $82      Dezimalpunkt schon gefunden ?
. afec  10 dd    bpl $afcb    Ja, dann $afcb
. afee  ee d4 02 inc $02d4    Vorkomma-Stellenzähler um 1 erhöhen
. aff1  4c cb af jmp $afcb    Nächstes Zeichen auswerten

. aff4  b9 00 01 lda $0100,y  Holt nächstes Zeichen
. aff7  c9 2d    cmp #$2d     Zeichen gleich Minuszeichen (-) ?
. aff9  d0 03    bne $affe    Nein, dann Exponent positiv
. affb  6e d2 02 ror $02d2    Setzt Negativ-Flag im Exponenten
. affe  c8       iny          Erhöht Offset im Zahlenstring
. afff  8e d3 02 stx $02d3    Schreibt Offset der 1. Exponenten-Ziffer
```

```
. b002  a5 82     lda $82       Dezimalpunkt gefunden ?
. b004  10 02     bpl $b008     Ja, dann $b008
. b006  86 82     stx $82       Schreibt 1. Nachkommastelle hinter Mantisse
. b008  20 b7 b2  jsr $b2b7     Formatstring auswerten
. b00b  ad d6 02  lda $02d6     Holt Anzahl der Vorkommastellen
. b00e  c9 ff     cmp #$ff      Nur Dollarzeichen ($) ohne Vorkommastellen ?
. b010  f0 29     beq $b03b     Ja, dann Sternchen (*) ausgeben
. b012  ad d9 02  lda $02d9     Exponent im Formatstring vorhanden ?
. b015  f0 3f     beq $b056     Nein, dann $b056
. b017  ad d3 02  lda $02d3     Exponent im Zahlenstring vorhanden ?
. b01a  d0 12     bne $b02e     Ja, dann $b02e
. b01c  ae ce 02  ldx $02ce     Holt Zeiger auf Ende des Zahlenstrings
. b01f  20 45 b1  jsr $b145     Schreibt '01' als Exponenten
. b022  de 02 01  dec $0102,x   Macht aus '01' den Exponenten '00'
. b025  e8        inx           Erhöht Offset im Zahlenstring
. b026  8e d3 02  stx $02d3     Zeiger auf Exponenten abspeichern
. b029  20 cc b1  jsr $b1cc     Exponenten-Darstellung normalisieren
. b02c  f0 25     beq $b053     Zahl gleich Null ?, dann $b053
. b02e  ac d8 02  ldy $02d8     Vorzeichen im Formatstring enthalten ?
. b031  d0 17     bne $b04a     Ja, dann $b04a
. b033  ac dc 02  ldy $02dc     Minuszeichen vor dem Zahlenstring ?
. b036  30 12     bmi $b04a     Nein, dann $b04a
. b038  ad d6 02  lda $02d6     Vorkommastellen im Formatstring ?
. b03b  f0 6a     beq $b0a7     Nein, dann Sternchen (*) ausgeben
. b03d  ce d6 02  dec $02d6     Vorkommastellenzähler um 1 erniedrigen
. b040  d0 05     bne $b047     Noch nicht Null ?, dann $b047
. b042  ad d7 02  lda $02d7     Nachkommastellen im Formatstring ?
. b045  f0 60     beq $b0a7     Nein, dann Sternchen (*) ausgeben
. b047  ee d1 02  inc $02d1     Zähler um 1 erniedrigen
. b04a  20 bf b0  jsr $b0bf     Formatiert Zahl
. b04d  20 8a b1  jsr $b18a     Zahl nach Formatanweisung runden
. b050  20 bf b0  jsr $b0bf     Formatiert Zahl
. b053  4c ed b1  jmp $b1ed     Gibt Zahl mit Zusätzen aus

. b056  ac d3 02  ldy $02d3     Exponent im Zahlenstring ?
. b059  f0 16     beq $b071     Nein, dann $b071
. b05b  85 77     sta $77       (Akku := 0)
. b05d  38        sec           Setzt Flag für
. b05e  6e da 02  ror $02da     ... Exponent entfernen
. b061  a4 82     ldy $82       Holt Offset der 1. Nachkommastelle
. b063  ad d2 02  lda $02d2     Holt Vorzeichen des Exponenten
. b066  10 06     bpl $b06e     Vorzeichen positiv (+) ?, dann $b06e
. b068  20 f8 b0  jsr $b0f8     Erhöht Exponent auf Null
. b06b  4c 7a b0  jmp $b07a     Springt nach $b07a

. b06e  20 d9 b0  jsr $b0d9     Erniedrigt Exponent auf Null
. b071  a4 82     ldy $82       Holt Offset der 1. Nachkommastelle
. b073  f0 05     beq $b07a     Keine Vorkommastelle ?, dann $b07a
. b075  20 d0 b1  jsr $b1d0     Ist 1. Vorkommastelle gleich Null ?
. b078  f0 06     beq $b080     Ja, dann $b080
. b07a  20 8a b1  jsr $b18a     Zahl nach Formatanweisung runden
. b07d  4c 83 b0  jmp $b083     Springt nach $b083
```

```
. b080  ce d4 02 dec $02d4    Erniedrigt Zahl der Vorkommastellen um 1
. b083  38       sec          Subtrahiert Vorkommastellen
. b084  ad d6 02 lda $02d6    ... in der Zahl von den
. b087  ed d4 02 sbc $02d4    ... Vorkommastellen im Formatstring
. b08a  90 1b    bcc $b0a7    Wenn Ergebnis negativ, dann Sternchen ausgeben
. b08c  8d d1 02 sta $02d1    Differenz in Zähler zwischenspeichern
. b08f  ac d8 02 ldy $02d8    Vorzeichen im Formatstring vorhanden ?
. b092  d0 1b    bne $b0af    Ja, dann $b0af
. b094  ac dc 02 ldy $02dc    Vorzeichen im Zahlenstring vorhanden ?
. b097  30 16    bmi $b0af    Nein, dann $b0af
. b099  a8       tay          Vorkommastellen-Differenz gleich Null ?
. b09a  f0 0b    beq $b0a7    Ja, dann Sternchen (*) ausgeben
. b09c  88       dey          Vorkommastellen-Differenz gleich 1 ?
. b09d  d0 13    bne $b0b2    Nein, dann $b0b2
. b09f  ad d7 02 lda $02d7    Sind Nachkommastellen im Formatstring oder
. b0a2  0d d4 02 ora $02d4    ... Vorkommastellen im Zahlenstring vorhanden ?
. b0a5  d0 ac    bne $b053    Ja, dann Zahl ausgeben
. b0a7  a9 2a    lda #$2a     Holt Sternchen (*)
. b0a9  20 b0 b2 jsr $b2b0    Zeichen ausgeben und Formatzähler erniedrigen
. b0ac  d0 fb    bne $b0a9    Zähler noch nicht Null ?, dann $b0a9
. b0ae  60       rts          ENDE

. b0af  a8       tay          Vorkommastellen-Differenz gleich Null ?
. b0b0  f0 a1    beq $b053    Ja, dann Zahl ausgeben
. b0b2  ad d4 02 lda $02d4    Vorkommastellen im Zahlenstring vorhanden ?
. b0b5  d0 9c    bne $b053    Ja, dann Zahl ausgeben
. b0b7  ce d1 02 dec $02d1    Zähler (Vorkommastellen-Differenz) erniedrigen
. b0ba  e6 76    inc $76
. b0bc  4c 53 b0 jmp $b053    Zahl ausgeben

      EXPONENTIAL-DARSTELLUNG ERZEUGEN

. b0bf  38       sec          Subtrahiert Vorkommastellen
. b0c0  ad d6 02 lda $02d6    ... in der Zahl von den
. b0c3  ed d4 02 sbc $02d4    ... Vorkommastellen im Formatstring
. b0c6  f0 39    beq $b101    Differenz gleich Null ?, dann fertig
. b0c8  a4 82    ldy $82      Holt Offset der 1. Nachkommastelle
. b0ca  90 16    bcc $b0e2    Mehr Vorkommastellen in Zahl ?, dann $b0e2
. b0cc  85 77    sta $77      Schreibt Differenz in Zähler
. b0ce  cc ce 02 cpy $02ce    1. Nachkommastelle
. b0d1  f0 02    beq $b0d5    ... gleich Ende des Zahlenstrings ?, dann $b0d5
. b0d3  b0 01    bcs $b0d6    ... hinter der letzten Stelle ?, dann $b0d6
. b0d5  c8       iny
. b0d6  ee d4 02 inc $02d4    Erhöht Zahl der Vorkommastellen um 1
. b0d9  20 0e b1 jsr $b10e    Erniedrigt Exponent
. b0dc  c6 77    dec $77      Erniedrigt Differenzzähler um 1
. b0de  d0 ee    bne $b0ce    Noch nicht Null ?, dann weitermachen
. b0e0  f0 1d    beq $b0ff    Sonst, fertig

. b0e2  49 ff    eor #$ff     Differenz
. b0e4  69 01    adc #$01     ... invertieren und
```

```
. b0e6  85 77     sta $77       ... in Zähler schreiben
. b0e8  cc cd 02  cpy $02cd     Ist 1. Nachkommastelle gleich Anfang
. b0eb  f0 07     beq $b0f4     ... des Zahlenstrings ?, dann $b0f4
. b0ed  88        dey
. b0ee  ce d4 02  dec $02d4     Vorkommastellen im Zahlenstring erniedrigen
. b0f1  4c f6 b0  jmp $b0f6     Springt nach $b0f6

. b0f4  e6 76     inc $76       Erhöht Abbruch-Flag um 1
. b0f6  a9 80     lda #$80      Vorgabe des Exponenten-Vorzeichens
. b0f8  20 10 b1  jsr $b110     Erhöht Exponenten
. b0fb  c6 77     dec $77       Erniedrigt Differenzzähler um 1
. b0fd  d0 e9     bne $b0e8     Noch nicht gleich Null ?, dann $b0e8
. b0ff  84 82     sty $82       Schreibt Offset der 1. Nachkommastelle
. b101  60        rts           ENDE

. b102  d0 39     bne $b13d     Kein Überlauf in Exponenten-Ziffer ?,dann $b13d
. b104  49 09     eor #$09      Macht aus '9' eine '0' und umgekehrt
. b106  9d 00 01  sta $0100,x   Schreibt Ergebnis in Zahlenstring
. b109  ca        dex           Erniedrigt Offset im Zahlenstring
. b10a  ec d3 02  cpx $02d3     Exponenten-Anfang erreicht ?
. b10d  60        rts           ENDE

. b10e  a9 00     lda #$00      Vorgabe des Exponenten-Vorzeichens
. b110  ae d3 02  ldx $02d3     Holt Offset zur Zehnerstelle des Exponenten
. b113  e8        inx           Exponent auf Einerstelle erhöhen
. b114  2c da 02  bit $02da     Soll Exponent entfernt werden ?
. b117  30 10     bmi $b129     Ja, dann $b129
. b119  4d d2 02  eor $02d2     Ist Vorzeichen in Zahlenstring gleich Vorgabe ?
. b11c  f0 0b     beq $b129     Ja, dann $b129
. b11e  20 53 b1  jsr $b153     Erhöht Exponent-Ziffer
. b121  20 02 b1  jsr $b102     Wenn nötig, dann Übertrag
. b124  b0 f8     bcs $b11e     Übertrag bei Einerstelle ?,dann $b11e
. b126  4c b2 9f  jmp $9fb2     'OVERFLOW ERROR' ausgeben

. b129  bd 00 01  lda $0100,x   Erniedrigt
. b12c  de 00 01  dec $0100,x   ... Exponent-Ziffer
. b12f  c9 30     cmp #$30      War Ziffer gleich Null ?
. b131  20 02 b1  jsr $b102     Wenn nötig, dann Übertrag
. b134  b0 f3     bcs $b129     Übertrag bei Einerstelle ?, dann $b129
. b136  2c da 02  bit $02da     Soll Exponent entfernt werden ?
. b139  10 05     bpl $b140     Nein, dann $b140
. b13b  84 82     sty $82       Schreibt Offset der ersten Nachkommastelle
. b13d  68        pla           Rücksprung auf
. b13e  68        pla           ... nächsthöhere Ebene
. b13f  60        rts           ENDE

. b140  ad d2 02  lda $02d2     Wechselt Vorzeichen
. b143  49 80     eor #$80      ... des Exponenten
. b145  8d d2 02  sta $02d2     Schreibt neues Vorzeichen des Exponenten
. b148  a9 30     lda #$30      Schreibt
. b14a  9d 01 01  sta $0101,x   ... Exponenten '01'
. b14d  a9 31     lda #$31      ... in
```

```
. b14f  9d Ø2 Ø1 sta $Ø1Ø2,x  ... Zahlenstring
. b152  6Ø       rts          ENDE

. b153  bd ØØ Ø1 lda $Ø1ØØ,x  Erhöht
. b156  fe ØØ Ø1 inc $Ø1ØØ,x  ... Exponenten-Ziffer
. b159  c9 39    cmp #$39     War Ziffer gleich 9 ?
. b15b  6Ø       rts          ENDE
```

HOLT NÄCHSTES ZEICHEN AUS FORMATSTRING

```
. b15c  18       clc
. b15d  c8       iny          Erhöht Zeiger im Formatstring
. b15e  fØ Ø5    beq $b165    Überlauf ?, dann $b165
. b16Ø  cc df Ø2 cpy $Ø2df    Ende des Formatstrings erreicht ?
. b163  9Ø Ø4    bcc $b169    Nein, dann $b169
. b165  a4 76    ldy $76      Flag für Ausgabe-Abbruch gesetzt ?
. b167  dØ d4    bne $b13d    Ja, dann Abbruch
. b169  2Ø 71 81 jsr $8171    Holt Zeichen aus Formatstring (LDA($3d),Y)
. b16c  ee db Ø2 inc $Ø2db    Erhöht Formatzähler um 1
. b16f  6Ø       rts          ENDE
```

PARAMETER INITIALISIEREN

```
. b17Ø  2Ø 4e 9c jsr $9c4e    (FRESTR)
. b173  85 77    sta $77      Schreibt Länge des Ausgabestrings
. b175  a2 Øa    ldx #$Øa
. b177  a9 ØØ    lda #$ØØ
. b179  9d d1 Ø2 sta $Ø2d1,x  Löscht
. b17c  ca       dex          ... Parameter-
. b17d  1Ø fa    bpl $b179    ... Bereich
. b17f  8e dØ Ø2 stx $Ø2dØ    Setzt Trennkomma-Flag
. b182  86 82    stx $82      Setzt Offset der 1. Nachkommastelle
. b184  8e cf Ø2 stx $Ø2cf    Setzt Dollar-Flag
. b187  aa       tax          X-Reg. := Ø
. b188  a8       tay          Y-Reg. := Ø
. b189  6Ø       rts          ENDE
```

ZAHL NACH FORMATANWEISUNGEN RUNDEN

```
. b18a  18       clc
. b18b  a5 82    lda $82      Addiert zum Offset der 1. Nachkommastelle
. b18d  6d d7 Ø2 adc $Ø2d7    ... Die Nachkommastellen im Formatstring
. b19Ø  bØ 39    bcs $b1cb    Überlauf ?, dann fertig
. b192  38       sec          Subtrahiert
. b193  e5 76    sbc $76      ... Zeichen vor der Zahl
. b195  9Ø 34    bcc $b1cb    Ergebnis negativ ?, dann fertig
. b197  cd ce Ø2 cmp $Ø2ce    Vergleich mit Offset des Zahlenstring-Endes
. b19a  fØ Ø2    beq $b19e    Gleich ?, dann $b19e
. b19c  bØ 2d    bcs $b1cb    Größer ?, dann fertig
. b19e  cd cd Ø2 cmp $Ø2cd    Vergleich mit Offset des Zahlenstring-Anfangs
```

```
. b1a1  90 28    bcc $b1cb    Kleiner ?, dann fertig
. b1a3  aa       tax          Offset der Rundungstelle nach Akku
. b1a4  bd 00 01 lda $0100,x  Holt Ziffer aus Zahlenstring
. b1a7  c9 35    cmp #$35     Ziffer kleiner 5 ?
. b1a9  90 20    bcc $b1cb    Ja, dann fertig
. b1ab  ec cd 02 cpx $02cd    Zeiger auf Beginn des Zahlenstrings ?
. b1ae  f0 0a    beq $b1ba    Ja, dann $b1ba
. b1b0  ca       dex          Erniedrigt Offset um 1
. b1b1  20 53 b1 jsr $b153    Ziffer um 1 erniedrigen
. b1b4  8e ce 02 stx $02ce    Setzt Offset gleich Ende der Zahl
. b1b7  f0 f2    beq $b1ab    Bei Übertrag, nach $b1ab
. b1b9  60       rts          ENDE

. b1ba  a9 31    lda #$31     Aufrunden
. b1bc  9d 00 01 sta $0100,x  ... auf 1
. b1bf  e8       inx          Erhöht Offset um 1
. b1c0  86 82    stx $82      Schreibt Offset der 1. Nachkommastelle
. b1c2  c6 76    dec $76      Abbruch-Flag erniedrigen
. b1c4  10 05    bpl $b1cb
. b1c6  e6 76    inc $76      Abbruch-Flag erhöhen
. b1c8  ee d4 02 inc $02d4    Vorkommastellen der Zahl erniedrigen
. b1cb  60       rts          ENDE
```

EXPONENT-DARSTELLUNG NORMALISIEREN

```
. b1cc  a4 82    ldy $82      Holt Offset der 1. Nachkommastelle
. b1ce  f0 17    beq $b1e7    Keine Vorkommastellen ?, dann $b1e7
. b1d0  ac cd 02 ldy $02cd    Holt Offset-Zeiger auf Beginn der Zahl
. b1d3  b9 00 01 lda $0100,y  Holt Ziffer aus Zahl
. b1d6  c9 30    cmp #$30     Ziffer gleich Null ?
. b1d8  60       rts          ENDE

. b1d9  e6 82    inc $82      Erhöht Offset der 1. Nachkommastelle
. b1db  20 0e b1 jsr $b10e    Erniedrigt Exponent
. b1de  ee cd 02 inc $02cd    Erhöht Offset auf Beginn der Zahl
. b1e1  cc ce 02 cpy $02ce    Ende der Zahl erreicht ?
. b1e4  f0 e5    beq $b1cb    Ja, dann fertig
. b1e6  c8       iny          Erhöht Offset-Zeiger in Zahl
. b1e7  20 d3 b1 jsr $b1d3    Ziffer aus Zahl holen und prüfen ob Null
. b1ea  f0 ed    beq $b1d9    Ja, dann $b1d9
. b1ec  60       rts          ENDE
```

FORMATIERTE ZAHL AUSGEBEN

```
. b1ed  ad cf 02 lda $02cf    Holt Dollar-Flag
. b1f0  30 02    bmi $b1f4    Schon Dollar-Zeichen ausgegeben ?, dann $b1f4
. b1f2  e6 76    inc $76
. b1f4  ae cd 02 ldx $02cd    Erniedrigt Offset
. b1f7  ca       dex          ... auf Anfang der Zahl
. b1f8  ac de 02 ldy $02de    Holt Zeiger auf Feldanfang
. b1fb  20 71 81 jsr $8171    Holt Zeichen aus Formatstring (LDA($3d),Y)
. b1fe  c8       iny          Erhöht Offset im Formatstring
```

```
. b1ff  c9 2c     cmp #$2c      Zeichen gleich Trennkomma (,) ?
. b201  d0 11     bne $b214     Nein, dann $b214
. b203  2c d0 02  bit $02d0     Schon Ziffer ausgebeben ?
. b206  30 06     bmi $b20e     Nein, dann Füllzeichen ausgeben
. b208  ad e8 04  lda $04e8     Holt Trennkomma oder PUDEF-Ersatz
. b20b  4c 76 b2  jmp $b276     Zeichen ausgeben und Formatzähler erniedrigen

. b20e  ad dd 02  lda $02dd     Holt Leerzeichen oder PUDEF-Ersatz
. b211  4c 76 b2  jmp $b276     Zeichen ausgeben und Formatzähler erniedrigen

. b214  c9 2e     cmp #$2e      Befindet sich Dezimalpunkt im Formatstring ?
. b216  d0 06     bne $b21e     Nein, dann $b21e
. b218  ad e9 04  lda $04e9     Holt Dezimalpunkt (.) oder PUDEF-Ersatz
. b21b  4c 76 b2  jmp $b276     Zeichen ausgeben und Formatzähler erniedrigen

. b21e  c9 2b     cmp #$2b      Befindet sich Pluszeichen (+) im Formatstring ?
. b220  f0 3b     beq $b25d     Ja, dann $b25d
. b222  c9 2d     cmp #$2d      Befindet sich Minuszeichen (-) im Formatstring?
. b224  f0 32     beq $b258     Ja, dann $b258
. b226  c9 5e     cmp #$5e      Befindet sich Exponent-Symbol im Formatstring ?
. b228  d0 63     bne $b28d     Nein, dann $b28d
. b22a  a9 45     lda #$45      Holt 'e'-Zeichen
. b22c  20 b0 b2  jsr $b2b0     Zeichen ausgeben und Formatzähler erniedrigen
. b22f  ac d3 02  ldy $02d3     Holt Offset-Zeiger auf Exponenten der Zahl
. b232  20 d3 b1  jsr $b1d3     Ist 1. Exponentenziffer gleich Null ?
. b235  d0 06     bne $b23d     Nein, dann $b23d
. b237  c8        iny           Offset auf 2. Ziffer
. b238  20 d3 b1  jsr $b1d3     Ist 2. Exponentenziffer gleich Null ?
. b23b  f0 07     beq $b244     Ja, dann $b244
. b23d  a9 2d     lda #$2d      Holt Minuszeichen (-)
. b23f  2c d2 02  bit $02d2     Exponent-Vorzeichen negativ ?
. b242  30 02     bmi $b246     Ja, dann $b246
. b244  a9 2b     lda #$2b      Holt Pluszeichen (+)
. b246  20 b0 b2  jsr $b2b0     Zeichen ausgeben und Formatzähler erniedrigen
. b249  ae d3 02  ldx $02d3     Holt Offset auf 1. Exponent-Ziffer
. b24c  bd 00 01  lda $0100,x   Holt 1. Exponent-Ziffer
. b24f  20 b0 b2  jsr $b2b0     Zeichen ausgeben und Formatzähler erniedrigen
. b252  ac e0 02  ldy $02e0     Holt Zeiger auf Ende des Feldes
. b255  4c 6c b2  jmp $b26c     Springt nach $b26c

. b258  ad dc 02  lda $02dc     Holt Vorzeichen-Flag im Zahlenstring
. b25b  30 b1     bmi $b20e     Ausgabe gesperrt ?, dann Füllzeichen ausgeben
. b25d  ad dc 02  lda $02dc     Holt Vorzeichen-Flag im Zahlenstring
. b260  4c 76 b2  jmp $b276     Zeichen ausgeben und Formatzähler erniedrigen

. b263  a5 76     lda $76       Holt Abbruch-Flag
. b265  d0 15     bne $b27c
. b267  ec ce 02  cpx $02ce     Ende der Zahl erreicht ?
. b26a  f0 05     beq $b271     Ja, dann Ersatzziffer '0' ausgeben
. b26c  e8        inx           Erhöht Offset in Zahl
. b26d  bd 00 01  lda $0100,x   Holt nächste Ziffer aus Zahl
```

```
. b27Ø  .byte $2c

. b271  a9 3Ø     lda #$3Ø      Holt Nullzeichen (Ø)
. b273  4e dØ Ø2  lsr $Ø2dØ     Setzt Trennkomma-Flag
. b276  2Ø bØ b2  jsr $b2bØ     Zeichen ausgeben und Formatzähler erniedrigen
. b279  dØ 8Ø     bne $b1fb     Keine Endmarke ?, dann $b1fb
. b27b  6Ø        rts           ENDE

. b27c  c6 76     dec $76
. b27e  ad cf Ø2  lda $Ø2cf     Dollarzeichen ($) schon ausgegeben ?
. b281  3Ø ee     bmi $b271     Ja, dann $b271
. b283  38        sec           Setzt
. b284  6e cf Ø2  ror $Ø2cf     ... Dollar-Flag
. b287  ad ea Ø4  lda $Ø4ea     Holt '$'-Zeichen oder PUDEF-Ersatz
. b28a  4c 73 b2  jmp $b273     Zeichen ausgeben und Formatzähler erniedrigen

. b28d  ad d1 Ø2  lda $Ø2d1     Differenz der Vorkommastellen gleich Null ?
. b29Ø  fØ d1     beq $b263     Ja, dann $b263
. b292  ce d1 Ø2  dec $Ø2d1     Erniedrigt Differenz um 1
. b295  fØ Ø3     beq $b29a     Differenz der Vorkommastellen jetzt gleich Ø ?
. b297  4c Øe b2  jmp $b2Øe     Füllzeichen ausgeben

. b29a  ad d8 Ø2  lda $Ø2d8     Befindet sich Vorzeichen im Formatstring ?
. b29d  3Ø f6     bmi $b295     Ja, dann Füllzeichen ausgeben
. b29f  2Ø 71 81  jsr $8171     Holt Zeichen aus Formatstring (LDA($3d),Y)
. b2a2  c9 2c     cmp #$2c      Zeichen gleich Trennkomma (,) ?
. b2a4  dØ b2     bne $b258     Nein, dann $b258
. b2a6  ad dd Ø2  lda $Ø2dd     Holt Leerzeichen oder PUDEF-Ersatz
. b2a9  2Ø bØ b2  jsr $b2bØ     Zeichen ausgeben und Formatzähler erniedrigen
. b2ac  c8        iny           Erhöht Offset im Formatstring
. b2ad  4c 9f b2  jmp $b29f     Springt nach $b29f

. b2bØ  2Ø b2 9Ø  jsr $9Øb2     Zeichen ausgeben
. b2b3  ce db Ø2  dec $Ø2db     Erniedrigt Formatzähler um 1
. b2b6  6Ø        rts           ENDE

     FORMATSTRING AUSWERTEN

. b2b7  ac eØ Ø2  ldy $Ø2eØ     Holt Zeiger auf Ende des Feldes
. b2ba  2Ø 5c b1  jsr $b15c     Holt nächstes Zeichen aus Formatstring
. b2bd  2Ø 6c b3  jsr $b36c     Zeichen aus +,-,.,=,# oder Exponent-Symbol ?
. b2cØ  dØ 14     bne $b2d6     Nein, dann $b2d6
. b2c2  8c de Ø2  sty $Ø2de     Schreibt Zeiger auf Anfang des Feldes
. b2c5  9Ø 1a     bcc $b2e1     Zeichen gleich '#'-Zeichen ?, dann $b2e1
. b2c7  aa        tax           Zeichen nach X-Reg.
. b2c8  2Ø 5c b1  jsr $b15c     Holt nächstes Zeichen aus Formatstring
. b2cb  bØ Ø5     bcs $b2d2     Formatstring zu Ende ?, dann $b2d2
. b2cd  2Ø 74 b3  jsr $b374     Besteht Zeichen aus .,=,# oder Exponent-Symbol?
. b2dØ  fØ Øa     beq $b2dc     Ja, dann $b2dc
. b2d2  ac de Ø2  ldy $Ø2de     Holt Zeiger auf Anfang des Feldes
. b2d5  8a        txa           Zeichen aus Formatstring nach Akku
. b2d6  2Ø b2 9Ø  jsr $9Øb2     Zeichen ausgeben
```

```
. b2d9  4c ba b2 jmp $b2ba    Zurück zum Anfang der Schleife

. b2dc  bØ ea    bcs $b2c8    Zeichen ungleich '#'-Zeichen ?, dann $b2c8
. b2de  ac de Ø2 ldy $Ø2de    Holt Zeiger auf Anfang des Feldes
. b2e1  a6 76    ldx $76      Abbruch-Flag gesetzt ?
. b2e3  dØ 7a    bne $b35f    Ja, dann $b35f
. b2e5  8e db Ø2 stx $Ø2db    Setzt Format-Zähler gleich Null
. b2e8  88       dey
. b2e9  ce db Ø2 dec $Ø2db    Erniedrigt Format-Zähler
. b2ec  2Ø 5c b1 jsr $b15c    Holt nächstes Zeichen aus Formatstring
. b2ef  bØ 74    bcs $b365    Ende des Formatstrings ?, dann $b365
. b2f1  c9 2c    cmp #$2c     Zeichen gleich Trennkomma (,) ?
. b2f3  fØ f7    beq $b2ec    Ja, dann $b2ec
. b2f5  2Ø 43 b3 jsr $b343    Wenn Zeichen gleich '+','-', dann übernehmen
. b2f8  9Ø ef    bcc $b2e9    ... und nach $b2e9 springen
. b2fa  c9 2e    cmp #$2e     Zeichen gleich Dezimalpunkt ?
. b2fc  dØ Ø8    bne $b3Ø6    Nein, dann $b3Ø6
. b2fe  e8       inx          Dezimalpunkt-Zähler erhöhen
. b2ff  eØ Ø2    cpx #$Ø2     Zweiter Dezimalpunkt ?
. b3Ø1  9Ø e9    bcc $b2ec    Nein, dann $b2ec
. b3Ø3  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. b3Ø6  2Ø 78 b3 jsr $b378    Zeichen aus =,# oder Exponent-Zeichen ?
. b3Ø9  dØ Øb    bne $b316    Nein, dann $b316
. b3Øb  9Ø Ø3    bcc $b31Ø    Zeichen gleich '#'-Zeichen ?
. b3Ød  8d d5 Ø2 sta $Ø2d5    Zeichen zwischenspeichern
. b31Ø  fe d6 Ø2 inc $Ø2d6,x  Zählt Vor- bzw. Nachkommastellen
. b313  4c ec b2 jmp $b2ec    Nächstes Zeichen

. b316  c9 24    cmp #$24     Zeichen gleich Dollar-Zeichen ($) ?
. b318  dØ Øf    bne $b329    Nein, dann $b329
. b31a  2c cf Ø2 bit $Ø2cf    Schon ein '$'-Zeichen gefunden ?
. b31d  1Ø f1    bpl $b31Ø    Ja, dann $b31Ø
. b31f  18       clc          Setzt
. b32Ø  6e cf Ø2 ror $Ø2cf    ... Dollar-Flag
. b323  ce d6 Ø2 dec $Ø2d6    Erniedrigt Vorkommastellen
. b326  4c 1Ø b3 jmp $b31Ø    Springt nach $b31Ø

. b329  c9 5e    cmp #$5e     Zeichen gleich Exponent-Symbol ?
. b32b  dØ 16    bne $b343    Nein, dann $b343
. b32d  a2 Ø2    ldx #$Ø2
. b32f  2Ø 5c b1 jsr $b15c    Holt nächstes Zeichen aus Formatstring ?
. b332  bØ cf    bcs $b3Ø3    Endmarke ?, dann Fehler
. b334  c9 5e    cmp #$5e     Insgesamt 4 Exponent-Symbole
. b336  dØ cb    bne $b3Ø3    Nein, dann Fehler
. b338  ca       dex
. b339  1Ø f4    bpl $b32f
. b33b  ee d9 Ø2 inc $Ø2d9    Setzt Exponent-Symbol
. b33e  2Ø 5c b1 jsr $b15c    Holt nächstes Zeichen aus Formatstring
. b341  bØ 22    bcs $b365    Endmarke ?, dann $b365
. b343  c9 2b    cmp #$2b     Zeichen gleich Pluszeichen (+) ?
. b345  dØ 19    bne $b36Ø    Nein, dann $b36Ø
```

```
. b347  ad dc Ø2 lda $Ø2dc    Befindet sich Vorzeichen im Zahlenstring ?
. b34a  1Ø Ø5    bpl $b351    Ja, dann $b351
. b34c  a9 2b    lda #$2b     Holt Pluszeichen (+) ?
. b34e  8d dc Ø2 sta $Ø2dc    Ersetzt Vorzeichen-Flag durch '+'-Zeichen
. b351  ad d8 Ø2 lda $Ø2d8    Schon Vorzeichen im Formatstring ?
. b354  dØ ad    bne $b3Ø3    Ja, dann Fehler
. b356  6e d8 Ø2 ror $Ø2d8    Setzt Vorzeichen-Flag für Formatstring
. b359  8c eØ Ø2 sty $Ø2eØ    Schreibt Zeiger auf Feldende
. b35c  ee db Ø2 inc $Ø2db    Erhöht Formatzähler
. b35f  6Ø       rts          ENDE

. b36Ø  c9 2d    cmp #$2d     Zeichen gleich Minuszeichen (-) ?
. b362  fØ ed    beq $b351    Ja, dann $b351
. b364  38       sec
. b365  8c eØ Ø2 sty $Ø2eØ    Feldende um 1
. b368  ce eØ Ø2 dec $Ø2eØ    ... vermindern
. b36b  6Ø       rts          ENDE

     FORMATZEICHEN TESTEN

. b36c  c9 2b    cmp #$2b     Zeichen gleich Pluszeichen (+) ?
. b36e  fØ 15    beq $b385    Ja, dann $b385
. b37Ø  c9 2d    cmp #$2d     Zeichen gleich Minuszeichen (-) ?
. b372  fØ 11    beq $b385    Ja, dann $b385
. b374  c9 2e    cmp #$2e     Zeichen gleich Dezimalpunkt (.) ?
. b376  fØ Ød    beq $b385    Ja, dann $b385
. b378  c9 3d    cmp #$3d     Zeichen gleich Zentriersymbol (=) ?
. b37a  fØ Ø9    beq $b385    Ja, dann $b385
. b37c  c9 3e    cmp #$3e     Zeichen gleich Exponent-Symbol ?
. b37e  fØ Ø5    beq $b385    Ja, dann $b385
. b38Ø  c9 23    cmp #$23     Zeichen gleich Ziffernsymbol (#) ?
. b382  dØ Ø1    bne $b385    Ja, dann $b385
. b384  18       clc
. b385  6Ø       rts          ENDE

. b386  :
. b386  *** I N S T R ***
. b386  :
. b386  a5 64    lda $64      Umspeichern
. b388  8d eb Ø4 sta $Ø4eb    ... des 1.
. b38b  a5 65    lda $65      ... Deskriptor-
. b38d  8d ec Ø4 sta $Ø4ec    ... Zeigers
. b39Ø  2Ø 2c 93 jsr $932c    (FRMEVL) Ausdruck auswerten
. b393  2Ø 1a 93 jsr $931a    (CHKSTR) Prüft ob String
. b396  a5 64    lda $64      Umspeichern
. b398  8d ed Ø4 sta $Ø4ed    ... des 2.
. b39b  a5 65    lda $65      ... Deskriptor-
. b39d  8d ee Ø4 sta $Ø4ee    ... Zeigers
. b3aØ  a2 Ø1    ldx #$Ø1     Suchzeiger
. b3a2  86 65    stx $65      ... auf 1. Zeichen (Anfang)
. b3a4  2Ø 79 Ø4 jsr $Ø479    CHRGOT-Routine
. b3a7  c9 29    cmp #$29     Zeichen gleich Klammer zu ')' ?
```

```
. b3a9  f0 03    beq $b3ae    Keine Parameter ?, dann $b3ae
. b3ab  20 d8 9d jsr $9dd8    Übernimmt Parameter
. b3ae  20 8b 94 jsr $948b    Prüft ob ")" folgt
. b3b1  a6 65    ldx $65      Parameterwert ungleich 0 ?
. b3b3  d0 03    bne $b3b8    Ja, dann $b3b8
. b3b5  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. b3b8  ca       dex          Zeiger in Suchstring um 1 erniedrigen
. b3b9  86 61    stx $61      Schreibt Zeiger in Suchstring
. b3bb  a2 03    ldx #$03
. b3bd  bd eb 04 lda $04eb,x  Umspeichern
. b3c0  95 57    sta $57,x    ... des
. b3c2  ca       dex          ... Deskriptor-
. b3c3  10 f8    bpl $b3bd    ... Zeigers...
. b3c5  a0 02    ldy #$02
. b3c7  20 75 81 jsr $8175    Holt 1. Deskriptor (LDA($57),Y)
. b3ca  99 5b 00 sta $005b,y
. b3cd  20 79 81 jsr $8179    Holt 2. Deskriptor (LDA($59),Y)
. b3d0  99 5e 00 sta $005e,y
. b3d3  88       dey
. b3d4  10 f1    bpl $b3c7
. b3d6  a5 5e    lda $5e      Länge des gesuchten Strings gleich 0 ?
. b3d8  f0 37    beq $b411    Ja, dann fertig
. b3da  a9 00    lda #$00
. b3dc  85 62    sta $62      Schreibt Zeiger in gesuchten String
. b3de  18       clc          Addiert Zeiger
. b3df  a5 5e    lda $5e      ... in Suchstring und
. b3e1  65 61    adc $61      ... Länge des gesuchten Strings
. b3e3  b0 2c    bcs $b411    Ergebnis größer 255 ?, dann fertig
. b3e5  c5 5b    cmp $5b      Ergebnis größer Suchstring ?
. b3e7  90 02    bcc $b3eb    Nein, dann $b3eb
. b3e9  d0 26    bne $b411    Ja, dann fertig
. b3eb  a4 62    ldy $62      Ist Zeiger in gesuchtem String
. b3ed  c4 5e    cpy $5e      ... gleich Länge des gesuchten Strings ?
. b3ef  f0 1b    beq $b40c    Ja, dann $b40c
. b3f1  98       tya          Addiert Zeiger in
. b3f2  18       clc          ... gesuchten String mit
. b3f3  65 61    adc $61      ... Zeiger in Suchstring
. b3f5  a8       tay
. b3f6  20 69 81 jsr $8169    Holt Zeichen aus Suchstring (LDA($5c),Y)
. b3f9  85 78    sta $78      Zwischenspeichern
. b3fb  a4 62    ldy $62      Holt Zeiger in gesuchten String
. b3fd  20 6d 81 jsr $816d    Holt Zeichen aus gesuchtem String (LDA($5f),Y)
. b400  c5 78    cmp $78      Beide Zeichen gleich ?
. b402  f0 04    beq $b408    Ja, dann $b408
. b404  e6 61    inc $61      Erhöht Zeiger in Suchstring
. b406  d0 d2    bne $b3da    Springt immer nach $b3da

. b408  e6 62    inc $62      Erhöht Zeiger in gesuchten String
. b40a  d0 df    bne $b3eb    Springt immer nach $b3eb

. b40c  e6 61    inc $61      Erhöht Zeiger in Suchstring
```

```
. b40e  a5 61    lda $61      Holt Zeiger in Suchstring

. b410  .byte $2c

. b411  a9 00    lda #$00
. b413  48       pha
. b414  ad ed 04 lda $04ed    Holt 2.
. b417  ac ee 04 ldy $04ee    ... Deskriptor-Zeiger
. b41a  20 52 9c jsr $9c52    (FRESTR)
. b41d  ad eb 04 lda $04eb    Holt 1.
. b420  ac ec 04 ldy $04ec    ... Deskriptor-Zeiger
. b423  20 52 9c jsr $9c52    (FRESTR)
. b426  68       pla
. b427  a8       tay          Bringt Zeiger in Suchstring
. b428  4c 81 9a jmp $9a81    ... nach FAC

. b42b  :
. b42b  *** T R A P ***
. b42b  :
. b42b  20 86 9a jsr $9a86    Prüft auf Direktmodus
. b42e  20 79 04 jsr $0479    CHRGOT-Routine
. b431  f0 07    beq $b43a    Folgt Trennzeichen ?, dann abschalten
. b433  20 e1 9d jsr $9de1    (GETADR) Adresse holen
. b436  8c f2 04 sty $04f2    Schreibt ON-ERROR-Zeilennummer (Low)

. b439  .byte $2c

. b43a  a9 ff    lda #$ff
. b43c  8d f3 04 sta $04f3    Schreibt ON-ERROR-Zeilennummer (High)
. b43f  60       rts          ENDE

. b440  :
. b440  *** R E S U M E ***
. b440  :
. b440  20 86 9a jsr $9a86    Prüft auf Direktmodus
. b443  ae f1 04 ldx $04f1    ON-ERROR-Flag
. b446  e8       inx          ... gesetzt ?
. b447  f0 70    beq $b4b9    Nein, dann $b4b9
. b449  20 79 04 jsr $0479    CHRGOT-Routine
. b44c  f0 47    beq $b495    Folgt Trennzeichen ?, dann $b495
. b44e  90 3a    bcc $b48a    Folgt Ziffer ?, dann $b48a
. b450  c9 82    cmp #$82     Folgt 'NEXT'-Token ?
. b452  d0 62    bne $b4b6    Nein, dann $b4b6
. b454  20 95 b4 jsr $b495    Holt Fehleradresse in Textpointer
. b457  a0 00    ldy #$00
. b459  20 a5 04 jsr $04a5    Holt Zeichen aus Zeile (LDA($3b),Y)
. b45c  d0 26    bne $b484    Keine Endmarke ?, dann $b484
. b45e  c8       iny
. b45f  20 a5 04 jsr $04a5    Holt Zeichen (LDA($3b),Y)
. b462  d0 09    bne $b46d    Noch nicht Programmende ?, dann $b46d
. b464  c8       iny
. b465  20 a5 04 jsr $04a5    Holt Zeichen (LDA($3b),Y)
```

```
. b468  dØ Ø3    bne $b46d    Noch nicht Programmende ?, dann $b46d
. b46a  4c 7e 86 jmp $867e    READY.

. b46d  aØ Ø3    ldy #$Ø3     Offset auf Zeilennummer setzen
. b46f  2Ø a5 Ø4 jsr $Ø4a5    Holt Zeilennummer (Low) (LDA($3b),Y)
. b472  85 39    sta $39      Zeilennummer zwischenspeichern
. b474  c8       iny          Offset um 1 erhöhen
. b475  2Ø a5 Ø4 jsr $Ø4a5    Holt Zeilennummer (High) (LDA($3b),Y)
. b478  85 3a    sta $3a      Zeilennummer zwischenspeichern
. b47a  98       tya          Offset nach Akku
. b47b  18       clc          Addiert
. b47c  65 3b    adc $3b      ... Offset
. b47e  85 3b    sta $3b      ... zum
. b48Ø  9Ø Ø2    bcc $b484    ... Textpointer
. b482  e6 3c    inc $3c
. b484  2Ø 73 Ø4 jsr $Ø473    CHRGET-Routine
. b487  4c bØ 8d jmp $8dbØ    DATA

. b48a  2Ø e1 9d jsr $9de1    (GETADR) Adresse holen
. b48d  85 15    sta $15      ???
. b48f  2Ø a4 b4 jsr $b4a4    Fehlerstatus zurücksetzen
. b492  4c 69 8d jmp $8d69    Sucht Zeile

. b495  a2 Ø1    ldx #$Ø1     Stellt
. b497  bd fØ Ø4 lda $Ø4fØ,x  ... Zeilennummer
. b49a  95 39    sta $39,x    ... und
. b49c  bd f5 Ø4 lda $Ø4f5,x  ... Programm-
. b49f  95 3b    sta $3b,x    ... Adresse
. b4a1  ca       dex          ... wieder
. b4a2  1Ø f3    bpl $b497    ... her
. b4a4  a2 ff    ldx #$ff     Setzt
. b4a6  8e ef Ø4 stx $Ø4ef    ... Fehlercode,
. b4a9  8e fØ Ø4 stx $Ø4fØ    ... Fehlerzeilennummer (Low),
. b4ac  8e f1 Ø4 stx $Ø4f1    ... und Fehlerzeilennummer mit $ff belegen
. b4af  ae f4 Ø4 ldx $Ø4f4    ON-ERROR-Zeilennummer (High)
. b4b2  8e f3 Ø4 stx $Ø4f3    ... wiederherstellen
. b4b5  6Ø       rts          ENDE

. b4b6  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. b4b9  a2 1f    ldx #$1f     Holt Fehlercode 31 ('CAN'T RESUME')
. b4bb  4c 83 86 jmp $8683    Fehlerausgabe

. b4be  :
. b4be  *** E R R $ ***
. b4be  :
. b4be  2Ø 87 9d jsr $9d87    (GETBYT) Übernimmt Fehlercode
. b4c1  ca       dex
. b4c2  8a       txa
. b4c3  c9 24    cmp #$24     Fehlercode größer 36 ?
. b4c5  bØ 34    bcs $b4fb    Ja, dann Fehler
. b4c7  2Ø 53 86 jsr $8653    Setzt Zeiger auf Fehlermeldung (X-Reg.)
```

```
. b4ca  a0 ff     ldy #$ff      Zeiger in Fehlermeldung initialisieren
. b4cc  a2 00     ldx #$00      Zähler für Länge der Meldung initialisieren
. b4ce  e8        inx           Zähler um 1 erhöhen
. b4cf  c8        iny           Zeiger um 1 erhöhen
. b4d0  b1 24     lda ($24),y   Holt Zeichen aus Fehlermeldung
. b4d2  30 06     bmi $b4da
. b4d4  c9 20     cmp #$20      Zeichen gleich Steuercode ?
. b4d6  90 f7     bcc $b4cf     Ja, dann Zeichen nicht mitzählen
. b4d8  b0 f4     bcs $b4ce     Springt nach $b4ce

. b4da  8a        txa           Länge der Meldung (ohne Steuercodes)
. b4db  20 5c 9b  jsr $9b5c     Reserviert Platz für Fehlermeldung
. b4de  a2 00     ldx #$00      Zähler für Länge der Meldung initialisieren
. b4e0  a0 ff     ldy #$ff      Zeiger in Fehlermeldung initialisieren
. b4e2  c8        iny           Zeiger um 1 erhöhen
. b4e3  b1 24     lda ($24),y   Holt Zeichen aus Fehlermeldung
. b4e5  c9 20     cmp #$20      Zeichen gleich Steuercode ?
. b4e7  90 f9     bcc $b4e2     Ja, dann Steuercode übernehmen
. b4e9  20 fe b4  jsr $b4fe     Vertauscht X- und Y-Register
. b4ec  48        pha
. b4ed  29 7f     and #$7f
. b4ef  91 62     sta ($62),y   Kopiert Zeichen in Stringbereich
. b4f1  20 fe b4  jsr $b4fe     Vertauscht X- und Y-Register
. b4f4  e8        inx
. b4f5  68        pla
. b4f6  10 ea     bpl $b4e2     Noch nicht letztes Zeichen, dann weiterkopieren
. b4f8  4c ca 9c  jmp $9cca     String in String-Stack übernehmen

. b4fb  4c 1c 99  jmp $991c     'ILLEGAL QUANTITY ERROR' ausgeben

. b4fe  48        pha           Vertauschen
. b4ff  8a        txa           ... von
. b500  48        pha           ... x-
. b501  98        tya           ... und
. b502  aa        tax           ... Y-Register...
. b503  68        pla
. b504  a8        tay
. b505  68        pla
. b506  60        rts           ENDE

. b507  :
. b507  *** H E X $ ***
. b507  :
. b507  20 17 93  jsr $9317     (CHKNUM) Prüft ob numerisch
. b50a  a5 14     lda $14       Adresse
. b50c  48        pha           ... auf
. b50d  a5 15     lda $15       ... Stack
. b50f  48        pha           ... retten
. b510  20 e4 9d  jsr $9de4     (GETADR) Adresse holen
. b513  a9 04     lda #$04      Reserviert 4 Byte (für HEX-Adresse)
. b515  20 5c 9b  jsr $9b5c     Reserviert Platz (4 Byte) für String
. b518  a0 00     ldy #$00
```

```
. b51a  a5 15    lda $15      Holt High-Byte der Adresse
. b51c  20 2d b5 jsr $b52d    Als 2 HEX-Ziffern in String schreiben
. b51f  a5 14    lda $14      Holt Low-Byte der Adresse
. b521  20 2d b5 jsr $b52d    Als 2 HEX-Ziffern in String schreiben
. b524  68       pla          Alte Adresse
. b525  85 15    sta $15      ... wieder
. b527  68       pla          ... vom Stack
. b528  85 14    sta $14      ... holen
. b52a  4c ca 9c jmp $9cca    String in String-Stack übernehmen

. b52d  48       pha          Byte auf Stack legen
. b52e  4a       lsr          Oberes Halbbyte
. b52f  4a       lsr          ... nach
. b530  4a       lsr          ... unten
. b531  4a       lsr          ... verschieben
. b532  20 36 b5 jsr $b536    In ASCII wandeln und in String schreiben
. b535  68       pla          Byte wiederherstellen
. b536  29 0f    and #$0f     Unteres Halbbyte isolieren
. b538  c9 0a    cmp #$0a     Unteres Halbbyte kleiner 10 ?
. b53a  90 02    bcc $b53e    Ja, dann $b53e
. b53c  69 06    adc #$06     In HEX-Ziffern A-F verwandeln
. b53e  69 30    adc #$30     In ASCII-Code wandeln
. b540  91 62    sta ($62),y  HEX-Ziffer in Stringbereich speichern
. b542  c8       iny
. b543  60       rts          ENDE

. b544  :
. b544  *** P U D E F ***
. b544  :
. b544  20 48 9c jsr $9c48    (FRMEVL), (CHKSTR), (FRESTR)
. b547  a8       tay          Länge nach Y-Reg.
. b548  88       dey          Um 1 erniedrigen
. b549  c0 04    cpy #$04     Länge größer als 4 ?
. b54b  b0 ae    bcs $b4fb    Ja, dann Fehler
. b54d  20 b0 04 jsr $04b0    Holt Zeichen aus String (LDA($22),Y)
. b550  99 e7 04 sta $04e7,y  Zeichen in PUDEF-Bereich schreiben
. b553  88       dey          Zähler noch nicht Null ?
. b554  10 f7    bpl $b54d    Ja, dann weiterkopieren
. b556  60       rts          ENDE

. b557  :
. b557  *** D O ***
. b557  :
. b557  a0 01    ldy #$01     Textpointer
. b559  b9 3b 00 lda $003b,y  ... und
. b55c  99 f8 04 sta $04f8,y  ... Zeilennummer
. b55f  b9 39 00 lda $0039,y  ... umspeichern...
. b562  99 fa 04 sta $04fa,y
. b565  88       dey
. b566  10 f1    bpl $b559
. b568  20 79 04 jsr $0479    CHRGOT-Routine
. b56b  f0 1c    beq $b589    Folgt Trennzeichen ?, dann $b589
```

```
. b56d  c9 fc     cmp #$fc      Zeichen gleich 'UNTIL'-Token ?
. b56f  f0 11     beq $b582     Ja, dann $b582
. b571  c9 fd     cmp #$fd      Zeichen gleich 'WHILE'-Token ?
. b573  d0 3f     bne $b5b4     Nein, dann Fehler
. b575  20 4c b6  jsr $b64c     CHRGET-Routine & (FRMEVL) Ausdruck auswerten
. b578  a5 61     lda $61       Bedingung erfüllt ?
. b57a  d0 0d     bne $b589     Ja, dann $b589
. b57c  20 79 04  jsr $0479     CHRGOT-Routine
. b57f  4c ba b5  jmp $b5ba     LOOP verlassen

. b582  20 4c b6  jsr $b64c     CHRGET-Routine & (FRMEVL) Ausdruck auswerten
. b585  a5 61     lda $61       Bedingung erfüllt ?
. b587  d0 f3     bne $b57c     Ja, dann $b57c
. b589  a0 05     ldy #$05      Ist im Basic-Stack
. b58b  20 05 89  jsr $8905     ... Platz für 5 Byte ?
. b58e  88        dey
. b58f  ad f9 04  lda $04f9     Holt Programmzeiger (High)
. b592  91 7c     sta ($7c),y   Auf Basic-Stack legen
. b594  88        dey
. b595  ad f8 04  lda $04f8     Holt Programmzeiger (Low)
. b598  91 7c     sta ($7c),y   Auf Basic-Stack legen
. b59a  88        dey
. b59b  ad fb 04  lda $04fb     Holt Zeilennummer (High)
. b59e  91 7c     sta ($7c),y   Auf Basic-Stack legen
. b5a0  88        dey
. b5a1  ad fa 04  lda $04fa     Holt Zeilennummer (Low)
. b5a4  91 7c     sta ($7c),y   Auf Basic-Stack legen
. b5a6  88        dey
. b5a7  a9 eb     lda #$eb      Holt 'DO'-Token
. b5a9  91 7c     sta ($7c),y   Auf Basic-Stack legen
. b5ab  60        rts           ENDE

. b5ac  :
. b5ac  *** E X I T ***
. b5ac  :
. b5ac  20 14 b6  jsr $b614     Sucht 'DO'-Token im Basic-Stack
. b5af  20 79 04  jsr $0479     CHRGOT-Routine
. b5b2  f0 06     beq $b5ba     Folgt Trennzeichen ?, dann $b5ba
. b5b4  4c a1 94  jmp $94a1     'SYNTAX ERROR' ausgeben

. b5b7  20 73 04  jsr $0473     CHRGET-Routine
. b5ba  f0 1d     beq $b5d9     Folgt Trennzeichen ?, dann $b5d9
. b5bc  c9 ec     cmp #$ec      Zeichen gleich 'LOOP'-Token ?
. b5be  f0 40     beq $b600     Ja, dann $b600
. b5c0  c9 22     cmp #$22      Zeichen gleich Anführungszeichen (für Text) ?
. b5c2  f0 0a     beq $b5ce     Ja, dann $b5ce
. b5c4  c9 eb     cmp #$eb      Zeichen gleich 'DO'-Token
. b5c6  d0 ef     bne $b5b7     Nein, dann $b5b7
. b5c8  20 b7 b5  jsr $b5b7     Rekursion ($b5b7)
. b5cb  4c 7c b5  jmp $b57c     CHRGET-Routine & LOOP verlassen

. b5ce  20 73 04  jsr $0473     CHRGET-Routine
```

```
. b5d1  f0 06     beq $b5d9     Folgt Trennzeichen ?, dann $b5d9
. b5d3  c9 22     cmp #$22      Zeichen gleich Anführungszeichen (für Textende)?
. b5d5  d0 f7     bne $b5ce     Nein, dann weiter überlesen
. b5d7  f0 de     beq $b5b7     Ja, dann $b5b7

. b5d9  c9 3a     cmp #$3a      Zeichen gleich Doppelpunkt (kein Zeilenende) ?
. b5db  f0 da     beq $b5b7     Ja, dann $b5b7
. b5dd  24 81     bit $81       Direktmodus ?
. b5df  10 44     bpl $b625     Ja, dann Fehler
. b5e1  a0 02     ldy #$02
. b5e3  20 a5 04  jsr $04a5     Holt Linkadresse (High) (LDA($3b),Y)
. b5e6  f0 3d     beq $b625     Programmende ?, dann Fehler
. b5e8  c8        iny
. b5e9  20 a5 04  jsr $04a5     Holt Zeilennummer (Low) (LDA($3b),Y)
. b5ec  85 39     sta $39       Als laufende Basic-Zeilennummer zurückschreiben
. b5ee  c8        iny
. b5ef  20 a5 04  jsr $04a5     Holt Zeilennummer (High) (LDA($3b),Y)
. b5f2  85 3a     sta $3a       Als laufende Basic-Zeilennummer zurückschreiben
. b5f4  98        tya
. b5f5  18        clc           Setzt
. b5f6  65 3b     adc $3b       ... Textpointer
. b5f8  85 3b     sta $3b       ... auf das
. b5fa  90 bb     bcc $b5b7     ... Ende des
. b5fc  e6 3c     inc $3c       ... Zeilenkopfes
. b5fe  d0 b7     bne $b5b7     Springt immer nach $b5b7

. b600  4c b0 8d  jmp $8db0     DATA

. b603  :
. b603  *** L O O P ***
. b603  :
. b603  f0 2d     beq $b632     Folgt Trennzeichen ?, dann $b632
. b605  c9 fd     cmp #$fd      Folgt 'WHILE'-Token ?
. b607  f0 24     beq $b62d     Ja, dann $b62d
. b609  c9 fc     cmp #$fc      Folgt 'UNTIL'-Token ?
. b60b  d0 a7     bne $b5b4     Nein, dann $b5b4
. b60d  20 4c b6  jsr $b64c     CHRGET-Routine und (FRMEVL) Ausdruck auswerten
. b610  a5 61     lda $61       Bedingung erfüllt ?
. b612  f0 1e     beq $b632     Nein, dann $b632
. b614  a9 eb     lda #$eb      Holt 'DO'-Token
. b616  85 02     sta $02       Zur Suche im Stack speichern
. b618  20 71 88  jsr $8871     Sucht 'DO'-Token im Basic-Stack
. b61b  d0 0b     bne $b628     Kein 'DO' gefunden ?, dann Fehler
. b61d  20 69 a7  jsr $a769     Stackzeiger := Hilfszeiger
. b620  a0 05     ldy #$05      Holt Anzahl der Stack-Bytes bei 'DO'
. b622  4c 72 a7  jmp $a772     Stackzeiger um 5 Byte erhöhen

. b625  a2 20     ldx #$20      Holt Fehlercode 32 ('LOOP NOT FOUND')

. b627  .byte $2c

. b628  a2 21     ldx #$21      Holt Fehlercode 33 ('LOOP WITHOUT DO')
```

```
. b62a  4c 83 86 jmp $8683    Fehlerausgabe

. b62d  20 4c b6 jsr $b64c    CHRGET-Routinen und (FRMEVL) Ausdruck auswerten
. b630  f0 e2    beq $b614    Bedingung nicht erfüllt ?, dann $b614
. b632  20 14 b6 jsr $b614    Sucht 'DO' in Basic-Stack
. b635  88       dey
. b636  b1 3d    lda ($3d),y  Übernimmt Textpointer (High) aus Basic-Stack
. b638  85 3c    sta $3c      Schreibt Textpointer zurück
. b63a  88       dey
. b63b  b1 3d    lda ($3d),y  Übernimmt Textpointer (Low) aus Basic-Stack
. b63d  85 3b    sta $3b      Schreibt Textpointer zurück
. b63f  88       dey
. b640  b1 3d    lda ($3d),y  Übernimmt Zeilennummer (High) aus Textpointer
. b642  20 7f cd jsr $cd7f    Schreibt Zeilennummer oder setzt Direktmodus
. b645  b1 3d    lda ($3d),y  Übernimmt Zeilennummer (Low) aus Textpointer
. b647  85 39    sta $39      Schreibt laufende Basic-Zeilennummer zurück
. b649  4c 57 b5 jmp $b557    DO

. b64c  20 73 00 jsr $0073    CHRGET-Routine
. b64f  4c 2c 93 jmp $932c    (FRMEVL) Ausdruck auswerten

. b652  :
. b652  *** T R O N ***
. b652  :
. b652  a9 ff    lda #$ff     Flag setzen

. b654  .byte $2c

. b655  :
. b655  *** T R O F F ***
. b655  :
. b655  a9 00    lda #$00     Flag löschen
. b657  8d eb 02 sta $02eb    Schreibt TRACE-Flag
. b65a  60       rts          ENDE

. b65b  :
. b65b  *** MID$(...) = A$ ***
. b65b  :
. b65b  20 8e 94 jsr $948e    Prüft ob Klammer '(' folgt
. b65e  20 a5 96 jsr $96a5    Variable suchen oder anlegen
. b661  85 49    sta $49      Schreibt
. b663  84 4a    sty $4a      ... Variablenzeiger
. b665  20 1a 93 jsr $931a    (CHKSTR) Prüft auf String
. b668  20 d8 9d jsr $9dd8    (CHKCOM), (GETBYT)
. b66b  ca       dex
. b66c  86 77    stx $77      Schreibt 1. Parameter
. b66e  c9 29    cmp #$29     Gleich Klammer ')' ?
. b670  f0 04    beq $b676    Ja, dann kein 2. Parameter
. b672  20 d8 9d jsr $9dd8    (CHKCOM), (GETBYT)

. b675  .byte $2c
```

```
. b676  a2 ff     ldx #$ff      Setzt 2. Parameter
. b678  86 78     stx $78       ... gleich 255
. b67a  20 8b 94  jsr $948b     Prüft ob Klammer ')' folgt
. b67d  a9 b2     lda #$b2      Holt '='-Token
. b67f  20 93 94  jsr $9493     Prüft ob '='-Zeichen folgt
. b682  20 2c 93  jsr $932c     (FRMEVL) Ausdruck auswerten
. b685  20 1a 93  jsr $931a     (CHKSTR) Prüft auf String
. b688  a0 02     ldy #$02
. b68a  a9 49     lda #$49
. b68c  20 94 04  jsr $0494     Holt Deskriptor des Zielstrings (LDA($49),Y)
. b68f  99 5b 00  sta $005b,y   ... und schreibt diesen nach $5b, $5c, 5d
. b692  20 dc 04  jsr $04dc     Holt Deskriptor des Einbaustrings (LDA($64),Y)
. b695  99 5e 00  sta $005e,y   ... und schreibt diesen nach $5e, $5f, 60...
. b698  88        dey
. b699  10 ef     bpl $b68a
. b69b  38        sec           Subtrahiert
. b69c  a5 5f     lda $5f       ... 1. Parameter
. b69e  e5 77     sbc $77       ... von
. b6a0  85 5f     sta $5f       ... Low-Byte
. b6a2  b0 02     bcs $b6a6     ... der Adresse
. b6a4  c6 60     dec $60       ... des Einbaustrings
. b6a6  a5 78     lda $78       Holt 2. Parameter
. b6a8  c5 5e     cmp $5e       Kleiner Länge des Einbaustrings
. b6aa  90 02     bcc $b6ae     Ja, dann $b6ae
. b6ac  a5 5e     lda $5e       Sonst durch Stringlänge ersetzen
. b6ae  aa        tax
. b6af  f0 16     beq $b6c7     Ergebnis gleich Null ?, dann fertig
. b6b1  18        clc           Addiert
. b6b2  65 77     adc $77       ... 1. Parameter
. b6b4  b0 14     bcs $b6ca     Ergebnis größer 255 ?, dann Fehler
. b6b6  c5 5b     cmp $5b       Ergebnis größer Länge des Zielstrings ?
. b6b8  90 02     bcc $b6bc
. b6ba  d0 0e     bne $b6ca     Ja, dann Fehler
. b6bc  a4 77     ldy $77       Holt 1. Parameter
. b6be  20 6d 81  jsr $816d     Holt Zeichen aus Einbaustring (LDA($5f),Y)
. b6c1  91 5c     sta ($5c),y   Zeichen in Zielstring schreiben
. b6c3  c8        iny
. b6c4  ca        dex           Einbaulänge um 1 erniedrigen
. b6c5  d0 f7     bne $b6be     Noch nicht Null ?, dann weiterkopieren
. b6c7  4c 4e 9c  jmp $9c4e     (FRESTR)

. b6ca  4c 1c 99  jmp $991c     'ILLEGAL QUANTITY ERROR' ausgeben

. b6cd  :
. b6cd  *** A U T O ***
. b6cd  :
. b6cd  20 de b6  jsr $b6de     Prüft auf Direktmodus
. b6d0  20 3e 8e  jsr $8e3e     Übernimmt Zeilennummer
. b6d3  a5 14     lda $14       ... und speichert
. b6d5  85 73     sta $73       ... diese
. b6d7  a5 15     lda $15       ... als
. b6d9  85 74     sta $74       ... AUTO-Schrittweite
```

```
. b6db  4c 7e 86 jmp $867e    READY.

. b6de  24 81    bit $81      Direktmodus ?
. b6eØ  3Ø Ø1    bmi $b6e3    Nein, dann Fehler
. b6e2  6Ø       rts          ENDE

. b6e3  a2 22    ldx #$22     Holt Fehlernummer 34 ('DIRECT MODE ONLY')
. b6e5  4c 83 86 jmp $8683    Fehlerausgabe

. b6e8  :
. b6e8  *** H E L P ***
. b6e8  :
. b6e8  ae ef Ø4 ldx $Ø4ef    Holt Fehlercode
. b6eb  e8       inx
. b6ec  fØ 1b    beq $b7Ø9    Fehler nicht definiert ?, dann $b7Ø9
. b6ee  ad fØ Ø4 lda $Ø4fØ    Holt
. b6f1  ac f1 Ø4 ldy $Ø4f1    ... Fehlerzeilennummer
. b6f4  85 14    sta $14      Zeilennummer
. b6f6  84 15    sty $15      ... zwischenspeichern
. b6f8  2Ø 3d 8a jsr $8a3d    Sucht Zeile und holt Adresse
. b6fb  9Ø Øc    bcc $b7Ø9    Zeile nicht vorhanden ?, dann $b7Ø9
. b6fd  66 53    ror $53      Setzt Flag für Fehlerliste
. b6ff  2Ø 3e 9Ø jsr $9Ø3e    Gibt (CR) aus
. b7Ø2  a6 14    ldx $14      Holt
. b7Ø4  a5 15    lda $15      ... Fehlerzeilennummer
. b7Ø6  2Ø 4Ø 8b jsr $8b4Ø    Zeile ausgeben
. b7Ø9  4c 3e 9Ø jmp $9Ø3e    (CR)-Code ausgeben

     FLASH ON, WENN HELP-FLAG GESETZT

. b7Øc  a6 6Ø    ldx $6Ø      Addiert
. b7Øe  98       tya          ... Offset in
. b7Øf  18       clc          ... Zeile
. b71Ø  65 5f    adc $5f      ... zum Zeiger
. b712  9Ø Ø1    bcc $b715    ... auf gelistete
. b714  e8       inx          ... Zeile
. b715  ec f6 Ø4 cpx $Ø4f6    Adresse
. b718  dØ Øe    bne $b728    ... der
. b71a  cd f5 Ø4 cmp $Ø4f5    ... Fehlerstelle
. b71d  9Ø Ø9    bcc $b728    ... erreicht ?
. b71f  fØ Ø7    beq $b728    Nein, dann $b728
. b721  46 53    lsr $53      Löscht Fehler-Flag
. b723  a9 82    lda #$82     Setzt Blink-Flag
. b725  4c b2 9Ø jmp $9Øb2    Zeichenausgabe

. b728  6Ø       rts          ENDE

. b729  :
. b729  *** K E Y   (ANZEIGE) ***
. b729  :
. b729  dØ 7c    bne $b7a7    Folgt Nummer ?, dann $b7a7
. b72b  a2 ØØ    ldx #$ØØ
```

```
. b72d  a0 00    ldy #$00
. b72f  e8       inx           Holt Code
. b730  bd 5e 05 lda $055e,x   ... aus KEY-Längentabelle
. b733  f0 53    beq $b788     Endmarke ?, dann $b788
. b735  85 77    sta $77       Speichert Textlänge
. b737  86 76    stx $76       Speichert Nummer der Funktionstaste (1-8)
. b739  a2 05    ldx #$05      Holt
. b73b  bd 6e cd lda $cd6e,x   ... den String 'KEY 0,'
. b73e  ca       dex
. b73f  d0 02    bne $b743
. b741  05 76    ora $76       Setzt aktuelle Funktionstasten-Nummer ein
. b743  20 d2 ff jsr $ffd2     (BSOUT) Zeichen ausgeben
. b746  8a       txa
. b747  10 f2    bpl $b73b
. b749  a2 07    ldx #$07      Offset auf Textanfang setzen
. b74b  b9 67 05 lda $0567,y   Holt Zeichen aus Text-Tabelle
. b74e  c8       iny
. b74f  48       pha           Zeichen auf Stack
. b750  86 80    stx $80       Offset zwischenspeichern
. b752  a2 04    ldx #$04
. b754  dd 39 b8 cmp $b839,x   Zeichen gleich einer der Codes : 13,141,34,27 ?
. b757  f0 34    beq $b78d     Ja, dann $b78d
. b759  ca       dex
. b75a  d0 f8    bne $b754
. b75c  a6 80    ldx $80       Holt Offset
. b75e  e0 08    cpx #$08      Offset auf Textanfang ?
. b760  90 07    bcc $b769     Ja, dann $b769
. b762  d0 0a    bne $b76e     Textfortsetzung ?, dann $b76e
. b764  a9 2b    lda #$2b      Holt Pluszeichen (+)
. b766  20 d2 ff jsr $ffd2     Nach Ausgabe von 'CHR$( )','+'-Zeichen ausgeben
. b769  a9 22    lda #$22      Holt Anführungszeichen (")
. b76b  20 d2 ff jsr $ffd2     (BSOUT) Zeichen ausgeben
. b76e  68       pla           Holt Zeichen vom Stack
. b76f  20 d2 ff jsr $ffd2     (BSOUT) Zeichen ausgeben
. b772  a2 09    ldx #$09      Holt Flag für Textfortsetzung
. b774  c6 77    dec $77       Textlänge um 1 erniedrigen
. b776  d0 d3    bne $b74b     Noch nicht Null ?, dann $b74b
. b778  e0 09    cpx #$09      Textfortsetzung ?
. b77a  90 05    bcc $b781     Nein, dann $b781
. b77c  a9 22    lda #$22      Holt Anführungszeichen (")
. b77e  20 d2 ff jsr $ffd2     (BSOUT) Zeichen ausgeben
. b781  a9 8d    lda #$8d      Holt 'SHIFT RETURN'- Code
. b783  20 d2 ff jsr $ffd2     (BSOUT) Zeichen ausgeben
. b786  a6 76    ldx $76       Holt Nummer der Funktionstaste
. b788  e0 08    cpx #$08      Nummer kleiner 8 ?
. b78a  d0 a3    bne $b72f     Ja, dann nächste Taste
. b78c  60       rts           ENDE

. b78d  a6 80    ldx $80       String:
. b78f  bd 30 b8 lda $b830,x   ... '"+CHR$('
. b792  20 d2 ff jsr $ffd2     ... ausgeben (BSOUT)...
. b795  ca       dex
```

```
. b796  eØ Ø3     cpx #$Ø3
. b798  bØ f5     bcs $b78f
. b79a  68        pla
. b79b  2Ø 74 cd  jsr $cd74     Gibt Code als Bytewert aus
. b79e  a9 29     lda #$29      Holt Klammer-Zeichen ')'
. b7aØ  2Ø d2 ff  jsr $ffd2     (BSOUT) Zeichen ausgeben
. b7a3  a2 Ø8     ldx #$Ø8
. b7a5  dØ cd     bne $b774     Springt immer nach $b774

. b7a7  :
. b7a7  *** K E Y   (DEFINITION) ***
. b7a7  :
. b7a7  2Ø 84 9d  jsr $9d84     (GETBYT) Byte übernehmen
. b7aa  ca        dex
. b7ab  eØ Ø8     cpx #$Ø8      Eingabe kleiner 9 ?
. b7ad  9Ø Ø3     bcc $b7b2     Ja, dann $b7b2
. b7af  4c 1c 99  jmp $991c     'ILLEGAL QUANTITY ERROR' ausgeben

. b7b2  86 76     stx $76       Speicher-Funktionstastennummer (-1)
. b7b4  2Ø 91 94  jsr $9491     (CHKCOM) Prüft ob Komma folgt
. b7b7  2Ø 48 9c  jsr $9c48     (FRMEVL), (CHKSTR), (FRESTR)
. b7ba  2Ø c2 b7  jsr $b7c2     Übernimmt Text in Textpuffer
. b7bd  9Ø 72     bcc $b831     Limit nicht überschritten ? ,dann $b831
. b7bf  4c 81 86  jmp $8681     'OUT OF MEMORY ERROR' ausgeben

      TEXT IN TEXTPUFFER ÜBERNEHMEN

. b7c2  85 77     sta $77       Speichert Textlänge
. b7c4  a2 Ø8     ldx #$Ø8
. b7c6  2Ø 3e b8  jsr $b83e     Akku := Offset des Textes (X-Reg.)
. b7c9  8d cd Ø2  sta $Ø2cd     Schreibt Ende des letzten Textes + 1
. b7cc  a6 76     ldx $76       Holt Nummer der Funktionstaste
. b7ce  e8        inx           X-Reg. erhöhen
. b7cf  2Ø 3e b8  jsr $b83e     Akku := Offset des Textes (X-Reg.)
. b7d2  8d ce Ø2  sta $Ø2ce     Schreibt Ende des aktuellen Textes + 1
. b7d5  a6 76     ldx $76       Holt Nummer der Funktionstaste
. b7d7  a5 77     lda $77       Holt Länge des neuen Textes
. b7d9  38        sec           ... und subtrahiert
. b7da  fd 5f Ø5  sbc $Ø55f,x   ... Länge des alten Textes
. b7dd  fØ 35     beq $b814     Beide gleichlang ?, dann $b814
. b7df  9Ø 1d     bcc $b7fe     Ist neuer Text kürzer ?, dann $b7fe
. b7e1  18        clc           Addiert Differenz
. b7e2  6d cd Ø2  adc $Ø2cd     ... zum Ende des alten Textes addieren
. b7e5  bØ 4b     bcs $b832     Bei Überlauf, Fehler
. b7e7  c9 81     cmp #$81      Ist Ergebnis größer $8Ø ?
. b7e9  bØ 47     bcs $b832     Ja, dann ebenfalls Fehler
. b7eb  aa        tax           Akku := neues Ende + 1
. b7ec  ac cd Ø2  ldy $Ø2cd     Y-Reg. := altes Ende + 1
. b7ef  cc ce Ø2  cpy $Ø2ce     Verschiebebereich-Beginn erreicht ?
. b7f2  fØ 2Ø     beq $b814     Ja, dann $b814
. b7f4  88        dey           Quellzeiger um 1 erniedrigen
. b7f5  ca        dex           Zielzeiger um 1 erniedrigen
```

```
. b7f6  b9 67 05 lda $0567,y  Kopiert Zeichen aus Quellbereich
. b7f9  9d 67 05 sta $0567,x  ... in Zielbereich
. b7fc  b0 f1    bcs $b7ef    Springt immer nach $b7ef

. b7fe  6d ce 02 adc $02ce    Addiert altes Ende mit negativer Differenz
. b801  aa       tax          Ergebnis gleich Ende des neuen Textes
. b802  ac ce 02 ldy $02ce    Holt Ende des alten Textes
. b805  cc cd 02 cpy $02cd    Bereits altes Ende des letzten Textes erreicht?
. b808  b0 0a    bcs $b814    Ja, dann $b814
. b80a  b9 67 05 lda $0567,y  Kopiert Zeichen aus Quellbereich
. b80d  9d 67 05 sta $0567,x  ... in Zielbereich
. b810  c8       iny          Erhöht Quellzeiger um 1
. b811  e8       inx          Erhöht Zielzeiger um 1
. b812  90 f1    bcc $b805    Springt immer nach $b805

. b814  a6 76    ldx $76      Holt Nummer der Funktionstaste
. b816  20 3e b8 jsr $b83e    Akku := Offset des Textes (X-Reg.)
. b819  aa       tax
. b81a  a4 76    ldy $76      Holt Nummer der Funktionstaste
. b81c  a5 77    lda $77      Speichert Länge des
. b81e  99 5f 05 sta $055f,y  ... neuen Textes in Längentabelle
. b821  a0 00    ldy #$00
. b823  20 b0 04 jsr $04b0    Holt Zeichen aus String (LDA($22),Y)
. b826  c6 77    dec $77      Länge um 1 verringern
. b828  30 07    bmi $b831    Ende erreicht ?, dann fertig
. b82a  9d 67 05 sta $0567,x  Kopiert Zeichen in Textpuffer
. b82d  e8       inx          Erhöht Offset in Puffer um 1
. b82e  c8       iny          Erhöht Offset in String um 1
. b82f  d0 f2    bne $b823    Springt immer nach $b823

. b831  18       clc
. b832  60       rts          ENDE

. b833  .byte $28 $24 $52 $48 $43 $2b $22  String '($RHC+"'
. b83a  .byte $0d $8d $22 $1b              Code-Tabelle für KEY-Definition

     AKKU := OFFSET DES TEXTES (X-REG.)

. b83e  a9 00    lda #$00     Offset auf Null setzen
. b840  18       clc
. b841  ca       dex          Erniedrigt Funktionstasten-Nummer um 1
. b842  30 ee    bmi $b832    Ende erreicht ?, dann $b832
. b844  7d 5f 05 adc $055f,x  Addiert Länge des Textes (X-Reg.)
. b847  90 f8    bcc $b841    Springt immer nach $b841

. b849  :
. b849  *** S O U N D ***
. b849  :
. b849  20 84 9d jsr $9d84    (GETBYT) Holt Kanal des Tongenerators
. b84c  ca       dex          Erniedrigt Kanal um 1
. b84d  e0 03    cpx #$03     Mehr als 2 Kanäle ?
. b84f  b0 64    bcs $b8b5    Ja, dann Fehler
```

```
. b851  86 80     stx $80       Kanal speichern
. b853  20 de 9d  jsr $9dde     (CHKCOM) und (GETADR) Holt Tonhöhe
. b856  c9 04     cmp #$04      Tonhöhe größer 1023 ?
. b858  b0 5b     bcs $b8b5     Ja, dann Fehler
. b85a  84 7e     sty $7e       Tonhöhe
. b85c  85 7f     sta $7f       ... zwischenspeichern
. b85e  20 de 9d  jsr $9dde     (CHKCOM) und (GETADR) Holt Tondauer
. b861  a6 80     ldx $80       Holt Kanal
. b863  e0 02     cpx #$02      Kanal 2 ?
. b865  d0 01     bne $b868     Nein, dann $b868
. b867  ca        dex           Kanal auf 1 setzen
. b868  48        pha           Legt Tondauer (High) auf Stack
. b869  c0 00     cpy #$00      Tondauer
. b86b  d0 07     bne $b874     ... gleich
. b86d  c9 00     cmp #$00      ... Null ?
. b86f  d0 03     bne $b874     Nein, dann $b874
. b871  c8        iny           Tondauer auf 1 setzen
. b872  d0 0f     bne $b883     Springt immer nach $b883

. b874  98        tya
. b875  48        pha
. b876  20 c0 8c  jsr $8cc0     Abfrage der STOP-Taste
. b879  bd fe 04  lda $04fe,x   Tondauer des laufenden
. b87c  1d fc 04  ora $04fc,x   ... Sounds verstrichen ? (Interrupt)
. b87f  d0 f5     bne $b876     Nein, dann warten
. b881  68        pla
. b882  a8        tay
. b883  98        tya           Invertiert
. b884  49 ff     eor #$ff      ... Tondauer...
. b886  18        clc
. b887  69 01     adc #$01
. b889  78        sei           Sperrt Interrupt
. b88a  9d fc 04  sta $04fc,x   Ergebnis wird für Kanal 1 in $04fc-$04fd
. b88d  68        pla           ... für Kanal 2 in $04fe-$04ff gespeichert
. b88e  49 ff     eor #$ff      ... Im Interrupt werden beide Zähler
. b890  69 00     adc #$00      ... stets um 1 erniedrigt. Ist die Tondauer
. b892  9d fe 04  sta $04fe,x   ... abgelaufen, wird der Sound abgeschaltet
. b895  a5 7e     lda $7e       Holt Tonhöhe (Low)
. b897  9d 0e ff  sta $ff0e,x   Schreibt Tonhöhe nach $ff0e bzw. $ff0f
. b89a  bd b8 b8  lda $b8b8,x   Holt $02 bzw. $00
. b89d  aa        tax
. b89e  bd 10 ff  lda $ff10,x   Holt Tonhöhe (High)
. b8a1  29 fc     and #$fc      Isoliert
. b8a3  05 7f     ora $7f       ... Bit 0 und 1
. b8a5  9d 10 ff  sta $ff10,x   Schreibt Tonhöhe (High) nach $ff12 bzw. $ff10
. b8a8  a6 80     ldx $80       Holt Kanal
. b8aa  bd ba b8  lda $b8ba,x   Setzt je nach Kanal (0,1,2)
. b8ad  0d 11 ff  ora $ff11     ... die Bits 4,5 oder 6
. b8b0  8d 11 ff  sta $ff11     ... in Register $ff11
. b8b3  58        cli           Gewährt Interrupt
. b8b4  60        rts           ENDE
```

```
. b8b5  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. b8b8  .byte $02, $00
. b8ba  .byte $10, $20, $40   Werte für Bit 4, Bit 5 bzw. Bit 6

. b8bd  :
. b8bd  *** V O L ***
. b8bd  :
. b8bd  20 84 9d jsr $9d84    (FRMEVL) und (GETBYT) Holt Lautstärke
. b8c0  e0 09    cpx #$09     Lautstärke größer 8 ?
. b8c2  b0 f1    bcs $b8b5    Ja, dann Fehler
. b8c4  86 80    stx $80      Speichert Lautstärke
. b8c6  ad 11 ff lda $ff11    Holt alten
. b8c9  29 f0    and #$f0     ... Wert des Registers
. b8cb  05 80    ora $80      ... Setzt Bit 0-3 je nach Lautstärke
. b8cd  8d 11 ff sta $ff11    Wert zurückschreiben
. b8d0  60       rts          ENDE

. b8d1  :
. b8d1  *** P A I N T ***
. b8d1  :
. b8d1  20 b6 c3 jsr $c3b6    Holt Farbquelle
. b8d4  a2 04    ldx #$04     Holt Zeiger in Koordinatentabelle
. b8d6  20 d9 c3 jsr $c3d9    Holt Koordinaten
. b8d9  20 7b c3 jsr $c37b    Koordinaten in Tabelle schreiben
. b8dc  20 a5 c3 jsr $c3a5    Holt Modus
. b8df  e0 02    cpx #$02     Modus größer 1 ?
. b8e1  90 03    bcc $b8e6    Nein, dann $b8e6
. b8e3  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. b8e6  8a       txa          Modus nach Akku
. b8e7  4a       lsr
. b8e8  6a       ror
. b8e9  85 8b    sta $8b      Ergibt Begrenzungsflag (Bit 7)
. b8eb  10 04    bpl $b8f1    Begrenzungsflag nicht gesetzt ?, dann $b8f1
. b8ed  a5 84    lda $84      Holt Farbquelle
. b8ef  f0 07    beq $b8f8    Farbquelle gleich Null ?, dann fertig
. b8f1  20 f3 c1 jsr $c1f3    Prüft, ob Punkt (X,Y) gesetzt
. b8f4  b0 02    bcs $b8f8    Punkt nicht im Grafikbereich ?, dann $b8f8
. b8f6  d0 01    bne $b8f9    Punkt nicht gesetzt ?, dann $b8f9
. b8f8  60       rts          ENDE

. b8f9  20 54 a9 jsr $a954    GARBAGE COLLECT
. b8fc  a5 31    lda $31      Holt Variablenende
. b8fe  85 22    sta $22      ... und
. b900  a5 32    lda $32      ... schreibt Zeiger
. b902  85 23    sta $23      ... Koordinaten-Stack
. b904  38       sec          Maximaler
. b905  a5 33    lda $33      ... Stack-Bereich
. b907  e9 03    sbc #$03     ... ist
. b909  85 19    sta $19      ... String-Anfang - 3...
. b90b  a5 34    lda $34
```

```
. b9Ød  e9 ØØ    sbc #$ØØ
. b9Øf  85 1a    sta $1a

     X - SCHLEIFE

. b911  a2 ØØ    ldx #$ØØ
. b913  86 89    stx $89      Initialisiert Flag für 'Stack-Eintrag Links'
. b915  86 8a    stx $8a      Initialisiert Flag für 'Stack-Eintrag Rechts'
. b917  ae af Ø2 ldx $Ø2af    Y-Koordinate
. b91a  dØ Ø3    bne $b91f    ... des
. b91c  ce bØ Ø2 dec $Ø2bØ    ... Grafik-Cursor
. b91f  ce af Ø2 dec $Ø2af    ... um 1 vermindern
. b922  2Ø f3 c1 jsr $c1f3    Prüft, ob Punkt (X,Y) gesetzt
. b925  bØ Ø2    bcs $b929    Punkt nicht im Grafikbereich ?, dann $b929
. b927  dØ ee    bne $b917    Punkt nicht gesetzt ?, dann Y weiter vermindern
. b929  ee af Ø2 inc $Ø2af    Wenn Grenze erreicht, dann
. b92c  dØ Ø3    bne $b931    ... wieder einen Schritt
. b92e  ee bØ Ø2 inc $Ø2bØ    ... nach oben

     Y - SCHLEIFE

. b931  2Ø c3 c1 jsr $c1c3    Punkt (X,Y) setzen/löschen
. b934  ae ad Ø2 ldx $Ø2ad    X-Koordinate
. b937  dØ Ø3    bne $b93c    ... des
. b939  ce ae Ø2 dec $Ø2ae    ... Grafik-Cursor
. b93c  ce ad Ø2 dec $Ø2ad    ... um 1 vermindern
. b93f  a5 89    lda $89      Holt Flag für 'Stack-Eintrag Links'
. b941  2Ø 9f b9 jsr $b99f    Stack-Eintrag und Flag setzen
. b944  85 89    sta $89      Flag zurückschreiben
. b946  18       clc          Erhöht
. b947  ad ad Ø2 lda $Ø2ad    ... X-Koordinate
. b94a  69 Ø2    adc #$Ø2     ... um
. b94c  8d ad Ø2 sta $Ø2ad    ... 2 Byte...
. b94f  9Ø Ø3    bcc $b954
. b951  ee ae Ø2 inc $Ø2ae
. b954  a5 8a    lda $8a      Holt Flag für 'Stack-Eintrag Rechts'
. b956  2Ø 9f b9 jsr $b99f    Stack-Eintrag und Flag setzen
. b959  85 8a    sta $8a      Flag zurückschreiben
. b95b  ae ad Ø2 ldx $Ø2ad    Wiederherstellung
. b95e  dØ Ø3    bne $b963    ... der
. b96Ø  ce ae Ø2 dec $Ø2ae    ... ursprünglichen
. b963  ce ad Ø2 dec $Ø2ad    ... X-Koordinate
. b966  ee af Ø2 inc $Ø2af    Y-Koordinate
. b969  dØ Ø3    bne $b96e    ... um 1
. b96b  ee bØ Ø2 inc $Ø2bØ    ... erhöhen
. b96e  2Ø f3 c1 jsr $c1f3    Prüft, ob Punkt (X,Y) gesetzt
. b971  bØ Ø2    bcs $b975    Punkt nicht im Grafikbereich ?, dann $b975
. b973  dØ bc    bne $b931    Punkt nicht gesetzt ?, dann Y-Schleife
. b975  a2 Ø3    ldx #$Ø3
. b977  aØ ØØ    ldy #$ØØ
. b979  a5 23    lda $23      Holt Zeiger
. b97b  c5 32    cmp $32      ... in
```

```
. b97d  dØ Ø6     bne $b985     ... Koordinaten-Stack
. b97f  a5 22     lda $22       ... und prüft ob
. b981  c5 31     cmp $31       ... Stack leer ist
. b983  fØ 17     beq $b99c     Stack leer ?, dann fertig
. b985  a5 22     lda $22       Zeiger
. b987  dØ Ø2     bne $b98b     ... in Koordinaten-Stack
. b989  c6 23     dec $23       ... um 1
. b98b  c6 22     dec $22       ... erniedrigen
. b98d  2Ø bØ Ø4  jsr $Ø4bØ     Holt nächste X-,Y-Koordinate (LDA($22),Y)
. b99Ø  9d ad Ø2  sta $Ø2ad,x   ... aus Stack und schreibt
. b993  ca        dex           ... diese in die Koordinaten-Tabelle...
. b994  1Ø ef     bpl $b985
. b996  2Ø cØ 8c  jsr $8ccØ     Abfrage der STOP-Taste
. b999  4c 11 b9  jmp $b911     X-Schleife

. b99c  4c 7b c3  jmp $c37b     Wiederherstellung der Anfangskoordinaten

. b99f  48        pha           Flag auf Stack retten
. b9aØ  2Ø f3 c1  jsr $c1f3     Prüft, ob Punkt (X,Y) gesetzt
. b9a3  bØ 2b     bcs $b9dØ     Punkt nicht im Grafikbereich ?, dann $b9dØ
. b9a5  fØ 29     beq $b9dØ     Punkt gesetzt, dann $b9dØ
. b9a7  68        pla           Flag zurückholen
. b9a8  dØ 29     bne $b9d3     Flag bereits gesetzt ?, dann $b9d3
. b9aa  aa        tax           X-Reg. := Null
. b9ab  a8        tay           Y-Reg. := Null
. b9ac  a5 23     lda $23       Befindet sich
. b9ae  c5 1a     cmp $1a       ... Stack-Zeiger
. b9bØ  9Ø Øb     bcc $b9bd     ... schon
. b9b2  dØ Ø6     bne $b9ba     ... am
. b9b4  a5 22     lda $22       ... oberen
. b9b6  c5 19     cmp $19       ... Stack-Ende ?
. b9b8  9Ø Ø3     bcc $b9bd     Nein, dann $b9bd
. b9ba  4c 81 86  jmp $8681     'OUT OF MEMORY ERROR' ausgeben

. b9bd  bd ad Ø2  lda $Ø2ad,x   Legt Koordinaten aus
. b9cØ  91 22     sta ($22),y   ... Tabelle auf Stack
. b9c2  e6 22     inc $22       Erhöht
. b9c4  dØ Ø2     bne $b9c8     ... Stack-Zeiger
. b9c6  e6 23     inc $23       ... um 1
. b9c8  e8        inx           Zähler um 1 erhöhen
. b9c9  eØ Ø4     cpx #$Ø4      Schon 4. Koordinate kopiert ?
. b9cb  dØ fØ     bne $b9bd     Nein, dann weiterkopieren
. b9cd  a9 8Ø     lda #$8Ø      Holt Flag-Wert für 'Stack-Eintrag'
. b9cf  6Ø        rts           ENDE

. b9dØ  68        pla
. b9d1  a9 ØØ     lda #$ØØ      Holt Flag-Wert für 'kein Stack-Eintrag'
. b9d3  6Ø        rts           ENDE

. b9d4  :
. b9d4  *** C H A R ***
. b9d4  :
```

```
. b9d4  20 b9 c3 jsr $c3b9    Holt Farb-Parameter
. b9d7  20 d8 9d jsr $9dd8    (CHKCOM) und (GETBYT) Holt Spalte
. b9da  e0 28    cpx #$28     Spaltenwert zulässig ?
. b9dc  b0 0a    bcs $b9e8    Nein, dann Fehler
. b9de  8e da 02 stx $02da    Spalte abspeichern
. b9e1  20 d8 9d jsr $9dd8    (CHKCOM) und (GETBYT) Holt Spalte
. b9e4  e0 19    cpx #$19     Zeilenwert zulässig ?
. b9e6  90 03    bcc $b9eb    Ja, dann $b9eb
. b9e8  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. b9eb  8e db 02 stx $02db    Zeile abspeichern
. b9ee  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. b9f1  20 48 9c jsr $9c48    (FRESTR)
. b9f4  8d ea 02 sta $02ea    Stringlänge abspeichern
. b9f7  98       tya          Holt Adresse des Deskriptors (High)
. b9f8  48       pha          Auf Stack retten
. b9f9  8a       txa          Holt Adresse des Deskriptors (Low)
. b9fa  48       pha          Auf Stack retten
. b9fb  20 a5 c3 jsr $c3a5    (GETBYT) Holt RVS-Parameter
. b9fe  8a       txa
. b9ff  6a       ror
. ba00  6e b9 02 ror $02b9    Reverse wenn Bit 7 gesetzt
. ba03  68       pla          Zeiger auf
. ba04  85 22    sta $22      ... Deskriptor
. ba06  68       pla          ... wieder vom
. ba07  85 23    sta $23      ... Stack holen
. ba09  a5 83    lda $83      Grafik-Modus eingeschaltet ?
. ba0b  d0 1b    bne $ba28    Ja, dann $ba28
. ba0d  ae db 02 ldx $02db    Holt Zeile
. ba10  ac da 02 ldy $02da    Holt Spalte
. ba13  18       clc
. ba14  20 f0 ff jsr $fff0    (PLOT) Setzt Cursor
. ba17  a0 00    ldy #$00     Initialisiert Offset in String
. ba19  cc ea 02 cpy $02ea    Stringende bereits erreicht ?
. ba1c  f0 09    beq $ba27    Ja, dann $ba27
. ba1e  20 b0 04 jsr $04b0    Holt Zeichen aus String (LDA($22),Y)
. ba21  20 4c ff jsr $ff4c    PRINT
. ba24  c8       iny          Erhöht Offset um 1
. ba25  d0 f2    bne $ba19    Springt immer nach $ba19

. ba27  60       rts          ENDE

. ba28  20 bf c7 jsr $c7bf    Prüft ob GRAPHIC eingeschaltet
. ba2b  a5 86    lda $86      Vordergrund-Farbe
. ba2d  48       pha          ... auf Stack retten
. ba2e  a5 84    lda $84      Aktuelle Farbe
. ba30  48       pha          ... auf Stack retten
. ba31  24 83    bit $83      Multicolor-Modus ?
. ba33  10 0f    bpl $ba44    Nein, dann $ba44
. ba35  68       pla          Grafikmodus auf Stack retten
. ba36  f0 12    beq $ba4a    Aktueller Farbcode gleich Null ?, dann $ba4a
. ba38  4a       lsr
```

```
. ba39  f0 0f    beq $ba4a
. ba3b  a6 85    ldx $85       Holt Multicolor-Farbe 1
. ba3d  90 0d    bcc $ba4c
. ba3f  ae 16 ff ldx $ff16     Holt Multicolor-Farbe 2
. ba42  b0 08    bcs $ba4c

. ba44  ae 15 ff ldx $ff15     Holt Hintergrundfarbe
. ba47  68       pla           Aktueller Farbcode gleich Null ?
. ba48  f0 02    beq $ba4c     Ja, dann $ba4c
. ba4a  a6 86    ldx $86       Holt Vordergrundfarbe
. ba4c  86 86    stx $86       Schreibt Vordergrundfarbe
. ba4e  ae db 02 ldx $02db     Holt Zeile
. ba51  a0 00    ldy #$00      Initialisiert
. ba53  8c dc 02 sty $02dc     ... Offset in String
. ba56  ac dc 02 ldy $02dc     Holt Offset in String
. ba59  ee dc 02 inc $02dc     Erhöht Offset um 1
. ba5c  20 b0 04 jsr $04b0     Holt Zeichen aus String (LDA($22),Y)
. ba5f  ce ea 02 dec $02ea     Erniedrigt Stringlänge um 1
. ba62  30 17    bmi $ba7b     String zuende ?, dann $ba7b
. ba64  ac da 02 ldy $02da     Holt Spalte
. ba67  20 7f ba jsr $ba7f     Zeichen in Bit-Speicher bringen
. ba6a  ee da 02 inc $02da     Erhöht Spalte um 1
. ba6d  c0 27    cpy #$27      Letzte Spalte erreicht ?
. ba6f  90 e5    bcc $ba56     Nein, dann $ba56
. ba71  a0 00    ldy #$00      Spalte
. ba73  8c da 02 sty $02da     ... initialisieren
. ba76  e8       inx           Erhöht Zeile um 1
. ba77  e0 18    cpx #$18      Letzte Zeile erreicht ?
. ba79  90 db    bcc $ba56     Nein, dann $ba56
. ba7b  68       pla           Vordergrundfarbe
. ba7c  85 86    sta $86       ... wiederherstellen
. ba7e  60       rts           ENDE

     ZEICHEN IN BIT-SPEICHER BRINGEN

. ba7f  48       pha           Holt Zeichen aus String
. ba80  20 1a c2 jsr $c21a     Speichert Farbe
. ba83  20 91 c2 jsr $c291     Setzt Zeiger in Bitspeicher
. ba86  a9 00    lda #$00      Es wird
. ba88  85 7e    sta $7e       ... der Bildschirmcode
. ba8a  68       pla           ... aus dem
. ba8b  48       pha           ... Zeichencode erzeugt
. ba8c  0a       asl           ... und
. ba8d  26 7e    rol $7e       ... mit 8
. ba8f  0a       asl           ... multipliziert
. ba90  0a       asl           Akku := Ergebnis (Low)
. ba91  26 7e    rol $7e       $7e  := Ergebins (High)
. ba93  85 24    sta $24       Schreibt Zeiger in Zeichengenerator (Low)
. ba95  a5 7e    lda $7e       Holt Ergebnis (High)
. ba97  6d e4 02 adc $02e4     ... und addiert Anfang des Zeichengenerators
. ba9a  85 25    sta $25       Schreibt Zeiger in Zeichengenerator (High)
. ba9c  98       tya           Spalte merken
```

```
. ba9d  48       pha
. ba9e  a0 07    ldy #$07      8 Bitstrings je Zeichen
. baa0  ad b9 02 lda $02b9     Holt Reverse-Flag
. baa3  0a       asl           RVS-Flag in Carry schieben
. baa4  b1 24    lda ($24),y   Holt Bitstring aus Tabelle
. baa6  90 02    bcc $baaa     RVS-Flag nicht gesetzt ?, dann $baaa
. baa8  49 ff    eor #$ff      Invertiert Bitstring
. baaa  24 83    bit $83       Multicolor-Grafik ?
. baac  10 2b    bpl $bad9     Nein, dann $bad9
. baae  29 aa    and #$aa      Und-Verknüpfung mit %10101010
. bab0  85 7e    sta $7e       Bitstring zwischenspeichern
. bab2  a5 84    lda $84       Holt aktuellen Farbcode
. bab4  d0 0f    bne $bac5     Farbcode ungleich Null ?, dann $bac5
. bab6  a5 7e    lda $7e       Holt Bitstring
. bab8  b0 07    bcs $bac1     Revers-Flag gesetzt ?, dann $bac1
. baba  4a       lsr
. babb  45 7e    eor $7e
. babd  49 aa    eor #$aa      Exklusiv-Oder-Verknüpfung mit %10101010
. babf  d0 18    bne $bad9     Springt immer nach $bad9

. bac1  09 55    ora #$55      Oder-Verknüpfung mit %01010101
. bac3  d0 14    bne $bad9     Springt immer nach $bad9

. bac5  c9 02    cmp #$02      Aktueller Farbcode gleich 2 ?
. bac7  d0 04    bne $bacd     Nein, dann $bacd
. bac9  a5 7e    lda $7e       Holt Bitstring
. bacb  b0 0c    bcs $bad9     Springt immer nach $bad9

. bacd  90 07    bcc $bad6     Aktueller Farbcode kleiner 2 ?, dann $bad6
. bacf  a5 7e    lda $7e       Holt Bitstring
. bad1  4a       lsr
. bad2  45 7e    eor $7e
. bad4  90 03    bcc $bad9     Springt immer nach $bad9

. bad6  a5 7e    lda $7e       Holt Bitstring
. bad8  4a       lsr
. bad9  91 8c    sta ($8c),y   Schreibt Bitstring in Bit-Speicher
. badb  88       dey           Bereits 8 Bitstrings bearbeitet ?
. badc  10 c2    bpl $baa0     Nein, dann $baa0
. bade  68       pla           Holt Spalte
. badf  a8       tay           ... nach Y-Reg.
. bae0  68       pla           Holt Zeichen aus String
. bae1  60       rts           ENDE

. bae2  :
. bae2  *** B O X ***
. bae2  :
. bae2  20 b6 c3 jsr $c3b6     Holt Farb-Parameter
. bae5  a2 1f    ldx #$1f
. bae7  20 f4 c3 jsr $c3f4     (CHKCOM) und 1. Koordinatenpaar holen
. baea  a2 2b    ldx #$2b
. baec  20 d9 c3 jsr $c3d9     2. Koordinatenpaar holen
```

```
. baef  20 8f c3 jsr $c38f    Holt Winkel
. baf2  8c d0 02 sty $02d0    Schreibt Winkel
. baf5  8d d1 02 sta $02d1    ... (im Gradmaß)
. baf8  20 a5 c3 jsr $c3a5    Holt Füllparameter
. bafb  e0 02    cpx #$02     Füllparameter kleiner 2 ?
. bafd  90 03    bcc $bb02    Ja, dann $bb02
. baff  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. bb02  8e e8 02 stx $02e8    Schreibt Füllparameter
. bb05  8a       txa          Füllparameter
. bb06  48       pha          ... auf Stack retten
. bb07  20 b4 bb jsr $bbb4    Drehung ausführen
. bb0a  68       pla          Holt Füllparameter
. bb0b  d0 1c    bne $bb29    Fläche füllen ?, dann $bb29
. bb0d  f0 03    beq $bb12    Nur Rand zeichen ?, dann $bb12

. bb0f  20 36 bc jsr $bc36    Berechnet Bildpunkt bei Drehung
. bb12  20 da c0 jsr $c0da    Zeichnet Strecke
. bb15  ad ca 02 lda $02ca    Alle Seiten gezeichnet ?
. bb18  d0 f5    bne $bb0f    Nein, dann $bb0f
. bb1a  a2 04    ldx #$04
. bb1c  bd d7 02 lda $02d7,x  Stellt
. bb1f  9d ac 02 sta $02ac,x  ... Ausgangs-
. bb22  ca       dex          ... Koordinaten
. bb23  d0 f7    bne $bb1c    ... wieder her
. bb25  8e e8 02 stx $02e8    Initialisiert Füllparameter
. bb28  60       rts          ENDE

. bb29  a2 00    ldx #$00
. bb2b  ad c5 02 lda $02c5    Holt Quadrant-1
. bb2e  4a       lsr          Gerade ?
. bb2f  90 02    bcc $bb33    Ja, dann $bb33
. bb31  a2 02    ldx #$02
. bb33  bd dc 02 lda $02dc,x  Berechnet
. bb36  8d d6 02 sta $02d6    ... X-
. bb39  bd dd 02 lda $02dd,x  ... bzw.
. bb3c  8d d7 02 sta $02d7    ... Y-Koordinatendifferenz
. bb3f  a9 00    lda #$00     Löscht
. bb41  a2 03    ldx #$03     ... Bereich
. bb43  9d d2 02 sta $02d2,x  ... $02d2-$02d5...
. bb46  ca       dex
. bb47  10 fa    bpl $bb43
. bb49  a2 07    ldx #$07     Legt
. bb4b  bd ad 02 lda $02ad,x  ... Koordinaten
. bb4e  48       pha          ... auf
. bb4f  ca       dex          ... Stack
. bb50  10 f9    bpl $bb4b
. bb52  20 da c0 jsr $c0da    Zeichnet Strecke
. bb55  a2 00    ldx #$00     Holt
. bb57  68       pla          ... ursprüngliche
. bb58  9d ad 02 sta $02ad,x  ... Koordinaten
. bb5b  e8       inx          ... wieder
```

```
. bb5c  e0 08    cpx #$08      ... vom
. bb5e  d0 f7    bne $bb57     ... Stack
. bb60  ad d6 02 lda $02d6     Vermindert
. bb63  d0 05    bne $bb6a     ... Koordinaten-
. bb65  ce d7 02 dec $02d7     ... Differenz
. bb68  30 b0    bmi $bb1a     Unterlauf ?, dann fertig
. bb6a  ce d6 02 dec $02d6
. bb6d  a2 25    ldx #$25
. bb6f  a0 1b    ldy #$1b      Y-Reg. gleich Offset des Cosinus
. bb71  ad c5 02 lda $02c5     Holt Quadrant-1
. bb74  4a       lsr           Gerade ?
. bb75  90 02    bcc $bb79     Ja, dann $bb79
. bb77  a0 19    ldy #$19      Sonst Y-Reg. gleich Offset des Sinus
. bb79  a9 00    lda #$00
. bb7b  4a       lsr
. bb7c  48       pha
. bb7d  20 f6 c2 jsr $c2f6     (Akku, Y-Reg) := KO(Y) + KO(X)
. bb80  9d ad 02 sta $02ad,x   Speichert
. bb83  98       tya           ... aktuelle
. bb84  9d ae 02 sta $02ae,x   ... X-Position des Grafik-Cursors
. bb87  68       pla
. bb88  90 02    bcc $bb8c
. bb8a  09 a0    ora #$a0      Flags für Koordinatenerhöhung
. bb8c  e8       inx
. bb8d  e8       inx
. bb8e  a0 19    ldy #$19      Y-Reg. gleich Offset des Sinus
. bb90  4e c5 02 lsr $02c5     Quadrant-1 gerade ?
. bb93  90 02    bcc $bb97     Ja, dann $bb97
. bb95  a0 1b    ldy #$1b      Sonst Y-Reg. gleich Offset des Cosinus
. bb97  2e c5 02 rol $02c5     Quadrant wiederherstellen
. bb9a  e0 27    cpx #$27
. bb9c  f0 dd    beq $bb7b     Dasselbe mit der anderen Koordinate
. bb9e  a2 06    ldx #$06
. bba0  0a       asl           Flags für Koordinatenerhöhung gleich Null ?
. bba1  f0 bd    beq $bb60     Ja, dann $bb60
. bba3  90 08    bcc $bbad     Flag nicht gesetzt ?, dann $bbad
. bba5  fe ad 02 inc $02ad,x   Koordinate
. bba8  d0 03    bne $bbad     ... um 1
. bbaa  fe ae 02 inc $02ae,x   ... erhöhen
. bbad  0a       asl           Holt nächstes Flag nach Carry
. bbae  ca       dex
. bbaf  ca       dex
. bbb0  10 f1    bpl $bba3     Weitermachen, bis
. bbb2  30 95    bmi $bb49     ... alle Koordinaten bearbeitet sind

. bbb4  :
. bbb4  *** DREHUNG AUSFÜHREN ***
. bbb4  :
. bbb4  a0 23    ldy #$23      Holt Offset des Drehwinkels
. bbb6  20 56 bc jsr $bc56     Berechnet Sinus und Cosinus
. bbb9  a2 1f    ldx #$1f
. bbbb  a0 2b    ldy #$2b
```

```
. bbbd  98       tya
. bbbe  48       pha
. bbbf  20 22 c3 jsr $c322    (Akku,Y-Reg.) := ABS(KO(Y)-KO(X)), P=SGN
. bbc2  9d b1 02 sta $02b1,x
. bbc5  9d b5 02 sta $02b5,x
. bbc8  9d bd 02 sta $02bd,x
. bbcb  98       tya
. bbcc  9d b2 02 sta $02b2,x
. bbcf  9d b6 02 sta $02b6,x
. bbd2  9d be 02 sta $02be,x
. bbd5  68       pla
. bbd6  a8       tay
. bbd7  20 f6 c2 jsr $c2f6    (Akku,Y-Reg.) := KO(Y) + KO(X)
. bbda  9d ad 02 sta $02ad,x
. bbdd  98       tya
. bbde  9d ae 02 sta $02ae,x
. bbe1  a0 2d    ldy #$2d
. bbe3  e8       inx
. bbe4  e8       inx
. bbe5  e0 21    cpx #$21
. bbe7  f0 d4    beq $bbbd
. bbe9  a9 90    lda #$90     Holt SIN/COS-Flags (0: SIN; 1: COS)
. bbeb  20 d5 bc jsr $bcd5    Multipliziert Koordinaten mit SIN/COS
. bbee  ad c5 02 lda $02c5
. bbf1  29 03    and #$03
. bbf3  8d c5 02 sta $02c5
. bbf6  aa       tax
. bbf7  bd 18 bc lda $bc18,x  Holt Vorzeichenstring
. bbfa  20 36 bc jsr $bc36    Berechnet Bildpunkt des 1. Punktes
. bbfd  20 7b c3 jsr $c37b    Speichert Koordinaten
. bc00  ad ca 02 lda $02ca    Holt Plus/Minus-Flags
. bc03  20 36 bc jsr $bc36    Berechnet Bildpunkt des 2. Punktes
. bc06  ae c5 02 ldx $02c5    Holt Quadrant-1
. bc09  bd 18 bc lda $bc18,x  Holt Bitstring aus Tabelle
. bc0c  29 f0    and #$f0     UND-Verknüpfung mit %11110000
. bc0e  8d cb 02 sta $02cb    Als
. bc11  bd 1c bc lda $bc1c,x  ... Plus/Minus-
. bc14  8d ca 02 sta $02ca    ... Flags speichern
. bc17  60       rts          ENDE

. bc18  .byte %10111110 %11100100 %01000001 %00011011
. bc1c  .byte %01000001 %00011011 %10111110 %11100100

. bc20  :
. bc20  *** DREI NAMEN ***
. bc20  :
. bc20  .byte 'fred b'
. bc26  .byte $0d
. bc27  .byte 'terry r'
. bc2e  .byte $0d
. bc2f  .byte 'mike i'
. bc35  .byte $0d
```

```
. bc36  :
. bc36  *** BILDPUNKT BEI DREHUNG BERECHNEN ***
. bc36  :
. bc36  20 05 bd jsr $bd05    Berechnet verdoppelte Koordinaten
. bc39  a2 04    ldx #$04
. bc3b  bd ae 02 lda $02ae,x
. bc3e  0a       asl          Höchstes Bit abtrennen
. bc3f  7e ae 02 ror $02ae,x  Koordinaten
. bc42  7e ad 02 ror $02ad,x  ... durch 2 dividieren
. bc45  90 08    bcc $bc4f    War niederwertiges Bit gleich 0 ?, dann $bc4f
. bc47  fe ad 02 inc $02ad,x  Koordinaten
. bc4a  d0 03    bne $bc4f    ... aufrunden...
. bc4c  fe ae 02 inc $02ae,x
. bc4f  e8       inx
. bc50  e8       inx
. bc51  e0 06    cpx #$06
. bc53  f0 e6    beq $bc3b
. bc55  60       rts          ENDE

. bc56  :
. bc56  *** SIN & COS BERECHNEN ***
. bc56  :
. bc56  20 18 c3 jsr $c318    Holt Winkel nach (Akku,Y-Reg.)
. bc59  a2 00    ldx #$00
. bc5b  e8       inx          Winkel
. bc5c  38       sec          ... durch
. bc5d  e9 5a    sbc #$5a     ... 90
. bc5f  b0 fa    bcs $bc5b    ... dividieren...
. bc61  88       dey
. bc62  10 f7    bpl $bc5b
. bc64  8e c5 02 stx $02c5    Schreibt Quadrant des Winkels
. bc67  48       pha          Negativer Winkel im 4. Quadranten
. bc68  69 5a    adc #$5a     Positiver Winkel im 1. Quadranten
. bc6a  20 76 bc jsr $bc76    Berechnet Sinus
. bc6d  68       pla          Negativer Winkel im 4. Quadranten
. bc6e  18       clc
. bc6f  49 ff    eor #$ff     Vorzeichen
. bc71  69 01    adc #$01     ... wechseln
. bc73  ce c5 02 dec $02c5    Quadrant zur COS-Berechnung um 1 erniedrigen
. bc76  a2 ff    ldx #$ff
. bc78  e8       inx          Winkel
. bc79  38       sec          ... durch
. bc7a  e9 0a    sbc #$0a     ... 10
. bc7c  b0 fa    bcs $bc78    ... dividieren...
. bc7e  69 0a    adc #$0a
. bc80  85 8e    sta $8e      Rest zwischenspeichern
. bc82  8a       txa          Zehnerstelle
. bc83  0a       asl          ... mit 2 multiplizieren
 =27
```

```
. bc84  aa       tax          Ergebnis gleich Zeiger in Sinus-Tabelle
. bc85  bd b4 c4 lda $c4b4,x  Setzt Zeiger
. bc88  bc b3 c4 ldy $c4b3,x  ... auf Sinus-Tabelle
. bc8b  18       clc
. bc8c  c6 8e    dec $8e      Verringert Einerstelle um 1
. bc8e  30 0c    bmi $bc9c    Einerstelle negativ ?, dann fertig
. bc90  7d c8 c4 adc $c4c8,x  Addiert Interpolationswert
. bc93  48       pha          ... aus Tabelle
. bc94  98       tya          ... so oft
. bc95  7d c7 c4 adc $c4c7,x  ... wie Einerstelle
. bc98  a8       tay          ... des Winkels
. bc99  68       pla          ... angibt
. bc9a  90 ef    bcc $bc8b    Springt immer nach $bc8b

. bc9c  48       pha          Ergebnis (Low) auf Stack retten
. bc9d  a2 00    ldx #$00     X-Reg. gleich 0
. bc9f  ad c5 02 lda $02c5    Quadrant
. bca2  4a       lsr          ... ungerade ?
. bca3  b0 02    bcs $bca7    Ja, dann $bca7
. bca5  a2 02    ldx #$02     Sonst: X-Reg. gleich 2
. bca7  68       pla          Holt Ergebnis vom Stack
. bca8  9d c6 02 sta $02c6,x  $02c6-$02c7 := ABS (SIN(W))
. bcab  98       tya          $02c8-$02c9 := ABS (COS(W))
. bcac  9d c7 02 sta $02c7,x
. bcaf  60       rts          ENDE

. bcb0  :
. bcb0  *** (AKKU, Y-REG.) := KO(X) * SIN/COS ***
. bcb0  :
. bcb0  a0 19    ldy #$19     Carry := 0  Sinus
. bcb2  90 02    bcc $bcb6
. bcb4  a0 1b    ldy #$1b     Carry := 1  Cosinus
. bcb6  ad c5 02 lda $02c5    Holt Quadrant-1
. bcb9  69 02    adc #$02     Bestimmt
. bcbb  4a       lsr          ... Vorzeichen
. bcbc  4a       lsr          '+' setzt Carry als Vorzeichenflag
. bcbd  08       php
. bcbe  20 18 c3 jsr $c318    (akku,Y-Reg.) := SIN/COS
. bcc1  c0 ff    cpy #$ff     Ist SIN/COS kleiner als 1
. bcc3  90 07    bcc $bccc    Ja, dann $bccc
. bcc5  8a       txa
. bcc6  a8       tay          Y-Reg. := X-Reg.
. bcc7  20 18 c3 jsr $c318    (Akku, Y-Reg.) := KO(X)
. bcca  b0 03    bcs $bccf    Springt immer nach $bccf

. bccc  20 37 c3 jsr $c337    (Akku, Y-Reg.) := (Akku, Y-Reg.) * KO(X)
. bccf  28       plp          Holt Vorzeichen
. bcd0  b0 1b    bcs $bced    Vorzeichen positiv ?, dann $bced
. bcd2  4c 27 c3 jmp $c327    (Akku, Y-Reg.) := -(Akku, Y-Reg.)

. bcd5  :
. bcd5  *** MULTIPLIZIERT 4 KOORDINATEN MIT SIN/COS ***
```

```
. bcd5  :
. bcd5  8d ca Ø2 sta $Ø2ca    Schreibt SIN/COS-Flags
. bcd8  a2 23    ldx #$23
. bcda  Øe ca Ø2 asl $Ø2ca    Schiebt SIN/COS-Flag ins Carry
. bcdd  2Ø bØ bc jsr $bcbØ    (Akku, Y-Reg.) := KO(X) * SIN/COS
. bceØ  9d ad Ø2 sta $Ø2ad,x  KO(X) := (AKKU, Y-Reg.)...
. bce3  98       tya
. bce4  9d ae Ø2 sta $Ø2ae,x
. bce7  e8       inx
. bce8  e8       inx
. bce9  eØ 2b    cpx #$2b     Schon 4 Koordinaten multipliziert ?
. bceb  9Ø ed    bcc $bcda    Nein, dann $bcda
. bced  6Ø       rts          ENDE

. bcee  :
. bcee  *** KOORDINATEN - TRANSFORMATION ***
. bcee  :
. bcee  aØ 2b    ldy #$2b
. bcfØ  2Ø 56 bc jsr $bc56    Berechnet Sinus und Cosinus
. bcf3  a2 Ø7    ldx #$Ø7
. bcf5  bd dc Ø2 lda $Ø2dc,x  Koordinaten
. bcf8  9d dØ Ø2 sta $Ø2dØ,x  ... umkopieren...
. bcfb  ca       dex
. bcfc  1Ø f7    bpl $bcf5
. bcfe  a9 5Ø    lda #$5Ø     Holt SIN/COS-Flags (Ø:SIN, 1:COS)
. bdØØ  2Ø d5 bc jsr $bcd5    Koordinaten mit SIN/COS multiplizieren
. bdØ3  a9 1Ø    lda #$1Ø
. bdØ5  8d ca Ø2 sta $Ø2ca    Schreibt SIN/COS-Flags
. bdØ8  aØ 1f    ldy #$1f
. bdØa  a2 23    ldx #$23
. bdØc  Øe cb Ø2 asl $Ø2cb    Schiebt Plus/Minus-Flag ins Carry
. bdØf  2e ca Ø2 rol $Ø2ca
. bd12  2Ø f4 c2 jsr $c2f4    (Akku, Y-Reg.) := KO(Y) +/- KO(X) Carry=Ø : +
. bd15  e8       inx
. bd16  e8       inx
. bd17  Øe cb Ø2 asl $Ø2cb    Schiebt Plus/Minus-Flag ins Carry
. bd1a  2e ca Ø2 rol $Ø2ca
. bd1d  2Ø fØ c2 jsr $c2fØ    (Akku, Y-Reg.) := (A,Y) +/- KO(X) Carry=Ø : +
. bd2Ø  48       pha          Legt
. bd21  98       tya          ... (Akku, Y-Reg.)
. bd22  48       pha          ... auf Stack
. bd23  aØ 21    ldy #$21
. bd25  e8       inx
. bd26  e8       inx
. bd27  eØ 27    cpx #$27     Routine zweimal durchlaufen ?
. bd29  fØ e1    beq $bdØc    Nein, dann $bdØc
. bd2b  a2 Ø3    ldx #$Ø3
. bd2d  68       pla          Holt Ergebnisse
. bd2e  9d b1 Ø2 sta $Ø2b1,x  ... vom Stack
. bd31  ca       dex          ... und
. bd32  1Ø f9    bpl $bd2d    ... speichert diese
. bd34  6Ø       rts          ENDE
```

```
. bd35  :
. bd35  *** G S H A P E ***
. bd35  :
. bd35  20 bf c7 jsr $c7bf    Prüft ob Grafik eingeschaltet
. bd38  20 48 9c jsr $9c48    (FRMEVL) und (CHKSTR) und (FRESTR)
. bd3b  8d cf 02 sta $02cf    Schreibt Stringlänge
. bd3e  86 24    stx $24      Schreibt
. bd40  84 25    sty $25      ... Stringadresse
. bd42  a2 04    ldx #$04
. bd44  20 d9 c3 jsr $c3d9    Holt Koordinaten
. bd47  20 a5 c3 jsr $c3a5    (GETBYT) Parameter übernehmen
. bd4a  e0 05    cpx #$05     Parameter größer 4 ?
. bd4c  90 03    bcc $bd51    Nein, dann $bd51
. bd4e  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. bd51  8e d0 02 stx $02d0    Ausgabeparameter zwischenspeichern
. bd54  a2 03    ldx #$03
. bd56  ac cf 02 ldy $02cf    Holt Stringlänge
. bd59  c0 05    cpy #$05     Stringlänge größer 4 ?
. bd5b  b0 01    bcs $bd5e    Ja, dann $bd5e
. bd5d  60       rts          ENDE

. bd5e  88       dey
. bd5f  20 bb 04 jsr $04bb    Holt die letzten 4 Zeichen aus String
. bd62  9d d5 02 sta $02d5,x  ... (Diese 4 Byte
. bd65  ca       dex          ... enthalten Höhe
. bd66  10 f6    bpl $bd5e    ... und Breite des Shapes)
. bd68  8e d1 02 stx $02d1    Initialisiert Zeiger in String
. bd6b  20 7b c3 jsr $c37b    Koordinaten umspeichern
. bd6e  ad d5 02 lda $02d5    Speichert
. bd71  8d d9 02 sta $02d9    ... Breite
. bd74  ad d6 02 lda $02d6    ... des
. bd77  8d da 02 sta $02da    ... Shape-Rechtecks
. bd7a  a9 08    lda #$08     Initialisiert
. bd7c  8d e5 02 sta $02e5    ... Bit-Zähler
. bd7f  ee d1 02 inc $02d1    Erhöht Zeiger in String
. bd82  ac d1 02 ldy $02d1
. bd85  20 bb 04 jsr $04bb    Holt Zeichen aus String (LDA($24),Y)
. bd88  8d d3 02 sta $02d3    Zeichen speichern
. bd8b  20 f3 c1 jsr $c1f3    Testet Punkt (X-Reg., Y-Reg)
. bd8e  8d d2 02 sta $02d2    Schreibt Punktbits (Bit 0 und Bit 1)
. bd91  0e d3 02 asl $02d3    Shiftet Zeichen aus String nach links
. bd94  2a       rol
. bd95  ce e5 02 dec $02e5    Erniedrigt Bitzähler um 1
. bd98  24 83    bit $83      Prüft Grafik-Modus
. bd9a  10 07    bpl $bda3    Nicht Multicolor ?, dann $bda3
. bd9c  0e d3 02 asl $02d3    Shiftet Zeichen aus String nach links
. bd9f  2a       rol
. bda0  ce e5 02 dec $02e5    Erniedrigt Bitzähler um 1
. bda3  ae d0 02 ldx $02d0    Modus P
. bda6  e0 03    cpx #$03     P kleiner als 3 ?
```

```
. bda8  9Ø Øc     bcc $bdb6     Ja, dann $bdb6
. bdaa  fØ Ø5     beq $bdb1     P gleich 3 ?, dann $bdb1
. bdac  4d d2 Ø2  eor $Ø2d2     P gleich 4 ?
. bdaf  bØ 11     bcs $bdc2     Ja, dann XOR-Modus

. bdb1  2d d2 Ø2  and $Ø2d2     P gleich 3 ?
. bdb4  bØ Øc     bcs $bdc2     Ja, dann AND-Modus

. bdb6  eØ Ø1     cpx #$Ø1
. bdb8  9Ø Ø8     bcc $bdc2     P gleich Ø ?, dann keine Veränderung
. bdba  fØ Ø4     beq $bdcØ     P gleich 1 ?, dann $bdcØ
. bdbc  Ød d2 Ø2  ora $Ø2d2     P gleich 2 ?, dann OR-Modus

. bdbf  .byte $2c

. bdcØ  49 ff     eor #$ff      P gleich 1 ?, dann Inverse-Modus
. bdc2  29 Ø3     and #$Ø3      Isoliert Bit Ø und Bit 1
. bdc4  24 83     bit $83       Multicolor-Modus ?
. bdc6  3Ø Ø2     bmi $bdca     Ja, dann $bdca
. bdc8  29 Ø1     and #$Ø1      Isoliert Bit Ø
. bdca  85 84     sta $84       Speichert Akku als Farbquelle
. bdcc  2Ø c3 c1  jsr $c1c3     Punkt (X-Reg., Y-Reg.) setzen/löschen
. bdcf  ee ad Ø2  inc $Ø2ad     Erhöht
. bdd2  dØ Ø3     bne $bdd7     ... X-Koordinaten
. bdd4  ee ae Ø2  inc $Ø2ae     ... um 1
. bdd7  38        sec           Verringert
. bdd8  ad d9 Ø2  lda $Ø2d9     ... die
. bddb  24 83     bit $83       ... Breite
. bddd  1Ø Ø3     bpl $bde2     ... des
. bddf  e9 Ø2     sbc #$Ø2      ... Shape-Bereichs

. bde1  .byte $2c               ... je

. bde2  e9 Ø1     sbc #$Ø1      ... nach
. bde4  8d d9 Ø2  sta $Ø2d9     ... Grafikmodus
. bde7  ad da Ø2  lda $Ø2da     ... um 2 (Multicolor)
. bdea  e9 ØØ     sbc #$ØØ      ... bzw.
. bdec  8d da Ø2  sta $Ø2da     ... um 1 (HIRES)
. bdef  bØ 2d     bcs $be1e     Kein Unterlauf ?, dann $be1e
. bdf1  a2 Ø1     ldx #$Ø1
. bdf3  bd d5 Ø2  lda $Ø2d5,x   Ausgangswert der Breite
. bdf6  9d d9 Ø2  sta $Ø2d9,x   ... wiederherstellen
. bdf9  bd b1 Ø2  lda $Ø2b1,x   Koordinaten
. bdfc  9d ad Ø2  sta $Ø2ad,x   ... wiederherstellen
. bdff  ca        dex
. beØØ  1Ø f1     bpl $bdf3
. beØ2  ee af Ø2  inc $Ø2af     Erhöht
. beØ5  dØ Ø3     bne $beØa     ... Y-Koordinate
. beØ7  ee bØ Ø2  inc $Ø2bØ     ... um 1
. beØa  38        sec           Verringert
. beØb  ad d7 Ø2  lda $Ø2d7     ... Höhe
. beØe  e9 Ø1     sbc #$Ø1      ... des
```

```
. be1Ø  8d d7 Ø2 sta $Ø2d7     ... Shape-
. be13  ad d8 Ø2 lda $Ø2d8     ... Bereichs
. be16  e9 ØØ    sbc #$ØØ     ... um
. be18  8d d8 Ø2 sta $Ø2d8     ... 1
. be1b  bØ Ø9    bcs $be26     Kein Unterlauf ?, dann $be26
. be1d  6Ø       rts           ENDE

. be1e  ad e5 Ø2 lda $Ø2e5     Bitzähler gleich Null ?
. be21  fØ Ø3    beq $be26     Ja, dann $be26
. be23  4c 8b bd jmp $bd8b     Zum nächsten X-Wert

. be26  4c 7a bd jmp $bd7a     Zum nächsten Zeichen

. be29  :
. be29  *** S S H A P E ***
. be29  :
. be29  2Ø bf c7 jsr $c7bf     Prüft ob Grafik eingeschaltet ist
. be2c  2Ø a5 96 jsr $96a5     Variable suchen oder anlegen
. be2f  8d db Ø2 sta $Ø2db     Schreibt Zeiger
. be32  8c dc Ø2 sty $Ø2dc     ... auf Deskriptor
. be35  24 Ød    bit $Ød       String ?
. be37  3Ø Ø3    bmi $be3c     Ja, dann $be3c
. be39  4c 24 93 jmp $9324     'TYPE MISMATCH ERROR' ausgeben

. be3c  a2 28    ldx #$28      Holt
. be3e  2Ø f4 c3 jsr $c3f4     ... Anfangskoordinaten
. be41  a2 Ø4    ldx #$Ø4      Holt
. be43  2Ø d9 c3 jsr $c3d9     ... Endkoordinaten
. be46  a2 2a    ldx #$2a
. be48  aØ Ø6    ldy #$Ø6
. be4a  a9 Ø2    lda #$Ø2
. be4c  85 8e    sta $8e
. be4e  2Ø 22 c3 jsr $c322     (Akku,Y-Reg) := ABS(KO(Y-Reg)-KO(X-Reg)), P=SGN
. be51  aa       tax
. be52  98       tya
. be53  48       pha
. be54  a4 8e    ldy $8e
. be56  2Ø 82 c3 jsr $c382     KO(Y-Reg.) := KO(Y-Reg.+4)
. be59  9Ø Øc    bcc $be67     Koordinatendifferenz Negativ ?, dann $be67
. be5b  b9 d5 Ø2 lda $Ø2d5,y   Übernimmt
. be5e  99 ad Ø2 sta $Ø2ad,y   ... eingegebene
. be61  b9 d6 Ø2 lda $Ø2d6,y   ... Koordinaten...
. be64  99 ae Ø2 sta $Ø2ae,y
. be67  8a       txa           Speichert Betrag
. be68  99 d5 Ø2 sta $Ø2d5,y   ... der Differenz
. be6b  99 de Ø2 sta $Ø2de,y   ... als Höhe
. be6e  68       pla           ... bzw. Breite
. be6f  99 d6 Ø2 sta $Ø2d6,y   ... des
. be72  99 df Ø2 sta $Ø2df,y   ... Shape-Bereichs
. be75  a2 28    ldx #$28      Offset für X-Koordinaten initialisieren
. be77  aØ Ø4    ldy #$Ø4
. be79  c6 8e    dec $8e
```

```
. be7b  c6 8e     dec $8e
. be7d  f0 cf     beq $be4e     Nach 1. Durchlauf, nach $be4e springen
. be7f  a0 ff     ldy #$ff
. be81  8c d1 02  sty $02d1     Initialisiert Zeiger in String
. be84  ad ad 02  lda $02ad     Umspeichern
. be87  8d d9 02  sta $02d9     ... der
. be8a  ad ae 02  lda $02ae     ... X-Koordinaten...
. be8d  8d da 02  sta $02da
. be90  98        tya           Akku gleich $ff
. be91  20 5c 9b  jsr $9b5c     Platz für String reservieren
. be94  20 64 c2  jsr $c264     Setzt Zeiger in Bitspeicher
. be97  b1 8c     lda ($8c),y   Holt Byte aus Bitspeicher
. be99  90 0e     bcc $bea9     Gleich Punkt im Grafik-Bereich ?, dann $bea9
. be9b  ad ad 02  lda $02ad     Holt X-Koordinate (Low)
. be9e  24 83     bit $83       Multicolor ?
. bea0  10 02     bpl $bea4     Nein, dann $bea4
. bea2  38        sec           Bit 1
. bea3  2a        rol           ... einrotieren
. bea4  29 07     and #$07      Isoliert Bit 0 bis Bit 2
. bea6  aa        tax           Ergebnis gleich Bitoffset für X
. bea7  a9 00     lda #$00      Übernimmt bitweise Null
. bea9  24 83     bit $83       Multicolor ?
. beab  10 01     bpl $beae     Nein, dann $beae
. bead  ca        dex           Erniedrigt Bitoffset um 1
. beae  8e dd 02  stx $02dd     Bitoffset zwischenspeichern
. beb1  0a        asl           Rotiert
. beb2  ca        dex           ... Bitwert
. beb3  10 fc     bpl $beb1     ... ins Carry
. beb5  6a        ror           Wert zurück ins Bit 7
. beb6  85 8e     sta $8e       ... und abspeichern
. beb8  a9 08     lda #$08      Holt Inkrement für X
. beba  24 83     bit $83       Multicolor ?
. bebc  10 01     bpl $bebf     Nein, dann $bebf
. bebe  4a        lsr           Inkrement halbieren
. bebf  18        clc           Inkrement
. bec0  6d ad 02  adc $02ad     ... zu
. bec3  8d ad 02  sta $02ad     ... X-Koordinate
. bec6  90 03     bcc $becb     ... addieren...
. bec8  ee ae 02  inc $02ae
. becb  20 64 c2  jsr $c264     Setzt Zeiger in Bitspeicher
. bece  a9 00     lda #$00      Punkt im Grafikbereich ?
. bed0  b0 02     bcs $bed4     Nein, dann $bed4
. bed2  b1 8c     lda ($8c),y   Holt Byte aus Bitspeicher
. bed4  85 8f     sta $8f       Byte zwischenspeichern
. bed6  ae dd 02  ldx $02dd     Holt Bitoffset zum letzten Punkt
. bed9  4a        lsr           Byte aus Bitspeicher nach rechts schieben
. beda  e8        inx           X-Reg. erhöhen
. bedb  e0 08     cpx #$08      X ungleich 8 ?
. bedd  d0 fa     bne $bed9     Ja, dann $bed9
. bedf  05 8e     ora $8e       Ergebnis vom letzten Punkt einordnen
. bee1  ee d1 02  inc $02d1     Erhöht Zeiger in String um 1
. bee4  ac d1 02  ldy $02d1     Noch Platz
```

```
. bee7  c0 fc     cpy #$fc      ... für 4 Byte im String ?
. bee9  90 03     bcc $beee     Ja, dann $beee
. beeb  4c 4c cc  jmp $cc4c     'STRING TOO LONG ERROR' ausgeben

. beee  91 62     sta ($62),y   Schreibt Byte in String
. bef0  ae dd 02  ldx $02dd     Holt Bitoffset
. bef3  ad d5 02  lda $02d5     Holt Breite des Shape-Bereichs
. bef6  38        sec
. bef7  24 83     bit $83       Multicolor ?
. bef9  10 03     bpl $befe     Nein, dann $befe
. befb  e9 04     sbc #$04      Breite um 4 (Multicolor)

. befd  .byte $2c

. befe  e9 08     sbc #$08      bzw. um 8 (HIRES)
. bf00  8d d5 02  sta $02d5     ... verringern
. bf03  a5 8f     lda $8f       Holt letztes Byte aus Bitspeicher
. bf05  b0 aa     bcs $beb1     Kein Unterlauf ?, dann $beb1
. bf07  ce d6 02  dec $02d6     Erniedrigt Breite des Shapes (High) um 1
. bf0a  10 a5     bpl $beb1     Kein Unterlauf ?, dann $beb1
. bf0c  ae d7 02  ldx $02d7     Holt Höhe des Shapes (Low)
. bf0f  d0 42     bne $bf53     Höhe ungleich Null ?, dann $bf53
. bf11  ce d8 02  dec $02d8     Erniedrigt Höhe des Shapes (High) um 1
. bf14  10 3d     bpl $bf53     Kein Unterlauf ?, dann $bf53
. bf16  24 83     bit $83       Multicolor ?
. bf18  10 06     bpl $bf20     Nein, dann $bf20
. bf1a  0e de 02  asl $02de     Verdoppelt
. bf1d  2e df 02  rol $02df     ... Breite
. bf20  a2 00     ldx #$00      Schreibt
. bf22  bd de 02  lda $02de,x   ... Breite
. bf25  c8        iny           ... und
. bf26  91 62     sta ($62),y   ... Höhe
. bf28  e8        inx           ... als
. bf29  e0 04     cpx #$04      ... die letzten
. bf2b  d0 f5     bne $bf22     ... 4 Byte in den String
. bf2d  c8        iny
. bf2e  8c de 02  sty $02de     Schreibt Stringlänge
. bf31  a5 62     lda $62       Stringadresse
. bf33  8d df 02  sta $02df     ... umspeichern...
. bf36  a5 63     lda $63
. bf38  8d e0 02  sta $02e0
. bf3b  a9 de     lda #$de      Übergibt
. bf3d  85 64     sta $64       ... Adresse
. bf3f  a9 02     lda #$02      ... des
. bf41  85 65     sta $65       ... Quelldeskriptors
. bf43  ad db 02  lda $02db     Übergibt
. bf46  85 49     sta $49       ... Zeiger
. bf48  ad dc 02  lda $02dc     ... auf
. bf4b  85 4a     sta $4a       ... Zieldeskriptor
. bf4d  20 40 8f  jsr $8f40     Kopiert String in Stringbereich
. bf50  4c 7b c3  jmp $c37b     KO(0) := KO(4)
```

```
. bf53  ce d7 Ø2 dec $Ø2d7    Erniedrigt Höhe des Shapes um 1
. bf56  ee af Ø2 inc $Ø2af    Erhöht
. bf59  dØ Ø3    bne $bf5e    ... Y-Koordinate
. bf5b  ee bØ Ø2 inc $Ø2bØ    ... um 1
. bf5e  ad d9 Ø2 lda $Ø2d9    Ausgangswert
. bf61  8d ad Ø2 sta $Ø2ad    ... der
. bf64  ad da Ø2 lda $Ø2da    ... X-Koordinate
. bf67  8d ae Ø2 sta $Ø2ae    ... wiederherstellen
. bf6a  ad de Ø2 lda $Ø2de    Ausgangswert
. bf6d  8d d5 Ø2 sta $Ø2d5    ... der
. bf7Ø  ad df Ø2 lda $Ø2df    ... Shape-Breite
. bf73  8d d6 Ø2 sta $Ø2d6    ... wiederherstellen
. bf76  4c 94 be jmp $be94    Springt nach $be94

. bf79  :
. bf79  *** R G R ***
. bf79  :
. bf79  a5 83    lda $83      Grafik-Modus
. bf7b  18       clc          ... auswerten
. bf7c  2a       rol
. bf7d  2a       rol
. bf7e  2a       rol
. bf7f  69 ØØ    adc #$ØØ     Je nach Modus
. bf81  a8       tay          ... ist Y-Reg. gleich Ø, 1, 2, 3 oder 4
. bf82  4c 81 9a jmp $9a81    (Y-Reg.) als Gleitkommazahl nach FAC

. bf85  :
. bf85  *** R C L R ***
. bf85  :
. bf85  38       sec          Carry := 1

. bf86  .byte $24

. bf87  :
. bf87  *** R L U M ***
. bf87  :
. bf87  18       clc          Carry := Ø
. bf88  Ø8       php          Carry retten
. bf89  2Ø 87 9d jsr $9d87    (GETBYT) Byte übernehmen
. bf8c  ad 19 ff lda $ff19    Holt Rahmenfarbe
. bf8f  29 7f    and #$7f     Löscht Bit 7
. bf91  eØ Ø4    cpx #$Ø4     Eingabe gleich 4 ?
. bf93  fØ 19    beq $bfae    Ja, dann $bfae
. bf95  bØ 27    bcs $bfbe    Eingabe größer 4 ?, dann Fehler
. bf97  ad 15 ff lda $ff15    Holt Hintergrundfarbe
. bf9a  29 7f    and #$7f     Löscht Bit 7
. bf9c  ca       dex          Eingabe gleich Ø ?
. bf9d  3Ø Øf    bmi $bfae    Ja, dann $bfae
. bf9f  a5 86    lda $86      Holt Vordergrundfarbe
. bfa1  ca       dex          Eingabe gleich 1 ?
. bfa2  3Ø Øa    bmi $bfae    Ja, dann $bfae
. bfa4  a5 85    lda $85      Holt Multicolor 1
```

```
. bfa6  ca       dex          Eingabe gleich 2 ?
. bfa7  30 05    bmi $bfae    Ja, dann $bfae
. bfa9  ad 16 ff lda $ff16    Holt Multicolor 2
. bfac  29 7f    and #$7f     Löscht Bit 7
. bfae  28       plp          Carry wiederherstellen
. bfaf  b0 05    bcs $bfb6    Aufruf über 'RCLR' ? ,dann $bfb6
. bfb1  4a       lsr          Bits 4, 5 und 6
. bfb2  4a       lsr          ... (für LUM)
. bfb3  4a       lsr          ... werden nach unten
. bfb4  4a       lsr          ... geschoben
. bfb5  18       clc          Null addieren
. bfb6  69 00    adc #$00     Bei Aufruf durch 'RCLR' wird 1 addiert
. bfb8  29 0f    and #$0f     Löscht oberes Halbbyte
. bfba  a8       tay          Zahl nach Y-Reg.
. bfbb  4c 81 9a jmp $9a81    (Y-Reg.) als Gleitkomma-Zahl nach FAC

. bfbe  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. bfc1  :
. bfc1  *** J O Y ***
. bfc1  :
. bfc1  20 87 9d jsr $9d87    (GETBYT) Byte übernehmen
. bfc4  ca       dex          Eingabe um 1 erniedrigen
. bfc5  e0 02    cpx #$02     Eingabe zwischen 1 und 2 ?
. bfc7  b0 f5    bcs $bfbe    Nein, dann Fehler
. bfc9  bd fb bf lda $bffb,x  Holt Bitmuster zur
. bfcc  aa       tax          ... Abfrage des Joystickports
. bfcd  78       sei
. bfce  8e 00 ff stx $ff00    Aktiviert Joystickport
. bfd1  ad 00 ff lda $ff00    Liest Port
. bfd4  8e 00 ff stx $ff00    Daten
. bfd7  cd 00 ff cmp $ff00    ... stabil ?
. bfda  d0 f2    bne $bfce    Nein, dann warten
. bfdc  58       cli
. bfdd  49 ff    eor #$ff     Invertiert Datenbyte
. bfdf  a8       tay          im Y-Reg. zwischenspeichern
. bfe0  29 0f    and #$0f     Isoliert Richtungsbit
. bfe2  aa       tax
. bfe3  bd f0 bf lda $bff0,x  Holt Richtungswert aus Tabelle
. bfe6  c0 0f    cpy #$0f     Feuerknopf aktiv ?
. bfe8  90 02    bcc $bfec    Nein, dann $bfec
. bfea  09 80    ora #$80     Setzt Bit 7
. bfec  a8       tay
. bfed  4c 81 9a jmp $9a81    (Y-Reg.) als Gleitkommazahl nach FAC

. bff0  .byte $00 $01 $05 $00 $07 $08 $06 $00 $03 $02 $04
. bffb  .byte $fa $fd

. bffd  :
. bffd  *** R D O T ***
. bffd  :
. bffd  20 87 9d jsr $9d87    (GETBYT) Parameter übernehmen
```

```
. c000  e0 02    cpx #$02     Parameter kleiner 2 ?
. c002  90 0d    bcc $c011    Ja, dann Koordinaten ausgeben
. c004  d0 b8    bne $bfbe    Sonst Fehler
. c006  20 f3 c1 jsr $c1f3    Prüft ob Punkt gesetzt
. c009  a8       tay          Gesetzt: Y-Reg = Farbe; nicht gesetzt: Y-Reg =0
. c00a  90 02    bcc $c00e    Punkt im Grafikbereich ?, dann $c00e
. c00c  a0 00    ldy #$00     Ansonsten Null ausgeben
. c00e  4c 81 9a jmp $9a81    (Y-Reg) als Gleitkommazahl nach FAC

. c011  8a       txa          Parameter
. c012  0a       asl          ... verdoppeln...
. c013  aa       tax
. c014  bd ad 02 lda $02ad,x  Parameter gleich 0 ?, dann X-Koordinate
. c017  a8       tay          ... Parameter gleich 1 ?, dann Y-Koordinate
. c018  bd ae 02 lda $02ae,x  ... nach (Akku, Y-Reg.) holen
. c01b  4c 71 94 jmp $9471    (Akku, Y-Reg.) als Gleitkommazahl nach FAC

. c01e  :
. c01e  *** C I R C L E ***
. c01e  :
. c01e  20 b6 c3 jsr $c3b6    Holt Farb-Parameter
. c021  a2 1f    ldx #$1f
. c023  20 d9 c3 jsr $c3d9    Holt Koordinaten des Mittelpunktes
. c026  20 8f c3 jsr $c38f    Holt X-Radius
. c029  8c d0 02 sty $02d0    X-Radius
. c02c  8d d1 02 sta $02d1    ... abspeichern
. c02f  20 8f c3 jsr $c38f    Holt Y-Radius
. c032  8c d2 02 sty $02d2    Y-Radius
. c035  8d d3 02 sta $02d3    ... abspeichern
. c038  08       php
. c039  a2 23    ldx #$23
. c03b  20 d3 c2 jsr $c2d3    SCALE
. c03e  28       plp
. c03f  b0 11    bcs $c052    Letzte Eingabe nicht leer ?, dann $c052
. c041  ad d0 02 lda $02d0    Sonst ist
. c044  8d d2 02 sta $02d2    ... Y-Radius gleich
. c047  ad d1 02 lda $02d1    ... X-Radius...
. c04a  24 83    bit $83      Multicolor ?
. c04c  10 04    bpl $c052    Nein, dann $c052
. c04e  0e d2 02 asl $02d2    Sonst Radius
. c051  2a       rol          ... verdoppeln...
. c052  8d d3 02 sta $02d3
. c055  20 8f c3 jsr $c38f    Holt Anfangswinkel des Bogens
. c058  8c d8 02 sty $02d8    Anfangswinkel
. c05b  8d d9 02 sta $02d9    ... abspeichern
. c05e  20 8f c3 jsr $c38f    Holt Endwinkel des Bogens
. c061  8c da 02 sty $02da    Endwinkel
. c064  8d db 02 sta $02db    ... abspeichern
. c067  20 8f c3 jsr $c38f    Holt Drehwinkel
. c06a  85 80    sta $80      Vertauschen
. c06c  98       tya          ... von
. c06d  a4 80    ldy $80      ... Akku und Y-Reg
```

```
. c06f  20 59 bc jsr $bc59    Berechnet Sinus und Cosinus
. c072  a2 2d    ldx #$2d
. c074  a0 2b    ldy #$2b
. c076  20 05 c3 jsr $c305    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c079  90 0e    bcc $c089    Anfangswinkel kleiner Endwinkel ?, dann $c089
. c07b  a9 68    lda #$68     (akku, Y-Reg.) gleich
. c07d  a0 01    ldy #$01     ... 360 Grad
. c07f  20 f9 c2 jsr $c2f9    (Akku, Y-Reg.) := KO(X-Reg.) - (Akku, Y-Reg.)
. c082  9d ad 02 sta $02ad,x  Erhöht Endwinkel
. c085  98       tya          ... um
. c086  9d ae 02 sta $02ae,x  ... 360 Grad
. c089  a2 03    ldx #$03     Radien-
. c08b  bd d0 02 lda $02d0,x  ... Werte
. c08e  9d d4 02 sta $02d4,x  ... umkopieren...
. c091  ca       dex
. c092  10 f7    bpl $c08b
. c094  a9 90    lda #$90     Holt SIN/COS-Flags (0:SIN; 1:COS)
. c096  20 d5 bc jsr $bcd5    Multipliziert
. c099  a2 07    ldx #$07     ... Koordinaten
. c09b  bd d0 02 lda $02d0,x  ... mit
. c09e  9d dc 02 sta $02c,x   ... SIN/COS...
. c0a1  ca       dex
. c0a2  10 f7    bpl $c09b
. c0a4  20 ee bc jsr $bcee    Drehung ausführen
. c0a7  20 7b c3 jsr $c37b    Stellt (X-Reg., Y-Reg.) wieder her
. c0aa  a2 02    ldx #$02     Vorbelegung 2 Grad
. c0ac  20 a7 c3 jsr $c3a7    Holt Segmentwinkel zwischen 2 Segmenten
. c0af  86 e9    stx $e9      Segmentwinkel (in Grad) zwischenspeichern
. c0b1  18       clc
. c0b2  a5 e9    lda $e9      Segmentwinkel gleich Null ?
. c0b4  d0 03    bne $c0b9    Nein, dann kein Fehler
. c0b6  4c 1c 99 jmp $991c    Sonst 'ILLEGAL QUANTITY ERROR' ausgeben

. c0b9  6d d8 02 adc $02d8    Addiert
. c0bc  8d d8 02 sta $02d8    ... Segmentwinkel
. c0bf  90 03    bcc $c0c4    ... zum Startwinkel
. c0c1  ee d9 02 inc $02d9    ... des Bogens
. c0c4  a2 2d    ldx #$2d     Holt Offset des Bogenendes
. c0c6  a0 2b    ldy #$2b     Holt Offset der aktuellen Bogen-Koordinate
. c0c8  20 05 c3 jsr $c305    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c0cb  b0 08    bcs $c0d5    Differenz positiv ?, dann $c0d5
. c0cd  20 ee bc jsr $bcee    Drehung ausführen
. c0d0  20 da c0 jsr $c0da    Strecke zeichnen
. c0d3  90 dd    bcc $c0b2    Springt immer nach $c0b2

. c0d5  a0 2d    ldy #$2d
. c0d7  20 f0 bc jsr $bcf0    Drehung ausführen

. c0da  :
. c0da  *** STRECKE ZEICHNEN ***
. c0da  :
. c0da  a2 02    ldx #$02     Holt Offset von Y1
```

```
. c0dc  a0 06     ldy #$06     Holt Offset von Y2
. c0de  a9 00     lda #$00
. c0e0  9d b9 02  sta $02b9,x  Addend := 0
. c0e3  9d ba 02  sta $02ba,x
. c0e6  20 22 c3  jsr $c322    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c0e9  10 08     bpl $c0f3    Ergebnis nicht negativ ?, dann $c0f3
. c0eb  de b9 02  dec $02b9,x  Addend := -1
. c0ee  de ba 02  dec $02ba,x
. c0f1  d0 0b     bne $c0fe    Springt immer nach $c0fe

. c0f3  c9 00     cmp #$00     Differenz
. c0f5  d0 04     bne $c0fb    ... gleich
. c0f7  c0 00     cpy #$00     ... Null ?
. c0f9  f0 03     beq $c0fe    Ja, dann $c0fe
. c0fb  fe b9 02  inc $02b9,x  Addend := 1
. c0fe  9d b5 02  sta $02b5,x  Koordinatendifferenz
. c101  0a        asl          ... abspeichern
. c102  9d bd 02  sta $02bd,x  ... und
. c105  98        tya          ... anschließend
. c106  9d b6 02  sta $02b6,x  ... verdoppeln
. c109  2a        rol          ... und
. c10a  9d be 02  sta $02be,x  ... ebenfalls abspeichern
. c10d  ca        dex
. c10e  ca        dex
. c10f  a0 04     ldy #$04
. c111  e0 00     cpx #$00
. c113  f0 c9     beq $c0de    Springt nach dem 1. Durchlauf nach $c0de
. c115  a2 0a     ldx #$0a
. c117  a0 08     ldy #$08
. c119  20 05 c3  jsr $c305    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c11c  a9 00     lda #$00     (positives Ergebnis setzt Carry)
. c11e  2a        rol
. c11f  2a        rol
. c120  8d c3 02  sta $02c3    Akku = 0 wenn Ergebnis negativ, sonst Akku = 2
. c123  49 02     eor #$02
. c125  8d c4 02  sta $02c4    Akku = 2  bzw. Akku = 0
. c128  18        clc          Addiert
. c129  a9 10     lda #$10     ... 16 zum
. c12b  6d c3 02  adc $02c3    ... Akku
. c12e  a8        tay          Y-Reg. = 16 bwz. Y-Reg = 18
. c12f  48        pha          Ergebnis auf Stack
. c130  49 02     eor #$02
. c132  aa        tax          X-Reg. = 18 bwz. X-Reg = 16
. c133  20 05 c3  jsr $c305    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c136  9d ad 02  sta $02ad,x
. c139  98        tya
. c13a  9d ae 02  sta $02ae,x
. c13d  68        pla
. c13e  a8        tay          Y-Reg. = 16 bzw. Y-Reg = 18
. c13f  18        clc          Vom Akku werden zunächst
. c140  a9 08     lda #$08     ... 8 subtrahiert und
. c142  6d c4 02  adc $02c4    ... anschließend 2 bzw. 0 hinzuaddiert
```

```
. c145  aa       tax          X-Reg. = 10 bwz. X-Reg = 8
. c146  20 05 c3 jsr $c305    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c149  8d c1 02 sta $02c1
. c14c  8c c2 02 sty $02c2
. c14f  20 a5 c1 jsr $c1a5    Setzt Punkt (X-Reg., Y-Reg.)
. c152  ac c4 02 ldy $02c4
. c155  38       sec
. c156  b9 b5 02 lda $02b5,y  Verringert
. c159  e9 01    sbc #$01     Koordinatendifferenz...
. c15b  99 b5 02 sta $02b5,y
. c15e  b0 0b    bcs $c16b
. c160  b9 b6 02 lda $02b6,y
. c163  e9 00    sbc #$00
. c165  99 b6 02 sta $02b6,y
. c168  b0 01    bcs $c16b    Kein Unterlauf, dann $c16b
. c16a  60       rts          ENDE

. c16b  ae c3 02 ldx $02c3
. c16e  ad c2 02 lda $02c2
. c171  30 06    bmi $c179
. c173  20 94 c1 jsr $c194    Addiert Inkrement
. c176  ae c4 02 ldx $02c4
. c179  18       clc
. c17a  ad c1 02 lda $02c1
. c17d  7d bd 02 adc $02bd,x
. c180  8d c1 02 sta $02c1
. c183  ad c2 02 lda $02c2
. c186  7d be 02 adc $02be,x
. c189  8d c2 02 sta $02c2
. c18c  ae c4 02 ldx $02c4
. c18f  20 94 c1 jsr $c194    Addiert Inkrement
. c192  f0 bb    beq $c14f    Springt immer nach $c14f

     ADDIERT INKREMENT

. c194  a0 02    ldy #$02
. c196  18       clc
. c197  bd ad 02 lda $02ad,x
. c19a  7d b9 02 adc $02b9,x
. c19d  9d ad 02 sta $02ad,x
. c1a0  e8       inx
. c1a1  88       dey
. c1a2  d0 f3    bne $c197
. c1a4  60       rts          ENDE

. c1a5  :
. c1a5  *** DOPPELPUNKT SETZEN ***
. c1a5  :
. c1a5  ad e8 02 lda $02e8    Holt Füllparameter
. c1a8  0d e7 02 ora $02e7    Oder-Verknüpfung mit WIDTH-Parameter
. c1ab  f0 16    beq $c1c3    Beide gleich Null ?, dann $c1c3
. c1ad  ee ad 02 inc $02ad    X-Koordinate
```

```
. c1b0  d0 03    bne $c1b5     ... um 1
. c1b2  ee ae 02 inc $02ae     ... erhöhen
. c1b5  20 c3 c1 jsr $c1c3     Punkt (X-Reg., Y-Reg.) setzen
. c1b8  ae ad 02 ldx $02ad     X-Koordinate
. c1bb  d0 03    bne $c1c0     ... wieder
. c1bd  ce ae 02 dec $02ae     ... um 1
. c1c0  ce ad 02 dec $02ad     ... erniedrigen
. c1c3  :
. c1c3  *** PUNKT SETZEN ***
. c1c3  :
. c1c3  20 ad c2 jsr $c2ad     Zeile und Spalte nach (X-Reg., Y-Reg.)
. c1c6  b0 24    bcs $c1ec     Grafikbreich überschritten ?, dann $c1ec
. c1c8  20 1a c2 jsr $c21a     Speichert Farbe und Helligkeit
. c1cb  20 69 c2 jsr $c269     Setzt Zeiger in Bitspeicher
. c1ce  8d e9 02 sta $02e9     Schreibt Bitwert des Offsets
. c1d1  b1 8c    lda ($8c),y   Holt Byte aus Bit-Speicher
. c1d3  0d e9 02 ora $02e9     Ordnet Bitwert des Offsets ein
. c1d6  24 83    bit $83       Multicolor ?
. c1d8  10 13    bpl $c1ed     Nein, dann $c1ed
. c1da  48       pha           Speichert Byte für Bitspeicher
. c1db  a6 84    ldx $84       Holt Farbparameter
. c1dd  ad e9 02 lda $02e9     Holt Bitwert des Offsets
. c1e0  3d af c4 and $c4af,x   Und-Verknüpfung mit Farbparameter
. c1e3  8d e9 02 sta $02e9     ... ergibt Farbmaske
. c1e6  68       pla           Holt Byte für Bitspeicher
. c1e7  4d e9 02 eor $02e9     XOR-Verknüpfung mit Farbmaske
. c1ea  91 8c    sta ($8c),y   Byte wieder abspeichern
. c1ec  60       rts           ENDE

. c1ed  a6 84    ldx $84       Holt Farbparameter
. c1ef  d0 f9    bne $c1ea     Farbparameter ungleich Null ?, dann $c1ea
. c1f1  f0 f4    beq $c1e7     Farbparameter gleich Null ?, dann $c1e7

. c1f3  :
. c1f3  *** PUNKT TESTEN ***
. c1f3  :
. c1f3  20 64 c2 jsr $c264     Setzt Zeiger in Bitspeicher
. c1f6  b0 21    bcs $c219     Grafikbereich überschritten ?, dann $c219
. c1f8  8d e9 02 sta $02e9     Schreibt Bitwert des Punktes (X-Reg., Y-Reg.)
. c1fb  b1 8c    lda ($8c),y   Holt Byte aus Bitspeicher
. c1fd  2d e9 02 and $02e9     Isoliert Punktbits
. c200  2a       rol           Nach links rotieren
. c201  ca       dex           ... bis die Bits
. c202  10 fc    bpl $c200     ... an den Stellen 0 und 1 stehen
. c204  2a       rol
. c205  24 8b    bit $8b       Flag für Vergleich gesetzt ?
. c207  30 06    bmi $c20f     Nein, dann $c20f
. c209  29 03    and #$03      Bits isolieren
. c20b  c5 84    cmp $84       ... und mit Farb-Parameter
. c20d  18       clc           ... vergleichen
. c20e  60       rts           ENDE
```

```
. c20f  18       clc
. c210  29 03    and #$03      Bits isolieren
. c212  f0 03    beq $c217     Beide Null ?, dann $c217
. c214  a2 00    ldx #$00
. c216  60       rts           ENDE

. c217  a2 ff    ldx #$ff
. c219  60       rts           ENDE

. c21a  :
. c21a  *** FARBE UND HELLIGKEIT SETZEN ***
. c21a  :
. c21a  bd 02 d8 lda $d802,x   Holt Zeilenadresse (Low)
. c21d  85 8c    sta $8c       Schreibt Bit-Map-Zeiger für Farbinformation
. c21f  bd 1b d8 lda $d81b,x   Holt Zeilenadresse (High)
. c222  29 03    and #$03
. c224  48       pha
. c225  09 1c    ora #$1c
. c227  85 8d    sta $8d       Schreibt Bit-Map-Zeiger für Farbinformation
. c229  20 38 c2 jsr $c238     Erzeugt Farbcode
. c22c  91 8c    sta ($8c),y   Speichert Farbcode
. c22e  68       pla
. c22f  09 18    ora #$18
. c231  85 8d    sta $8d
. c233  20 4e c2 jsr $c24e     Erzeugt Helligkeits-Code
. c236  91 8c    sta ($8c),y   Helligkeits-Code speichern

      ERZEUGT FARBCODE

. c238  a5 86    lda $86       Holt Zeichenfarbe
. c23a  0a       asl           Schiebt
. c23b  0a       asl           ... Farbcode
. c23c  0a       asl           ... in oberes
. c23d  0a       asl           ... Halbbyte
. c23e  85 7e    sta $7e       Ergebnis zwischenspeichern
. c240  ad 15 ff lda $ff15     Holt Hintergrundfarbe
. c243  24 83    bit $83       Multicolor ?
. c245  10 02    bpl $c249     Nein, dann $c249
. c247  a5 85    lda $85       Holt Multicolor-Hintergrundfarbe
. c249  29 0f    and #$0f      Isoliert Farbcode
. c24b  05 7e    ora $7e       Ordnet Zeichenfarbe ein
. c24d  60       rts           ENDE

      ERZEUGT HELLIGKEITS-CODE

. c24e  a5 86    lda $86       Holt Zeichenfarbe
. c250  4a       lsr           Schiebt
. c251  4a       lsr           ... Helligkeits-Code
. c252  4a       lsr           ... ins untere
. c253  4a       lsr           ... Halbbyte
. c254  85 7e    sta $7e       Ergebnis zwischenspeichern
. c256  ad 15 ff lda $ff15     Holt Hintergrundfarbe
```

```
. c259  24 83     bit $83       Multicolor ?
. c25b  1Ø Ø2     bpl $c25f     Nein, dann $c25f
. c25d  a5 85     lda $85       Holt Multicolor-Hintergrundfarbe
. c25f  29 fØ     and #$fØ      Isoliert Helligkeits-Code
. c261  Ø5 7e     ora $7e       Ordnet Zeichenhelligkeit ein
. c263  6Ø        rts           ENDE

. c264  :
. c264  *** ZEIGER IN BITSPEICHER SETZEN ***
. c264  :
. c264  2Ø ad c2 jsr $c2ad      Holt Zeile und Spalte nach (X-Reg., Y-Reg.)
. c267  bØ 1f     bcs $c288     Grafikbereich überschritten ?, dann $c288
. c269  2Ø 91 c2 jsr $c291      Holt Byteadresse nach ($8c, $8d)
. c26c  ad af Ø2 lda $Ø2af      Holt Y-Koordinate (Low)
. c26f  29 Ø7     and #$Ø7      Isoliert Bit Ø bis Bit 2
. c271  a8        tay           Ergebnis gleich Y-Offset
. c272  ad ad Ø2 lda $Ø2ad      Holt X-Koordinate (Low)
. c275  24 83     bit $83       Holt Grafik-Modus
. c277  Ø8        php
. c278  1Ø Ø1     bpl $c27b     Nicht Multicolor ?, dann $c27b
. c27a  Øa        asl           Verdoppelt X-Koordinate (Low)
. c27b  29 Ø7     and #$Ø7      Isoliert Bit Ø bis Bit 2
. c27d  aa        tax           Ergebnis gleich X-Offset
. c27e  bd 89 c2 lda $c289,x    Holt Bitwert des X-Offsets
. c281  28        plp           Multicolor ?
. c282  1Ø Ø4     bpl $c288     Nein, dann $c288
. c284  e8        inx           Sonst auch noch
. c285  1d 89 c2 ora $c289,x    ... das nächste Bit setzen
. c288  6Ø        rts           ENDE

. c289  .byte $8Ø $4Ø $2Ø $1Ø $Ø8 $Ø4 $Ø2 $Ø1

. c259  24 83     bit $83       Multicolor ?
. c25b  1Ø Ø2     bpl $c25f     Nein, dann $c25f
. c25d  a5 85     lda $85       Holt Multicolor-Hintergrundfarbe
. c25f  29 fØ     and #$fØ      Isoliert Helligkeits-Code
. c261  Ø5 7e     ora $7e       Ordnet Zeichenhelligkeit ein
. c263  6Ø        rts           ENDE

. c264  :
. c264  *** ZEIGER IN BITSPEICHER SETZEN ***
. c264  :
. c264  2Ø ad c2 jsr $c2ad      Holt Zeile und Spalte nach (X-Reg., Y-Reg.)
. c267  bØ 1f     bcs $c288     Grafikbereich überschritten ?, dann $c288
. c269  2Ø 91 c2 jsr $c291      Holt Byteadresse nach ($8c, $8d)
. c26c  ad af Ø2 lda $Ø2af      Holt Y-Koordinate (Low)
. c26f  29 Ø7     and #$Ø7      Isoliert Bit Ø bis Bit 2
. c271  a8        tay           Ergebnis gleich Y-Offset
. c272  ad ad Ø2 lda $Ø2ad      Holt X-Koordinate (Low)
. c275  24 83     bit $83       Holt Grafik-Modus
. c277  Ø8        php
. c278  1Ø Ø1     bpl $c27b     Nicht Multicolor ?, dann $c27b
```

```
. c27a  Øa       asl          Verdoppelt X-Koordinate (Low)
. c27b  29 Ø7    and #$Ø7     Isoliert Bit Ø bis Bit 2
. c27d  aa       tax          Ergebnis gleich X-Offset
. c27e  bd 89 c2 lda $c289,x  Holt Bitwert des X-Offsets
. c281  28       plp          Multicolor ?
. c282  1Ø Ø4    bpl $c288    Nein, dann $c288
. c284  e8       inx          Sonst auch noch
. c285  1d 89 c2 ora $c289,x  ... das nächste Bit setzen
. c288  6Ø       rts          ENDE

. c289  .byte $8Ø $4Ø $2Ø $1Ø $Ø8 $Ø4 $Ø2 $Ø1

=27
. c291  :
. c291  *** BYTEZEIGER SETZEN ***
. c291  :
. c291  98       tya          Addiert
. c292  18       clc          ... Spalte und
. c293  7d Ø2 d8 adc $d8Ø2,x  ... Zeilenanfangsadresse (Low)
. c296  85 8c    sta $8c      Ergebnis gleich Bytezeiger (Low)
. c298  bd 1b d8 lda $d81b,x  Holt Zeilenanfangsadresse (High)
. c29b  29 Ø3    and #$Ø3     Isoliert Bit Ø-1
. c29d  69 ØØ    adc #$ØØ     ... und addiert Carry
. c29f  Ø6 8c    asl $8c      Multipliziert
. c2a1  2a       rol          ... ganze
. c2a2  Ø6 8c    asl $8c      ... Adresse
. c2a4  2a       rol          ... mit
. c2a5  Ø6 8c    asl $8c      ... 8
. c2a7  2a       rol          Addiert Adresse (High)
. c2a8  Ø9 2Ø    ora #$2Ø     ... mit Basisadresse des Bitspeichers
. c2aa  85 8d    sta $8d      Ergebnis gleich Bytezeiger (High)
. c2ac  6Ø       rts          ENDE

. c2ad  :
. c2ad  *** ZEILE und SPALTE NACH (X-REG., Y-REG.) ***
. c2ad  :
. c2ad  ad ae Ø2 lda $Ø2ae    Holt X-Koordinate (High)
. c2bØ  4a       lsr          X-Koordinate größer als 511 ?
. c2b1  dØ 1e    bne $c2d1    Ja, dann nicht im Grafikbereich
. c2b3  ad ad Ø2 lda $Ø2ad    Holt X-Koordinate (Low)
. c2b6  6a       ror          Dividiert X-Koordinate (Low)
. c2b7  4a       lsr          ... durch vier
. c2b8  24 83    bit $83      Multicolor ?
. c2ba  3Ø Ø1    bmi $c2bd    Ja, dann $c2bd
. c2bc  4a       lsr          X-Koordinate (Low) durch 2 dividieren
. c2bd  a8       tay          Ergebnis
. c2be  cØ 28    cpy #$28     ... größer 39 ?
. c2cØ  bØ Øf    bcs $c2d1    Ja, dann Spaltennummer unzulässig
. c2c2  ad bØ Ø2 lda $Ø2bØ    Holt Y-Koordinate (High)
. c2c5  dØ Øa    bne $c2d1    Ungleich Null ?, dann nicht im Grafikbereich
. c2c7  ad af Ø2 lda $Ø2af    Holt Y-Koordinate (Low)
. c2ca  4a       lsr          Dividiert
```

```
. c2cb  4a       lsr          ... Y-Koordinate (Low)
. c2cc  4a       lsr          ... durch
. c2cd  aa       tax          ... 8
. c2ce  c5 88    cmp $88      Vergleicht Y-Koordinate mit Grafik-Zeilenzahl
. c2dØ  6Ø       rts          ENDE

. c2d1  38       sec
. c2d2  6Ø       rts          ENDE

. c2d3  :
. c2d3  *** SCALE AUSFÜHREN ***
. c2d3  :
. c2d3  ad e6 Ø2 lda $Ø2e6    Holt Scale-Flag
. c2d6  fØ 17    beq $c2ef    Scale-Flag nicht gesetzt ?, dann $c2ef
. c2d8  a5 87    lda $87      Holt Spaltenzahl
. c2da  2Ø df c2 jsr $c2df    Akku in X-Koordinaten umrechnen
. c2dd  a5 88    lda $88      Holt Zeilenzahl

      AKKU IN X-KOORDINATEN UMRECHNEN

. c2df  Øa       asl          Verdoppelt Akku
. c2eØ  a8       tay          In High-Byte umwandeln
. c2e1  a9 ØØ    lda #$ØØ     Low-Byte gleich Null
. c2e3  2Ø 37 c3 jsr $c337    (Akku,Y-Reg) := (Akku,Y-Reg)*KO(X-Reg)/65536
. c2e6  9d ad Ø2 sta $Ø2ad,x  Speichert
. c2e9  98       tya          ... neue
. c2ea  e8       inx          ... Koordinaten...
. c2eb  9d ad Ø2 sta $Ø2ad,x
. c2ee  e8       inx
. c2ef  6Ø       rts          ENDE

. c2fØ  :
. c2fØ  *** (AKKU,Y-REG.) - (AKKU,Y-REG.) +/- KO(X-REG.) ***
. c2fØ  :
. c2fØ  9Ø Ø7    bcc $c2f9    Carry gleich Ø ?, dann addieren
. c2f2  bØ 14    bcs $c3Ø8    Carry gleich 1 ?, dann subtrahieren
. c2f4  :
. c2f4  *** (AKKU,Y-REG.) - KO(Y-REG.) +/- KO(X-REG.) ***
. c2f4  :
. c2f4  bØ Øf    bcs $c3Ø5    Carry gleich 1 ?, dann subtrahieren
. c2f6  2Ø 18 c3 jsr $c318    (Akku, Y-Reg.) := KO(Y-Reg.)
. c2f9  18       clc          Addiert
. c2fa  7d ad Ø2 adc $Ø2ad,x  ... KO(X-Reg.)...
. c2fd  48       pha
. c2fe  98       tya
. c2ff  7d ae Ø2 adc $Ø2ae,x
. c3Ø2  a8       tay
. c3Ø3  68       pla
. c3Ø4  6Ø       rts          ENDE

. c3Ø5  :
. c3Ø5  *** (AKKU,Y-REG.) := KO(Y-REG.) - KO(X-Reg.) ***
```

```
. c305  :
. c305  20 18 c3 jsr $c318    (Akku, Y-Reg.) := KO(Y-Reg.)
. c308  38       sec          Subtrahiert
. c309  fd ad 02 sbc $02ad,x  ... KO(X-Reg.)...
. c30c  85 57    sta $57
. c30e  98       tya
. c30f  fd ae 02 sbc $02ae,x
. c312  a8       tay
. c313  08       php
. c314  a5 57    lda $57
. c316  28       plp
. c317  60       rts          ENDE

. c318  :
. c318  *** (AKKU,Y-REG.) := KO(Y-REG.) ***
. c318  :
. c318  b9 ad 02 lda $02ad,y  Holt KO(Y-Reg.) (Low)
. c31b  48       pha
. c31c  b9 ae 02 lda $02ae,y  Holt KO(Y-Reg.) (High)
. c31f  a8       tay
. c320  68       pla
. c321  60       rts          ENDE

. c322  :
. c322  *** (AKKU,Y-REG.) := ABS(KO(Y-REG.) - KO(X-REG.)) ***
. c322  :
. c322  20 05 c3 jsr $c305    (Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
. c325  10 0f    bpl $c336    Größergleich Null ?, dann fertig
. c327  08       php          Vorzeichen auf Stack
. c328  18       clc          Akku
. c329  49 ff    eor #$ff     ... und
. c32b  69 01    adc #$01     ... Y-Reg.
. c32d  48       pha          ... invertieren...
. c32e  98       tya
. c32f  49 ff    eor #$ff
. c331  69 00    adc #$00
. c333  a8       tay
. c334  68       pla
. c335  28       plp          Holt Vorzeichen
. c336  60       rts          ENDE

. c337  :
. c337  *** (AKKU,Y-REG.) - (AKKU,Y-REG.) * KO(X-REG.) / 65536 ***
. c337  :
. c337  84 8e    sty $8e      Schreibt
. c339  85 8f    sta $8f      ... Multiplikator
. c33b  bd ad 02 lda $02ad,x  Holt
. c33e  bc ae 02 ldy $02ae,x  ... X-Koordinate
. c341  08       php          Vorzeichen auf Stack
. c342  20 25 c3 jsr $c325    Invertieren, wenn negativ
. c345  9d ad 02 sta $02ad,x  Schreibt
. c348  98       tya          ... X-Koordinate...
```

```
. c349  9d ae 02 sta $02ae,x
. c34c  a9 00    lda #$00      Löscht
. c34e  8d ef 02 sta $02ef     ... Ergebnis
. c351  a0 10    ldy #$10      Holt Bitzähler
. c353  46 8e    lsr $8e       Shiftet Multiplikator
. c355  66 8f    ror $8f       ... nach rechts
. c357  90 0f    bcc $c368     Bit gleich Null ?, dann $c368
. c359  18       clc           Addiert
. c35a  7d ad 02 adc $02ad,x   ... Koordinate
. c35d  48       pha           ... zum
. c35e  ad ef 02 lda $02ef     ... Ergebnis...
. c361  7d ae 02 adc $02ae,x
. c364  8d ef 02 sta $02ef
. c367  68       pla
. c368  4e ef 02 lsr $02ef     Shiftet Ergebnis
. c36b  6a       ror           ... nach rechts
. c36c  88       dey           Erniedrigt Bitzähler um 1
. c36d  d0 e4    bne $c353     Bitzähler ungleich Null ?, dann $c353
. c36f  69 00    adc #$00      Addiert
. c371  ac ef 02 ldy $02ef     ... letztes Bit...
. c374  90 01    bcc $c377
. c376  c8       iny
. c377  28       plp           Holt Vorzeichen vom Stack
. c378  4c 25 c3 jmp $c325     Invertieren, wenn negativ

. c37b  :
. c37b  *** KO(0) := KO (4) und KO(2) := KO (6) ***
. c37b  :
. c37b  a0 00    ldy #$00
. c37d  20 82 c3 jsr $c382     KO(Y-Reg.) := KO(Y-Reg. + 4)
. c380  a0 02    ldy #$02
. c382  b9 b1 02 lda $02b1,y
. c385  99 ad 02 sta $02ad,y
. c388  b9 b2 02 lda $02b2,y
. c38b  99 ae 02 sta $02ae,y
. c38e  60       rts           ENDE

. c38f  :
. c38f  *** ADRESSE ÜBERNEHMEN ***
. c38f  :
. c38f  20 79 04 jsr $0479     CHRGOT-Routine
. c392  f0 0c    beq $c3a0     Folgt Trennzeichen ?, dann $c3a0
. c394  20 91 94 jsr $9491     (CHKCOM) Prüft ob Komma folgt
. c397  c9 2c    cmp #$2c      Folgt 2. Komma ?
. c399  f0 05    beq $c3a0     Ja, dann $c3a0
. c39b  20 e1 9d jsr $9de1     (FRMEVL) und (GETADR)
. c39e  38       sec           Setzt Flag für 'Eingabe'
. c39f  60       rts           ENDE

. c3a0  a9 00    lda #$00      Keine Eingabe,
. c3a2  a8       tay           ... dann Ersatzwert gleich Null
. c3a3  18       clc           Setzt Flag für 'Ersatzwert'
```

```
. c3a4  60       rts          ENDE

. c3a5  :
. c3a5  *** BYTE ÜBERNEHMEN ***
. c3a5  :
. c3a5  a2 00    ldx #$00     Ersatzwert Null, bei fehlender Eingabe
. c3a7  20 79 04 jsr $0479    CHRGOT-Routine
. c3aa  f0 f8    beq $c3a4    Folgt Trennzeichen ?, dann fertig
. c3ac  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. c3af  c9 2c    cmp #$2c     Folgt 2. Komma ?
. c3b1  f0 f1    beq $c3a4    Ja, dann fertig
. c3b3  4c 84 9d jmp $9d84    (FRMEVL) und (GETADR)

. c3b3  :
. c3b3  *** FARB-PARAMETER ÜBERNEHMEN ***
. c3b3  :
. c3b6  20 bf c7 jsr $c7bf    Prüft ob Grafik eingeschaltet ist
. c3b9  a2 01    ldx #$01     Ersatzwert 1, bei fehlender Eingabe
. c3bb  20 79 04 jsr $0479    CHRGOT-Routine
. c3be  f0 13    beq $c3d3    Folgt Trennzeichen ?, dann $c3d3
. c3c0  c9 2c    cmp #$2c     Folgt Komma ?
. c3c2  f0 0f    beq $c3d3    Ja, dann $c3d3
. c3c4  20 84 9d jsr $9d84    (FRMEVL) und (GETADR)
. c3c7  e0 04    cpx #$04     Größer 3 ?
. c3c9  b0 0b    bcs $c3d6    Ja, dann Fehler
. c3cb  e0 02    cpx #$02     Größer 1 ?
. c3cd  24 83    bit $83      Multicolor ?
. c3cf  30 02    bmi $c3d3    Ja, dann $c3d3
. c3d1  b0 03    bcs $c3d6    Fehler, wenn Parameter größer 1
. c3d3  86 84    stx $84      Speichert Farbparameter
. c3d5  60       rts          ENDE

. c3d6  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. c3d9  :
. c3d9  *** KOORDINATEN ÜBERNEHMEN ***
. c3d9  :
. c3d9  20 79 04 jsr $0479    CHRGOT-Routine
. c3dc  f0 07    beq $c3e5    Folgt Trennzeichen ?, dann $c3e5
. c3de  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
. c3e1  c9 2c    cmp #$2c     Folgt 2. Komma ?
. c3e3  d0 12    bne $c3f7    Nein, dann $c3f7
. c3e5  a0 00    ldy #$00
. c3e7  b9 ad 02 lda $02ad,y  Ersatzwert
. c3ea  9d ad 02 sta $02ad,x  ... KO(0),
. c3ed  e8       inx          ... bei
. c3ee  c8       iny          ... fehlender
. c3ef  c0 04    cpy #$04     ... Eingabe...
. c3f1  d0 f4    bne $c3e7
. c3f3  60       rts          ENDE

. c3f4  20 91 94 jsr $9491    (CHKCOM) Prüft ob Komma folgt
```

```
. c3f7  8e f0 02 stx $02f0    Schreibt Koordinaten-Offset
. c3fa  20 8f c4 jsr $c48f    Holt 1. Koordinate
. c3fd  20 79 04 jsr $0479    CHRGOT-Routine
. c400  c9 2c    cmp #$2c     Rechtwinklige Koordinaten ?
. c402  f0 56    beq $c45a    Ja, dann $c45a
. c404  c9 3b    cmp #$3b     Polarkoordinaten ?
. c406  f0 03    beq $c40b    Ja, dann $c40b
. c408  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. c40b  :
. c40b  *** POLARKOORDINATEN ***
. c40b  :
. c40b  20 73 04 jsr $0473    CHRGET-Routine
. c40e  20 e1 9d jsr $9de1    (FRMEVL) und (GETADR)
. c411  85 80    sta $80      Vertauscht
. c413  98       tya          ... Akku
. c414  a4 80    ldy $80      ... und Y-Reg.
. c416  20 59 bc jsr $bc59    Berechnet Sinus und Cosinus
. c419  ae f0 02 ldx $02f0    Holt Koordinaten-Offset
. c41c  bd ad 02 lda $02ad,x  Koordinaten
. c41f  9d af 02 sta $02af,x  ... umspeichern
. c422  bd ae 02 lda $02ae,x
. c425  9d b0 02 sta $02b0,x
. c428  20 d3 c2 jsr $c2d3    Führt SCALE aus
. c42b  a9 0e    lda #$0e     SIN/COS-Flags
. c42d  8d f1 02 sta $02f1    ... speichern
. c430  18       clc
. c431  ae f0 02 ldx $02f0    Holt Koordinaten-Offset
. c434  20 b0 bc jsr $bcb0    (Akku, Y-Reg.) := KO(X-Reg.) * COS/SIN
. c437  9d ad 02 sta $02ad,x  Speichert
. c43a  98       tya          ... Ergebnis...
. c43b  9d ae 02 sta $02ae,x
. c43e  a0 00    ldy #$00
. c440  4e f1 02 lsr $02f1    Holt nächstes Flag ins Carry
. c443  90 02    bcc $c447    Flag nicht gesetzt ?, dann $c447
. c445  a0 02    ldy #$02
. c447  20 f4 c2 jsr $c2f4    (Akku, Y-Reg.) := KO(Y-Reg.) +- KO(X-Reg.)
. c44a  9d ad 02 sta $02ad,x  Speichert
. c44d  98       tya          ... rechtwinklige
. c44e  9d ae 02 sta $02ae,x  ... Koordinaten...
. c451  e8       inx
. c452  e8       inx
. c453  4e f1 02 lsr $02f1
. c456  d0 dc    bne $c434    Springt nach dem 1. Durchlauf nach $c434
. c458  18       clc
. c459  60       rts          ENDE

. c45a  :
. c45a  *** RECHTWINKLIGE KOORDINATEN ***
. c45a  :
. c45a  20 73 04 jsr $0473    CHRGET-Routine
. c45d  ee f0 02 inc $02f0    Erhöht Koordinaten-Offset
```

```
. c46Ø  ee fØ Ø2 inc $Ø2fØ     ... um 2
. c463  2Ø 8f c4 jsr $c48f     Holt Koordinaten
. c466  ae fØ Ø2 ldx $Ø2fØ     Holt Koordinaten-Offset
. c469  ca       dex
. c46a  ca       dex
. c46b  2Ø d3 c2 jsr $c2d3     Führt SCALE aus
. c46e  aØ Ø2    ldy #$Ø2
. c47Ø  ae fØ Ø2 ldx $Ø2fØ
. c473  e8       inx
. c474  e8       inx
. c475  ca       dex
. c476  ca       dex
. c477  4e f1 Ø2 lsr $Ø2f1     Relative Koordinaten ?
. c47a  9Ø Øa    bcc $c486     Nein, dann $c486
. c47c  2Ø f6 c2 jsr $c2f6     (Akku, Y-Reg.) := KO(Y-Reg.) + KO(X-Reg.)
. c47f  9d ad Ø2 sta $Ø2ad,x   Speichert
. c482  98       tya           ... neue
. c483  9d ae Ø2 sta $Ø2ae,x   ... Koordinaten...
. c486  aØ ØØ    ldy #$ØØ
. c488  ec fØ Ø2 cpx $Ø2fØ
. c48b  fØ e8    beq $c475     Springt nach dem 1. Durchlauf nach $c475
. c48d  18       clc
. c48e  6Ø       rts           ENDE

. c48f  :
. c48f  *** KOORDINATEN HOLEN ***
. c48f  :
. c48f  2Ø 79 Ø4 jsr $Ø479     CHRGOT-Routine
. c492  c9 aa    cmp #$aa      Zeichen gleich '+'-Token
. c494  fØ Ø5    beq $c49b     Ja, dann $c49b
. c496  c9 ab    cmp #$ab      Zeichen gleich '-'-Token
. c498  fØ Ø1    beq $c49b     Ja, dann $c49b
. c49a  18       clc
. c49b  2e f1 Ø2 rol $Ø2f1     Flag für 'Relative Koordinaten'
. c49e  2Ø 14 93 jsr $9314     (FRMNUM)
. c4a1  2Ø e8 9d jsr $9de8     (GETADR)
. c4a4  ae fØ Ø2 ldx $Ø2fØ     Holt Koordinaten-Offset
. c4a7  9d ae Ø2 sta $Ø2ae,x   Speichert
. c4aa  98       tya           ... Wert der
. c4ab  9d ad Ø2 sta $Ø2ad,x   ... Koordinaten
. c4ae  6Ø       rts           ENDE

. c4af  :
. c4af  *** KONSTANTEN ***
. c4af  :
. c4af  $ff $aa $55 $ØØ       Konstanten für Multicolor

     SINUS-TABELLE

. c4b3  $ØØ $ØØ                   Ø = 65536 * Ø.ØØØØ   SIN  Ø Grad
. c4b5  $2c $71                11377 = 65536 * Ø.1736   SIN 1Ø Grad
. c4b7  $57 $8d                22413 = 65536 * Ø.342Ø   SIN 2Ø Grad
```

```
. c4b9  $80 $00              32768 = 65536 * 0.5000   SIN 30 Grad
. c4bb  $a4 $8f              42127 = 65536 * 0.6428   SIN 40 Grad
. c4bd  $c4 $19              50201 = 65536 * 0.7660   SIN 50 Grad
. c4bf  $dd $b2              56754 = 65536 * 0.8660   SIN 60 Grad
. c4c1  $f0 $90              61584 = 65536 * 0.9397   SIN 70 Grad
. c4c3  $fc $1c              64540 = 65536 * 0.9848   SIN 80 Grad
. c4c5  $ff $ff              65535 = 65536 * 1.0000   SIN 90 Grad

     INTERPOLATIONSTABELLE

. c4c7  $04 $72               1138
. c4c9  $04 $50               1104
. c4cb  $04 $0b               1035
. c4cd  $03 $a8                936
. c4cf  $03 $28                808
. c4d1  $02 $90                656
. c4d3  $01 $e3                483
. c4d5  $01 $28                296
. c4d7  $00 $63                 99

. c4d9  :
. c4d9  *** D R A W ***
. c4d9  :
. c4d9  20 bf c7 jsr $c7bf    Prüft ob Grafik eingeschaltet ist
. c4dc  20 79 04 jsr $0479    CHRGOT-Routine
. c4df  f0 12    beq $c4f3    Folgt Trennzeichen ?, dann fertig
. c4e1  a2 01    ldx #$01
. c4e3  c9 a4    cmp #$a4     Zeichen gleich 'TO'-Token
. c4e5  20 be c3 jsr $c3be    Holt Farbparameter, falls vorhanden
. c4e8  20 79 04 jsr $0479    CHRGOT-Routine
. c4eb  c9 2c    cmp #$2c     Zeichen gleich Komma (,) ?
. c4ed  f0 05    beq $c4f4    Ja, dann $c4f4
. c4ef  c9 a4    cmp #$a4     Zeichen gleich 'TO'-Token
. c4f1  f0 01    beq $c4f4    Ja, dann $c4f4
. c4f3  60       rts          ENDE

. c4f4  48       pha          Zeichen auf Stack
. c4f5  20 73 04 jsr $0473    CHRGET-Routine
. c4f8  a2 04    ldx #$04
. c4fa  20 f7 c3 jsr $c3f7    Holt Endkoordinaten
. c4fd  68       pla          Holt Zeichen vom Stack
. c4fe  10 06    bpl $c506    Zeichen kein 'TO'-Token ?, dann $c506
. c500  20 da c0 jsr $c0da    Zeichnet Strecke
. c503  4c e8 c4 jmp $c4e8    Weiter im Befehl

. c506  20 7b c3 jsr $c37b    Setzt Anfangskoordinaten gleich Endkoordinaten
. c509  20 a5 c1 jsr $c1a5    Setzt Punkt
. c50c  4c e8 c4 jmp $c4e8    Weiter im Befehl

. c50f  :
. c50f  *** L O C A T E ***
. c50f  :
```

```
. c50f  20 bf c7 jsr $c7bf    Prüft ob Grafik eingeschaltet ist
. c512  a2 04    ldx #$04
. c514  20 f7 c3 jsr $c3f7    Holt Koordinaten
. c517  4c 7b c3 jmp $c37b    Koordinaten als KO(0) speichern

. c51a  :
. c51a  *** C O L O R ***
. c51a  :
. c51a  20 84 9d jsr $9d84    Holt Farbparameter
. c51d  e0 05    cpx #$05     Farbparameter größer 4 ?
. c51f  b0 43    bcs $c564    Ja, dann Fehler
. c521  86 7e    stx $7e      Speichert Farbparameter
. c523  20 d8 9d jsr $9dd8    Holt Farbcode
. c526  ca       dex
. c527  e0 10    cpx #$10     Farbcode größer 16 ?
. c529  b0 39    bcs $c564    Ja, dann Fehler
. c52b  86 7f    stx $7f      Schreibt Farbcode
. c52d  a2 07    ldx #$07     Holt Ersatzwert für Helligkeit
. c52f  20 a7 c3 jsr $c3a7    Holt Helligkeit
. c532  e0 08    cpx #$08     Helligkeit größer 7 ?
. c534  b0 2e    bcs $c564    Ja, dann Fehler
. c536  8a       txa          Schiebt
. c537  0a       asl          ... Helligkeit
. c538  0a       asl          ... ins
. c539  0a       asl          ... obere
. c53a  0a       asl          ... Halbbyte
. c53b  05 7f    ora $7f      Ordnet Farbcode ein
. c53d  a6 7e    ldx $7e      Holt Farbparameter
. c53f  e0 01    cpx #$01     Farbparameter gleich 1 ?
. c541  f0 07    beq $c54a    Ja, dann $c54a
. c543  b0 0c    bcs $c551    Farbparameter größer 1 ? ,dann $c551
. c545  8d 15 ff sta $ff15    Setzt Hintergrundfarbe
. c548  d0 19    bne $c563    Springt immer nach $c563

. c54a  85 86    sta $86      Setzt Farbe für Grafik-Punkte
. c54c  8d 3b 05 sta $053b    Setzt Zeichenfarbe
. c54f  f0 12    beq $c563    Springt immer nach $c563

. c551  e0 03    cpx #$03
. c553  f0 06    beq $c55b    Farbparameter gleich 3 ?, dann $c55b
. c555  b0 09    bcs $c560    Farbparameter gleich 4 ?, dann $c560
. c557  85 85    sta $85      Setzt Multicolor 1
. c559  d0 08    bne $c563    Springt immer nach $c563

. c55b  8d 16 ff sta $ff16    Setzt Multicolor 2
. c55e  f0 03    beq $c563    Springt immer nach $c563

. c560  8d 19 ff sta $ff19    Setzt Randfarbe
. c563  60       rts          ENDE

. c564  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben
```

```
. c567  :
. c567  *** S C N C L R ***
. c567  :
. c567  a5 83     lda $83       Grafikmodus ?
. c569  d0 05     bne $c570     Ja, dann $c570
. c56b  a9 93     lda #$93      Holt 'CLR'
. c56d  4c d2 ff  jmp $ffd2     (BSOUT) Zeichen ausgeben

. c570  29 40     and #$40      Mit Text ?
. c572  f0 0b     beq $c57f     Nein, dann $c57f
. c574  20 6b c5  jsr $c56b     Löscht Bildschirm
. c577  a2 14     ldx #$14      Holt Zeile
. c579  a0 00     ldy #$00      Holt Spalte
. c57b  18        clc
. c57c  20 f0 ff  jsr $fff0     (PLOT) Holt Cursorspalte ins Y-Reg.
. c57f  a9 00     lda #$00
. c581  a0 20     ldy #$20
. c583  a2 20     ldx #$20      Holt Anzahl der Pages (32 Pages = 8 Kbyte)
. c585  20 a7 c5  jsr $c5a7     Löscht Bitspeicher
. c588  20 38 c2  jsr $c238     Akku := Zeichen-/Hintergrundfarbe
. c58b  a0 1c     ldy #$1c
. c58d  a2 04     ldx #$04      Holt Anzahl der Pages (4 Pages = 1 Kbyte)
. c58f  20 a7 c5  jsr $c5a7     Löscht Farbspeicher
. c592  20 4e c2  jsr $c24e     Akku := Helligkeit
. c595  a0 18     ldy #$18
. c597  a2 04     ldx #$04
. c599  20 a7 c5  jsr $c5a7     Löscht Helligkeitsspeicher
. c59c  a9 00     lda #$00
. c59e  a2 03     ldx #$03
. c5a0  9d ad 02  sta $02ad,x   Koordinaten
. c5a3  ca        dex           ... (X-Reg., Y-Reg.) := (0,0)...
. c5a4  10 fa     bpl $c5a0
. c5a6  60        rts           ENDE

. c5a7  84 8d     sty $8d       Schreibt
. c5a9  a0 00     ldy #$00      ... Farbzeiger- Zeiger
. c5ab  84 8c     sty $8c       ... in Bit-Map

     LÖSCHROUTINE

. c5ad  91 8c     sta ($8c),y
. c5af  88        dey
. c5b0  d0 fb     bne $c5ad
. c5b2  e6 8d     inc $8d
. c5b4  ca        dex
. c5b5  d0 f6     bne $c5ad
. c5b7  60        rts           ENDE

. c5b8  :
. c5b8  *** S C A L E ***
. c5b8  :
. c5b8  20 84 9d  jsr $9d84     (GETBYT) Holt SCALE-Byte
```

```
. c5bb  e0 02    cpx #$02     Byte größer als 1 ?
. c5bd  b0 a5    bcs $c564    Ja, dann Fehler
. c5bf  8e e6 02 stx $02e6    Setzt SCALE-Flag
. c5c2  60       rts          ENDE

. c5c3  :
. c5c3  *** G R A P H I C ***
. c5c3  :
. c5c3  c9 9c    cmp #$9c     Folgt 'CLR'-Token ?
. c5c5  d0 0a    bne $c5d1    Nein, dann $c5d1
. c5c7  20 38 c7 jsr $c738    Speicheraufteilung normalisieren
. c5ca  20 73 04 jsr $0473    CHRGET-Routine
. c5cd  a9 00    lda #$00
. c5cf  f0 0a    beq $c5db    Springt immer nach $c5db

. c5d1  20 84 9d jsr $9d84    (GETBYT) Holt Grafikmodus
. c5d4  e0 05    cpx #$05     Modus größer als 4 ?
. c5d6  b0 15    bcs $c5ed    Ja, dann Fehler
. c5d8  bd 37 c6 lda $c637,x  Holt Grafik-Modus aus Tabelle
. c5db  c5 83    cmp $83      Bereits vorhanden ?
. c5dd  f0 4b    beq $c62a    Ja, dann $c62a
. c5df  85 83    sta $83      Speichert neuen Grafik-Modus
. c5e1  aa       tax          Modus 0 ?
. c5e2  d0 0c    bne $c5f0    Nein, dann $c5f0
. c5e4  20 c9 c7 jsr $c7c9    Grafik abschalten
. c5e7  a9 28    lda #$28     Holt Spalte
. c5e9  a2 19    ldx #$19     Holt Zeile
. c5eb  d0 39    bne $c626    Springt immer nach $c626

. c5ed  4c 1c 99 jmp $991c    'ILLEGAL QUANTITY ERROR' ausgeben

. c5f0  20 3c c6 jsr $c63c    Grafik-Speicher einrichten
. c5f3  ad 06 ff lda $ff06    Setzt
. c5f6  09 20    ora #$20     ... Bitspeicher-
. c5f8  8d 06 ff sta $ff06    ... Freigabe-Bit
. c5fb  ad 07 ff lda $ff07    Multicolor-Freigabe-Bit
. c5fe  29 ef    and #$ef     ... löschen
. c600  24 83    bit $83      Multicolor ?
. c602  10 02    bpl $c606    Nein, dann $c606
. c604  09 10    ora #$10     Setzt
. c606  8d 07 ff sta $ff07    ... Multicolor-Freigabe-Bit
. c609  ad 12 ff lda $ff12
. c60c  29 c3    and #$c3
. c60e  09 08    ora #$08
. c610  8d 12 ff sta $ff12
. c613  ad 14 ff lda $ff14    Schaltet
. c616  29 03    and #$03     ... Bit-Speicherbereich
. c618  09 18    ora #$18     ... ein...
. c61a  8d 14 ff sta $ff14
. c61d  a9 28    lda #$28     Holt Spaltenzahl (40)
. c61f  a2 19    ldx #$19     Holt Zeilenzahl (25)
. c621  24 83    bit $83      Multicolor ?
```

```
. c623  10 01     bpl $c626     Nein, dann $c626
. c625  4a        lsr           Halbiert Spaltenzahl
. c626  85 87     sta $87       Schreibt Grafik-Spalten
. c628  86 88     stx $88       Schreibt Grafik-Zeilen
. c62a  20 a5 c3  jsr $c3a5     Holt SCNCLR-Flag
. c62d  8a        txa
. c62e  4a        lsr
. c62f  d0 bc     bne $c5ed     Größer als 1 ?, dann Fehler
. c631  90 03     bcc $c636     Gleich Null ?, dann fertig
. c633  4c 67 c5  jmp $c567     SCNCLR

. c636  60        rts           ENDE

. c637  :
. c637  *** TABELLE DER GRAFIK-MODI ***
. c637  :
. c637  $00                     Text-Modus
. c638  $20                     Hochauflösende Grafik
. c639  $60                     Hochauflösende Grafik mit Text
. c63a  $a0                     Multicolor-Grafik
. c63b  $e0                     Multicolor-Grafik mit Text
. c63c  a5 75     lda $75       Grafik-Speicher-Flag gleich Null ?
. c63e  f0 01     beq $c641     Ja, dann $c641
. c640  60                      ENDE

. c641  a5 38     lda $38       Basic-Obergrenze
. c643  c9 40     cmp #$40      ... unter $4000 ?
. c645  b0 34     bcs $c67b     Nein, dann $c67b
. c647  20 54 a9  jsr $a954     GARBAGE COLLECT
. c64a  20 6b c8  jsr $c86b     (X-Reg., Y-Reg.) := Länge des Stringbereichs
. c64d  8a        txa           Addiert
. c64e  18        clc           ... Variablenende...
. c64f  65 31     adc $31
. c651  98        tya
. c652  65 32     adc $32
. c654  c9 18     cmp #$18      Ergebnis größer 6 Kbyte ?
. c656  b0 20     bcs $c678     Ja, dann Fehler
. c658  c6 75     dec $75
. c65a  a9 00     lda #$00
. c65c  85 22     sta $22
. c65e  a9 18     lda #$18
. c660  85 23     sta $23
. c662  20 f0 c7  jsr $c7f0     Verschiebt Strings
. c665  a5 22     lda $22       Kopiert neue
. c667  85 33     sta $33       ... Stringbereich-Untergrenze
. c669  a5 23     lda $23       ... in
. c66b  85 34     sta $34       ... String-Anfangszeiger
. c66d  a9 00     lda #$00      Setzt
. c66f  85 37     sta $37       ... neue
. c671  a9 18     lda #$18      ... Basic-Obergrenze
. c673  85 38     sta $38       ... auf $1800
. c675  4c 25 c8  jmp $c825     Korrigiert R-Zeiger
```

```
. c678  4c 81 86 jmp $8681    'OUT OF MEMORY ERROR' ausgeben

. c67b  20 54 a9 jsr $a954    GARBAGE COLLECT

. c67e  a4 31    ldy $31      Holt Variablenende (Low)
. c680  84 5f    sty $5f      Schreibt Transportzeiger (Low)
. c682  a5 32    lda $32      Holt Variablenende (High)
. c684  18       clc          Addiert
. c685  69 30    adc #$30     ... 12 Kbyte
. c687  b0 ef    bcs $c678    Überlauf ?, dann Fehler
. c689  85 60    sta $60      Schreibt Transportzeiger (High)
. c68b  c5 34    cmp $34      Transportzeiger
. c68d  90 06    bcc $c695    ... kleiner
. c68f  d0 e7    bne $c678    ... als
. c691  c4 33    cpy $33      ... String-Untergrenze ?
. c693  b0 e3    bcs $c678    Nein, dann Fehler
. c695  c6 75    dec $75      Setzt Flag für Grafik-Speicher
. c697  a9 00    lda #$00
. c699  85 4e    sta $4e
. c69b  a9 30    lda #$30
. c69d  85 4f    sta $4f
. c69f  20 77 c8 jsr $c877    Korrigiert R-Zeiger der Strings
. c6a2  a5 5f    lda $5f      Kopiert
. c6a4  85 22    sta $22      ... Zielbereich-Ende
. c6a6  a5 60    lda $60      ... nach
. c6a8  85 23    sta $23      ... $22-$23
. c6aa  a6 31    ldx $31      Kopiert
. c6ac  86 24    stx $24      ... Quellbereichs-Ende
. c6ae  a5 32    lda $32      ... nach
. c6b0  85 25    sta $25      ... $24-$25
. c6b2  38       sec          Subtrahiert
. c6b3  e9 10    sbc #$10     ... Programmanfang
. c6b5  a8       tay          Ergebnis gleich Länge des Blocks
. c6b6  20 f8 c7 jsr $c7f8    Schiebt Block nach oben
. c6b9  18       clc          Setzt
. c6ba  a5 32    lda $32      ... Variablenzeiger
. c6bc  69 30    adc #$30     ... um 12 Kbyte
. c6be  85 32    sta $32      ... nach oben
. c6c0  a5 30    lda $30      Setzt Array-Anfangszeiger
. c6c2  69 30    adc #$30     ... um 12 Kbyte
. c6c4  85 30    sta $30      ... nach oben
. c6c6  a5 2e    lda $2e      Setzt Variablen-Anfangszeiger
. c6c8  69 30    adc #$30     ... um 12 Kbyte
. c6ca  85 2e    sta $2e      ... nach oben
. c6cc  a5 2c    lda $2c      Setzt Programm-Anfangszeiger
. c6ce  69 30    adc #$30     ... um 12 Kbyte
. c6d0  85 2c    sta $2c      ... nach oben
. c6d2  a5 42    lda $42      Setzt DATA-Zeiger
. c6d4  69 30    adc #$30     ... um 12 Kbyte
. c6d6  85 42    sta $42      ... nach oben
. c6d8  20 18 88 jsr $8818    Erzeugt Linkadresse
```

```
. c6db  20 4b 88 jsr $884b    Setzt Variablenanfang
. c6de  24 81    bit $81      Direktmodus ?
. c6e0  10 2d    bpl $c70f    Ja, dann fertig
. c6e2  a2 30    ldx #$30
. c6e4  24 75    bit $75
. c6e6  30 02    bmi $c6ea
. c6e8  a2 d0    ldx #$d0     X-Reg := -$3000 (High)
. c6ea  8a       txa
. c6eb  18       clc          Zum
. c6ec  65 3c    adc $3c      ... Textpointer
. c6ee  85 3c    sta $3c      ... addieren
. c6f0  8a       txa
. c6f1  18       clc          Zur Adresse
. c6f2  6d 5c 02 adc $025c    ... für TRAP und CONT
. c6f5  8d 5c 02 sta $025c    ... addieren
. c6f8  8a       txa
. c6f9  18       clc          Erhöht
. c6fa  6d f6 04 adc $04f6    ... Adresse für TRAP
. c6fd  8d f6 04 sta $04f6    ... um 12 Kbyte
. c700  20 60 a7 jsr $a760    ($3d-$3e) := ($7c-$7d)
. c703  a5 3d    lda $3d      Ist
. c705  c9 b0    cmp #$b0     ... Basic-Stack
. c707  d0 07    bne $c710    ... leer ?...
. c709  a5 3e    lda $3e
. c70b  c9 07    cmp #$07
. c70d  d0 01    bne $c710    Nein, dann $c710
. c70f  60       rts          ENDE

. c710  a0 00    ldy #$00     Korrigiert
. c712  b1 3d    lda ($3d),y  ... Basic-Stack
. c714  c9 81    cmp #$81     'FOR'-Token ?
. c716  d0 0e    bne $c726    Nein, dann $c726
. c718  a0 02    ldy #$02     Holt Variablenzeiger (High)
. c71a  20 ad c7 jsr $c7ad    Korrigiert Adresse
. c71d  a0 10    ldy #$10     Holt Startadresse der Schleife (High)
. c71f  20 ad c7 jsr $c7ad    Korrigiert Adresse
. c722  a9 12    lda #$12     Holt Länge der FOR-Datensätze im Stack
. c724  d0 07    bne $c72d    Springt immer nach $c72d

. c726  a0 04    ldy #$04     Holt Rücksprungadresse (High)
. c728  20 ad c7 jsr $c7ad    Korrigiert Adresse
. c72b  a9 05    lda #$05     Holt Länge der GOSUB-Datensätze im Stack
. c72d  18       clc          Setzt
. c72e  65 3d    adc $3d      ... Stack-Zeiger
. c730  85 3d    sta $3d      ... auf
. c732  90 cf    bcc $c703    ... nächsten
. c734  e6 3e    inc $3e      ... Eintrag
. c736  d0 cb    bne $c703    ... und weitermachen

. c738  :
. c738  *** NORMALE SPEICHERAUFTEILUNG ***
. c738  :
```

```
. c738  a5 75     lda $75       Speicheraufteilung bereits normal ?
. c73a  d0 01     bne $c73d     Nein, dann $c73d
. c73c  60        rts           ENDE

. c73d  a0 00     ldy #$00      Initialisiert
. c73f  84 75     sty $75       ... Speicher-Flag
. c741  a5 38     lda $38       Basic-Obergrenze unter $8000 ?
. c743  30 24     bmi $c769     Nein, dann $c769
. c745  20 54 a9  jsr $a954     GARBAGE COLLECT
. c748  20 6b c8  jsr $c86b     (X-Reg., Y-Reg.) := Länge des Stringbereichs
. c74b  ad 33 05  lda $0533     Kopiert
. c74e  85 22     sta $22       ... Basic-Ende
. c750  ad 34 05  lda $0534     ... nach
. c753  85 23     sta $23       ... $22-$23
. c755  20 f0 c7  jsr $c7f0     Verschiebt Programm und Variablen
. c758  a2 01     ldx #$01
. c75a  bd 33 05  lda $0533,x   Basic-Ende
. c75d  95 37     sta $37,x     ... := Basic-Obergrenze-Zeiger
. c75f  b5 22     lda $22,x     Untergrenze des verschobenen Blocks
. c761  95 33     sta $33,x     ... := Stringbereich-Anfang
. c763  ca        dex
. c764  10 f4     bpl $c75a
. c766  4c 25 c8  jmp $c825     Korrigiert R-Zeiger
. c769  a0 00     ldy #$00      Initialisiert
. c76b  84 75     sty $75       ... Speicherflag,
. c76d  84 22     sty $22       ... Zielzeiger (Low),
. c76f  84 24     sty $24       ... Quellzeiger (Low)
. c771  a9 10     lda #$10      Setzt Zielzeiger
. c773  85 23     sta $23       ... auf $1000
. c775  a9 40     lda #$40      Setzt Quellzeiger
. c777  85 25     sta $25       ... auf $4000
. c779  20 bb 04  jsr $04bb     Holt Zeichen aus Quellbereich (LDA($24),Y)
. c77c  91 22     sta ($22),y   Kopiert Programm
. c77e  c8        iny           ... und Variablen
. c77f  d0 f8     bne $c779     ... in Zielbereich
. c781  e6 23     inc $23
. c783  e6 25     inc $25
. c785  a5 32     lda $32       Variablenende
. c787  c5 25     cmp $25       ... erreicht ?
. c789  b0 ee     bcs $c779     Nein, dann $c779
. c78b  a5 32     lda $32       Erniedrigt
. c78d  38        sec           ... Variablenende-
. c78e  e9 30     sbc #$30      ... Zeiger
. c790  85 32     sta $32       ... um 12 Kbyte
. c792  a5 2c     lda $2c       Erniedrigt
. c794  e9 30     sbc #$30      ... Programmanfangs-Zeiger
. c796  85 2c     sta $2c       ... um 12 Kbyte
. c798  a5 2e     lda $2e       Erniedrigt
. c79a  e9 30     sbc #$30      ... Variablenanfangs-Zeiger
. c79c  85 2e     sta $2e       ... um 12 Kbyte
. c79e  a5 30     lda $30       Erniedrigt
. c7a0  e9 30     sbc #$30      ... Array-Anfangszeiger
```

```
. c7a2  85 30     sta $30       ... um 12 Kbyte
. c7a4  a5 42     lda $42       Erniedrigt
. c7a6  e9 30     sbc #$30      ... DATA-Zeiger
. c7a8  85 42     sta $42       ... um 12 Kbyte
. c7aa  4c d8 c6  jmp $c6d8     Erzeugt Linkadress und setzt Variablenanfang, etc.

. c7ad  b1 3d     lda ($3d),y   Holt Adresse (High) aus Basic-Stack
. c7af  24 75     bit $75       Normale Speicheraufteilung ?
. c7b1  d0 06     bne $c7b9     Nein, dann $c7b9
. c7b3  38        sec           Erniedrigt
. c7b4  e9 30     sbc #$30      ... Adresse
. c7b6  91 3d     sta ($3d),y   ... um 12 Kbyte
. c7b8  60        rts           ENDE

. c7b9  18        clc           Erhöht
. c7ba  69 30     adc #$30      ... Adresse
. c7bc  91 3d     sta ($3d),y   ... um 12 Kbyte
. c7be  60        rts           ENDE

. c7bf  :
. c7bf  *** GRAFIK PRÜFEN ***
. c7bf  :
. c7bf  a5 75     lda $75       Grafikspeicher eingerichtet ?
. c7c1  f0 01     beq $c7c4     Nein, dann $c7c4
. c7c3  60        rts           ENDE

. c7c4  a2 23     ldx #$23      Holt Fehlernummer 35 ('NO GRAPHICS AREA')
. c7c6  4c 83 86  jmp $8683     Fehlerausgabe

. c7c9  :
. c7c9  *** GRAFIK ABSCHALTEN ***
. c7c9  :
. c7c9  ad 06 ff  lda $ff06     Löscht
. c7cc  29 df     and #$df      ... Bitspeicher-
. c7ce  8d 06 ff  sta $ff06     ... Freigabe-Bit
. c7d1  ad 07 ff  lda $ff07     Löscht
. c7d4  29 ef     and #$ef      ... Multicolor-
. c7d6  8d 07 ff  sta $ff07     ... Freigabe-Bit
. c7d9  ad 14 ff  lda $ff14     Schaltet
. c7dc  29 07     and #$07      ... normalen
. c7de  09 08     ora #$08      ... Bildschirm-Speicherbereich
. c7e0  8d 14 ff  sta $ff14     ... ein
. c7e3  ad 12 ff  lda $ff12
. c7e6  09 04     ora #$04
. c7e8  8d 12 ff  sta $ff12
. c7eb  a9 00     lda #$00      Schaltet auf
. c7ed  85 83     sta $83       ... Modus 0
. c7ef  60        rts           ENDE

. c7f0  :
. c7f0  *** SPEICHERAUFTEILUNG FÜR GRAFIK ***
. c7f0  :
```

```
. c7f0  a5 37     lda $37       Quellzeiger ($24-$25)
. c7f2  85 24     sta $24       ... gleich
. c7f4  a5 38     lda $38       ... Basic-
. c7f6  85 25     sta $25       ... Obergrenze
. c7f8  8a        txa           Länge
. c7f9  49 ff     eor #$ff      ... des
. c7fb  85 4e     sta $4e       ... zu verschiebenden
. c7fd  98        tya           ... Bereiches invertieren
. c7fe  49 ff     eor #$ff      ... und nach
. c800  85 4f     sta $4f       ... $4e-$4f schreiben
. c802  a0 00     ldy #$00
. c804  e6 4e     inc $4e       Verschiebelänge
. c806  d0 04     bne $c80c     ... gleich
. c808  e6 4f     inc $4f       ... Null ?
. c80a  f0 18     beq $c824     Ja, dann fertig
. c80c  a5 22     lda $22       Zielbereich
. c80e  d0 02     bne $c812     ... um 1
. c810  c6 23     dec $23       ... verringern...
. c812  c6 22     dec $22
. c814  a5 24     lda $24       Quellbereich
. c816  d0 02     bne $c81a     ... um 1
. c818  c6 25     dec $25       ... verringern...
. c81a  c6 24     dec $24
. c81c  20 bb 04  jsr $04bb     Holt Zeichen aus Quellbereich (LDA($24),Y)
. c81f  91 22     sta ($22),y   ... und in Ziebereich kopieren
. c821  4c 04 c8  jmp $c804     Springt nach $c804

. c824  60        rts           ENDE

. c825  a5 37     lda $37       Schreibt
. c827  a4 38     ldy $38       ... Basic-Obergrenze
. c829  85 22     sta $22       ... nach
. c82b  84 23     sty $23       ... $22-$23
. c82d  38        sec
. c82e  a5 33     lda $33       Stringbereich-
. c830  e5 22     sbc $22       ... Untergrenze
. c832  a5 34     lda $34       ... erreicht ?
. c834  e5 23     sbc $23
. c836  b0 ec     bcs $c824     Ja, dann fertig
. c838  38        sec
. c839  a5 22     lda $22       Setzt
. c83b  e9 02     sbc #$02      ... $22-$23
. c83d  85 22     sta $22       ... auf
. c83f  b0 02     bcs $c843     ... R-Zeiger...
. c841  c6 23     dec $23
. c843  a0 01     ldy #$01
. c845  20 b0 04  jsr $04b0     Holt R-Zeiger (LDA($22),Y)
. c848  99 24 00  sta $0024,y   ... und
. c84b  88        dey           ... schreibt
. c84c  10 f7     bpl $c845     ... diesen
. c84e  c8        iny           ... nach $24-$25
. c84f  20 bb 04  jsr $04bb     LDA($24),Y
```

```
. c852  85 80     sta $80       Schreibt Stringlänge aus Deskriptor
. c854  a5 22     lda $22       Subtrahiert
. c856  38        sec           ... Stringlänge
. c857  e5 80     sbc $80       ... von
. c859  85 22     sta $22       ... R-Zeiger-Adresse
. c85b  b0 02     bcs $c85f     ... Ergebnis gleich
. c85d  c6 23     dec $23       ... String-Anfangsadresse
. c85f  a0 02     ldy #$02      Schreibt
. c861  b9 21 00  lda $0021,y   ... String-
. c864  91 24     sta ($24),y   ... Anfangsadresse
. c866  88        dey           ... in
. c867  d0 f8     bne $c861     ... Deskriptor
. c869  f0 c2     beq $c82d     Springt nach $c82d

. c86b  :
. c86b  *** (X-Reg., Y-Reg.) := LÄNGE DES STRINGBEREICHES ***
. c86b  :
. c86b  38        sec           Subtrahiert
. c86c  a5 37     lda $37       ... Stringbereich-Anfang
. c86e  e5 33     sbc $33       ... von
. c870  aa        tax           ... Basic-Obergrenze
. c871  a5 38     lda $38       ... Ergebnis gleich
. c873  e5 34     sbc $34       ... Länge des
. c875  a8        tay           ... Stringbereiches
. c876  60        rts           ENDE

. c877  :
. c877  *** R - ZEIGER KORRIGIEREN ***
. c877  :
. c877  a5 37     lda $37       Schreibt
. c879  85 22     sta $22       ... Basic-Obergrenze
. c87b  a5 38     lda $38       ... nach
. c87d  85 23     sta $23       ... $22-$23
. c87f  38        sec
. c880  a5 33     lda $33       Subtrahiert
. c882  e5 22     sbc $22       ... $22-$23
. c884  a5 34     lda $34       ... von
. c886  e5 23     sbc $23       ... String-Untergrenze
. c888  b0 31     bcs $c8bb     Ergebnis gleich Null ?, dann fertig
. c88a  38        sec
. c88b  a5 22     lda $22       Setzt
. c88d  e9 02     sbc #$02      ... $22-$23
. c88f  85 22     sta $22       ... auf
. c891  b0 02     bcs $c895     ... R-Zeiger
. c893  c6 23     dec $23
. c895  18        clc
. c896  a0 00     ldy #$00      Kopiert
. c898  20 b0 04  jsr $04b0     ... R-Zeiger (LDA($22),Y
. c89b  99 24 00  sta $0024,y   ... nach $24-$25,
. c89e  79 4e 00  adc $004e,y   ... addiert $4e-$4f
. c8a1  91 22     sta ($22),y   ... und schreibt R-Zeiger
. c8a3  c8        iny           ... zurück...
```

```
. c8a4  c0 01    cpy #$01     (??? Dadurch wird nur das Low-Byte addiert)
. c8a6  d0 f0    bne $c898
. c8a8  88       dey
. c8a9  20 bb 04 jsr $04bb    Holt Stringlänge (LDA($24),Y)
. c8ac  85 80    sta $80      Stringlänge zwischenspeichern
. c8ae  a5 22    lda $22      Holt Deskriptor-Adresse
. c8b0  38       sec          ... und
. c8b1  e5 80    sbc $80      ... subtrahiert Stringlänge
. c8b3  85 22    sta $22      Schreibt Anfangsadresse
. c8b5  b0 c8    bcs $c87f    ... des
. c8b7  c6 23    dec $23      ... Strings
. c8b9  90 c4    bcc $c87f    Springt immer nach $c87f

. c8bb  60       rts          ENDE

. c8bc  :
. c8bc  *** D I R E C T O R Y ***
. c8bc  :
. c8bc  20 1f cb jsr $cb1f    DOS-Kommandoanalyse
. c8bf  29 e6    and #$e6     SYNTAX OK?
. c8c1  d0 7b    bne $c93e    Nein, dann Fehler
. c8c3  a0 00    ldy #$00
. c8c5  20 3f ca jsr $ca3f    Erzeugt Kommando
. c8c8  a9 00    lda #$00     Holt logische Adresse
. c8ca  ae 77 02 ldx $0277    Holt Geräteadresse
. c8cd  a0 60    ldy #$60     Holt Sekundäradresse
. c8cf  20 ba ff jsr $ffba    (SETLFS) Logische Datei einrichten
. c8d2  38       sec
. c8d3  20 c0 ff jsr $ffc0    (OPEN) Logische Datei öffnen
. c8d6  90 09    bcc $c8e1    Kein Fehler ?, dann weiter
. c8d8  48       pha          Fehlernummer auf Stack
. c8d9  20 35 c9 jsr $c935    (CLRCH), (CLOSE)
. c8dc  68       pla          Fehlernummer vom Stack
. c8dd  aa       tax          ... nach X-Reg. kopieren
. c8de  4c 83 86 jmp $8683    Fehlerausgabe

. c8e1  a2 00    ldx #$00
. c8e3  20 c6 ff jsr $ffc6    (CHKIN) öffnet Eingabekanal
. c8e6  a0 03    ldy #$03     Überliest Programm- und Zeilenlink
. c8e8  8c ec 02 sty $02ec    Schreibt Zeichenzähler
. c8eb  20 cf ff jsr $ffcf    (BASIN) Zeicheneingabe
. c8ee  8d ed 02 sta $02ed    Zeichen 1 zwischenspeichern
. c8f1  20 b7 ff jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. c8f4  d0 3f    bne $c935    Fehler ?, dann Abbruch
. c8f6  20 cf ff jsr $ffcf    (BASIN) Zeicheneingabe
. c8f9  8d ee 02 sta $02ee    Zeichen 2 zwischenspeichern
. c8fc  20 b7 ff jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. c8ff  d0 34    bne $c935    Fehler ?, dann Abbruch
. c901  ce ec 02 dec $02ec    Schon 6 Zeichen gelesen ?
. c904  d0 e5    bne $c8eb    Nein, dann weiterlesen
. c906  ae ed 02 ldx $02ed    Holt
. c909  ad ee 02 lda $02ee    ... Blockzahl
```

```
. c90c  20 5f a4 jsr $a45f    Gibt Integerzahl aus
. c90f  a9 20    lda #$20     Holt Leerzeichen
. c911  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. c914  20 cf ff jsr $ffcf    (BASIN) Zeicheneingabe
. c917  48       pha          Zeichen auf Stack retten
. c918  20 b7 ff jsr $ffb7    (READST) Ein-/Ausgabe-Status lesen
. c91b  d0 17    bne $c934    Fehler ?, dann Abbruch
. c91d  68       pla          Holt Zeichen vom Stack
. c91e  f0 06    beq $c926    Zeilenende ?, dann $c926
. c920  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. c923  4c 14 c9 jmp $c914    Weiterlesen

. c926  a9 0d    lda #$0d     Holt CR-Code
. c928  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. c92b  20 e1 ff jsr $ffe1    (STOP) Stop-Taste gedrückt ?
. c92e  f0 05    beq $c935    Ja, dann Abbruch
. c930  a0 02    ldy #$02     Überliest Zeilenlink
. c932  d0 b4    bne $c8e8    Springt immer nach $c8e8

. c934  68       pla
. c935  20 cc ff jsr $ffcc    (CLRCH) Schließt Ein-/Ausgabe-Kanäle
. c938  a9 00    lda #$00
. c93a  18       clc
. c93b  4c c3 ff jmp $ffc3    (CLOSE) Schließt logische Datei

. c93e  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. c941  :
. c941  *** D S A V E ***
. c941  :
. c941  a9 66    lda #$66     Filename, Geräteadresse, Laufwerk,  ?
. c943  20 21 cb jsr $cb21    DOS-Kommando-Analyse
. c946  20 b5 cc jsr $ccb5    Fehler, wenn Filename fehlt
. c949  a0 04    ldy #$04     Offset in Tabelle
. c94b  20 3f ca jsr $ca3f    Erzeugt Kommando
. c94e  4c e1 a7 jmp $a7e1    SAVE-Routine

. c951  :
. c951  *** D L O A D ***
. c951  :
. c951  a9 e6    lda #$e6     Filename, Geräteadresse, Laufwerk ?
. c953  20 21 cb jsr $cb21    DOS-Kommando-Analyse
. c956  20 b5 cc jsr $ccb5    Fehler, wenn Filename fehlt
. c959  a9 00    lda #$00     Vorbesetzung
. c95b  8d 78 02 sta $0278    ... für Sekundäradresse (0)
. c95e  85 0a    sta $0a      Setzt LOAD-Flag
. c960  a0 05    ldy #$05
. c962  20 3f ca jsr $ca3f    Generiert Kommandostring
. c965  4c fa a7 jmp $a7fa    LOAD-Routine

. c968  :
. c968  *** H E A D E R ***
```

```
. c968  :
. c968  20 1f cb jsr $cb1f    DOS-Kommando-Analyse
. c96b  20 af cc jsr $ccaf    Syntax-Check
. c96e  29 11    and #$11     Bits für Diskname und Laufwerk
. c970  c9 11    cmp #$11     Beide angegeben ?
. c972  f0 03    beq $c977    Ja, dann $c977
. c974  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. c977  20 e7 ff jsr $ffe7    (CLALL) Schließt alle Dateien und Kanäle
. c97a  20 2b cd jsr $cd2b    'ARE YOU SURE?'
. c97d  d0 17    bne $c996    Nein, dann $c996
. c97f  a0 09    ldy #$09     Offset in Tabelle
. c981  20 3f ca jsr $ca3f    Kommandostring erzeugen
. c984  20 cf cc jsr $cccf    Übernimmt Disk-Status
. c987  24 81    bit $81      Direktmodus ?
. c989  30 0b    bmi $c996    Nein, dann $c996
. c98b  a0 00    ldy #$00
. c98d  a9 7a    lda #$7a
. c98f  20 94 04 jsr $0494    Holt 1. Stelle von DS (LDA(AKKU),Y)
. c992  c9 32    cmp #$32     Größer als '2' ?
. c994  b0 01    bcs $c997    Ja, dann Formatierungsfehler
. c996  60       rts          ENDE

. c997  a2 24    ldx #$24     Holt Fehlernummer 24 ('BAD DISK')
. c999  4c 83 86 jmp $8683    Fehlerausgabe

. c99c  :
. c99c  *** S C R A T C H ***
. c99c  :
. c99c  20 1f cb jsr $cb1f    DOS-Kommando-Analyse
. c99f  20 af cc jsr $ccaf    Syntax-Check
. c9a2  20 2b cd jsr $cd2b    'ARE YOU SURE?'
. c9a5  d0 ef    bne $c996    Nein, dann $c996
. c9a7  a0 0f    ldy #$0f     Holt Offset in Tabelle
. c9a9  20 3f ca jsr $ca3f    Erzeugt Kommando
. c9ac  20 cf cc jsr $cccf    Übernimmt Disk-Status
. c9af  24 81    bit $81      Direktmodus ?
. c9b1  30 e3    bmi $c996    Nein, dann $c996
. c9b3  a9 0d    lda #$0d     Holt CR-Code
. c9b5  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. c9b8  a0 00    ldy #$00     Holt Adresse
. c9ba  a9 7a    lda #$7a     ... von DS$
. c9bc  20 94 04 jsr $0494    Holt Zeichen aus Meldung (LDA(AKKU),Y)
. c9bf  f0 06    beq $c9c7    Endmarke ?, dann $c9c7
. c9c1  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. c9c4  c8       iny          Zeiger um 1 erhöhen
. c9c5  d0 f3    bne $c9ba    Springt immer nach $c9ba

. c9c7  a9 0d    lda #$0d     Holt CR-Code
. c9c9  4c d2 ff jmp $ffd2    (BSOUT) Zeichen ausgeben

. c9cc  :
```

```
. c9cc  *** C O L L E C T ***
. c9cc  :
. c9cc  20 1f cb jsr $cb1f    DOS-Kommando-Analyse
. c9cf  29 e7    and #$e7     Nur Gerätenummer und Laufwerk ?
. c9d1  d0 a1    bne $c974    Nein, dann Fehler
. c9d3  20 e7 ff jsr $ffe7    (CLALL) Schließt alle Dateien und Kanäle
. c9d6  a0 14    ldy #$14     Holt Offset in Tabelle
. c9d8  d0 65    bne $ca3f    Erzeugt Kommandostring

. c9da  :
. c9da  *** C O P Y ***
. c9da  :
. c9da  20 1f cb jsr $cb1f    DOS-Kommando-Analyse
. c9dd  29 30    and #$30     Zwei Laufwerknummern
. c9df  c9 30    cmp #$30     ... angegeben ?
. c9e1  d0 06    bne $c9e9    Nein, dann $c9e9
. c9e3  a5 82    lda $82      Holt SYNTAX-Flag
. c9e5  29 c7    and #$c7     Filenamen angegeben ?
. c9e7  f0 07    beq $c9f0    Nein, dann $c9f0
. c9e9  a5 82    lda $82      Holt SYNTAX-Flag
. c9eb  20 c0 cc jsr $ccc0    Syntax-Check
. c9ee  a5 82    lda $82      Holt SYNTAX-Flag
. c9f0  a0 17    ldy #$17     Holt Offset in Tabelle
. c9f2  d0 4b    bne $ca3f    Erzeugt Kommandostring

. c9f6  :
. c9f6  *** R E N A M E ***
. c9f6  :
. c9f4  a9 e4    lda #$e4     2 Filenamen, Geräteadresse, Laufwerk ?
. c9f6  20 21 cb jsr $cb21    DOS-Kommando-Analyse
. c9f9  20 c6 cc jsr $ccc6    Fehler, wenn keine 2 Filenamen
. c9fc  a0 1e    ldy #$1e     Holt Offset in Tabelle
. c9fe  d0 3f    bne $ca3f    Erzeugt Kommandostring

. ca00  :
. ca00  *** B A C K U P ***
. ca00  :
. ca00  a9 c7    lda #$c7     Geräteadresse, 2 Laufwerke
. ca02  20 21 cb jsr $cb21    DOS-Kommando-Analyse
. ca05  29 30    and #$30     2 Laufwerke
. ca07  c9 30    cmp #$30     ... angegeben ?
. ca09  f0 03    beq $ca0e    Ja, dann $ca0e
. ca0b  4c a1 94 jmp $94a1    'SYNTAX ERROR' ausgeben

. ca0e  20 e7 ff jsr $ffe7    (CLALL) Schließt alle Dateien und Kanäle
. ca11  a0 25    ldy #$25     Holt Offset in Tabelle
. ca13  4c 3f ca jmp $ca3f    Erzeugt Kommandostring

. ca16  :
. ca16  *** KOMMANDOSTRING SENDEN ***
. ca16  :
. ca16  48       pha          Endmarke in Tabelle auf Stack legen
```

```
. ca17  ad 5d 02 lda $025d    Holt Länge des Filenamens
. ca1a  a2 7c    ldx #$7c     Holt Adresse
. ca1c  a0 02    ldy #$02     ... des Filenamens
. ca1e  20 bd ff jsr $ffbd    (SETNAM) Dateinamen übergeben
. ca21  ad 76 02 lda $0276    Holt logische Adresse
. ca24  ae 77 02 ldx $0277    Holt Geräteadresse
. ca27  ac 78 02 ldy $0278    Holt Sekundäradresse
. ca2a  20 ba ff jsr $ffba    (SETLFS) Logische Datei einrichten
. ca2d  68       pla          Holt Endmarke vom Stack
. ca2e  f0 0e    beq $ca3e    Endmarke gleich Null ?, dann fertig
. ca30  ae 5d 02 ldx $025d    Holt Länge des Kommandos
. ca33  38       sec
. ca34  20 c0 ff jsr $ffc0    (OPEN) Sendet Kommando
. ca37  ad 76 02 lda $0276    Holt logische Adresse
. ca3a  38       sec
. ca3b  4c c3 ff jmp $ffc3    (CLOSE) logische Datei schließen

. ca3e  60       rts          ENDE

. ca3f  :
. ca3f  *** KOMMANDOSTRING ERZEUGEN ***
. ca3f  :
. ca3f  20 57 cd jsr $cd57    Löscht Disk-Status
. ca42  a2 00    ldx #$00     Initialisiert
. ca44  8e 5d 02 stx $025d    ... Zähler für Kommandolänge
. ca47  b9 f5 ca lda $caf5,y  Holt Zeichen aus Tabelle
. ca4a  f0 ca    beq $ca16    Endmarke $00 ?, dann Kommandostring senden
. ca4c  c9 80    cmp #$80     Endmarke $80 ?
. ca4e  f0 c6    beq $ca16    Ja, dann Kommandostring senden
. ca50  aa       tax          Code nach X-Reg.
. ca51  ca       dex          1: Laufwerknummer 1, falls vorhanden ?
. ca52  f0 1e    beq $ca72    Ja, dann $ca72
. ca54  ca       dex          2: Laufwerknummer 1 ?
. ca55  f0 21    beq $ca78    Ja, dann $ca78
. ca57  ca       dex          3: Laufwerknummer 2 ?
. ca58  f0 24    beq $ca7e    Ja, dann $ca7e
. ca5a  ca       dex          4: REPLACE-Code ?
. ca5b  f0 28    beq $ca85    Ja, dann $ca85
. ca5d  ca       dex          5: Dateiname 1, falls vorhanden ?
. ca5e  f0 2d    beq $ca8d    Ja, dann $ca8d
. ca60  ca       dex          6: Dateiname 1 ?
. ca61  f0 34    beq $ca97    Ja, dann $ca97
. ca63  ca       dex          7: Dateiname 2, falls vorhanden ?
. ca64  f0 3f    beq $caa5    Ja, dann $caa5
. ca66  ca       dex          8: Dateiname 2 ?
. ca67  f0 46    beq $caaf    Ja, dann $caaf
. ca69  ca       dex          9: ID, falls vorhanden
. ca6a  f0 66    beq $cad2    Ja, dann $cad2

. ca6c  20 eb ca jsr $caeb    Kommandostring einbauen
. ca6f  c8       iny          Erhöht Offset in Tabelle um 1
. ca70  d0 d5    bne $ca47    Springt immer nach $ca47
```

```
      LAUFWERKNUMMER 1

. ca72  a5 82     lda $82       Holt SYNTAX-Flag
. ca74  29 10     and #$10      Laufwerknummer angegeben ?
. ca76  f0 f7     beq $ca6f     Nein, dann $ca6f
. ca78  ad 6f 02  lda $026f     Holt Laufwerknummer 1
. ca7b  4c 81 ca  jmp $ca81     Springt nach $ca81

      LAUFWERKNUMMER 2

. ca7e  ad 73 02  lda $0273     Holt Laufwerknummer 2
. ca81  09 30     ora #$30      Erzeugt ASCII-Code
. ca83  d0 e7     bne $ca6c     Springt immer nach $ca6c

      REPLACE

. ca85  a9 40     lda #$40      Holt Replace-Code
. ca87  24 82     bit $82       SYNTAX-Flag ?
. ca89  30 e1     bmi $ca6c     Replace gefordert ?, dann $ca6c
. ca8b  10 e2     bpl $ca6f     Sonst: $ca6f

      DATEINAME 1

. ca8d  ad 6e 02  lda $026e     Holt Länge des Dateinamens 1
. ca90  f0 dd     beq $ca6f     Länge gleich Null ?, dann $ca6f
. ca92  a9 3a     lda #$3a      Holt Doppelpunkt
. ca94  20 eb ca  jsr $caeb     Schreibt Doppelpunkt in Kommandostring
. ca97  98        tya           Offset in Tabelle
. ca98  48        pha           ... auf Stack retten
. ca99  ad 70 02  lda $0270     Holt Adresse
. ca9c  ac 71 02  ldy $0271     ... des Dateinamens 1
. ca9f  ae 6e 02  ldx $026e     Holt Länge des Dateinamens 1
. caa2  4c ba ca  jmp $caba     Dateiname 1 in Kommandostring kopieren

      DATEINAME 2

. caa5  ad 72 02  lda $0272     Holt Länge des Dateinamens 2
. caa8  f0 c5     beq $ca6f     Länge gleich Null ?, dann $ca6f
. caaa  a9 3a     lda #$3a      Holt Doppelpunkt
. caac  20 eb ca  jsr $caeb     Schreibt Doppelpunkt in Kommandostring
. caaf  98        tya           Offset in Tabelle
. cab0  48        pha           ... auf Stack retten
. cab1  ad 74 02  lda $0274     Holt Adresse
. cab4  ac 75 02  ldy $0275     ... des Dateinamens 2
. cab7  ae 72 02  ldx $0272     Dateiname 2 in Kommandostring kopieren

      DATEINAME IN KOMMANDOSTRING KOPIEREN

. caba  85 22     sta $22       Schreibt Adresse des Dateinamens
. cabc  84 23     sty $23       ... in Quellzeiger
. cabe  86 80     stx $80       Schreibt Länge des Dateinamens
```

```
. cac0  a0 00    ldy #$00     Offset initialisieren
. cac2  20 b0 04 jsr $04b0    Holt Zeichen aus Quellbereich (LDA($22),Y)
. cac5  20 eb ca jsr $caeb    Schreibt Zeichen in Kommandostring
. cac8  c8       iny          Offset um 1 erhöhen
. cac9  c4 80    cpy $80      Ende des Dateinamens erreicht ?
. cacb  d0 f5    bne $cac2    Nein, dann weiterkopieren
. cacd  68       pla          Holt Offset in Tabelle
. cace  a8       tay          ... vom Stack
. cacf  4c 6f ca jmp $ca6f    Springt nach $ca6f

      ID

. cad2  ad 79 02 lda $0279    ID gleich Null ?
. cad5  f0 98    beq $ca6f    Ja, dann $ca6f
. cad7  a9 2c    lda #$2c     Holt Komma ','
. cad9  20 eb ca jsr $caeb    Komma in Kommandostring einbauen
. cadc  ad 79 02 lda $0279    Holt ID (Byte 1)
. cadf  20 eb ca jsr $caeb    ... und in Kommandostring einbauen
. cae2  ad 7a 02 lda $027a    Holt ID (Byte 2)
. cae5  20 eb ca jsr $caeb    ... und in Kommandostring einbauen
. cae8  4c 6f ca jmp $ca6f    Springt nach $ca6f

      KOMMANDOSTRING EINBAUEN

. caeb  ae 5d 02 ldx $025d    Holt Offset im Kommandostring
. caee  9d 7c 02 sta $027c,x  Schreibt Zeichen in Kommandostring
. caf1  ee 5d 02 inc $025d    Erhöht Offset um 1
. caf4  60       rts          ENDE

. caf5  :
. caf5  *** DOS-Tabelle ***
. caf5  :
. caf5  '$' $07 $05 $00 $04 $02      DIRECTORY
. cafb  ':' $06 $00                  DLOAD, DSAVE
. cafe  'n' $02 ':' $06 $09 $80      HEADER
. cb04  's' $02 ':' $06 $80          SCRATCH
. cb09  'v' $02 $80                  COLLECT
. cb0c  'c' $03 $07 '=' $02 $05 $80  COPY
. cb13  'r' $02 ':' $08 '=' $06 $80  RENAME
. cb1a  'd' $03 '=' $02 $80          BACKUP

. cb1f  :
. cb1f  *** DOS KOMMANDO-ANALYSE ***
. cb1f  :
. cb1f  a9 00    lda #$00     Testbyte für SYNTAX-Flag
. cb21  48       pha          Testbyte auf Stack retten
. cb22  a9 00    lda #$00     Vorbelegung
. cb24  85 82    sta $82      ... für Syntax-Flag
. cb26  a2 1e    ldx #$1e     Löscht
. cb28  9d 5e 02 sta $025e,x  ... DOS-
. cb2b  ca       dex          ... Arbeitsbereich...
. cb2c  d0 fa    bne $cb28
```

```
. cb2e  a2 Ø8     ldx #$Ø8      Vorbesetzung
. cb3Ø  8e 77 Ø2 stx $Ø277      ... für Geräteadresse (8)
. cb33  a2 6f     ldx #$6f      Vorbesetzung
. cb35  8e 78 Ø2 stx $Ø278      ... für Sekundäradresse (15)
. cb38  a2 ØØ     ldx #$ØØ      Vorbesetzung
. cb3a  8e 76 Ø2 stx $Ø276      ... für logische Adresse (Ø)
. cb3d  2Ø 79 Ø4 jsr $Ø479      CHRGOT-Routine
. cb4Ø  dØ Ø7     bne $cb49     Kein Trennzeichen ?, dann $cb49
. cb42  68        pla           Holt Testbyte für SYNTAX-Flag
. cb43  2Ø aa cc jsr $ccaa      SYNTAX-Flag prüfen
. cb46  a5 82     lda $82       Holt SYNTAX-Flag
. cb48  6Ø        rts           ENDE

. cb49  c9 44     cmp #$44      Zeichen gleich 'd' für DRIVE ?
. cb4b  fØ 1d     beq $cb6a     Ja, dann $cb6a
. cb4d  c9 91     cmp #$91      Zeichen 'ON'-Token ?
. cb4f  fØ 4e     beq $cb9f     Ja, dann $cb9f
. cb51  c9 55     cmp #$55      Zeichen gleich 'u' für UNIT ?
. cb53  fØ Øf     beq $cb64     Ja, dann $cb64
. cb55  c9 49     cmp #$49      Zeichen gleich 'i' für ID ?
. cb57  fØ 2a     beq $cb83     Ja, dann $cb83
. cb59  c9 22     cmp #$22      Zeichen gleich '"' für Dateinamen ?
. cb5b  fØ 48     beq $cba5     Ja, dann $cba5
. cb5d  c9 28     cmp #$28      Zeichen gleich '(' für Variable ?
. cb5f  fØ 44     beq $cba5     Ja, dann $cba5
. cb61  4c a1 94 jmp $94a1      'SYNTAX ERROR' ausgeben

     UNIT

. cb64  2Ø 58 cc jsr $cc58      Gerätenummer auswerten
. cb67  4c cd cb jmp $cbcd      Springt immer nach $cbcd

     LAUFWERK 1

. cb6a  a9 1Ø     lda #$1Ø      Holt Laufwerk-Bit 1
. cb6c  2Ø aa cc jsr $ccaa      Fehler, wenn schon gesetzt
. cb6f  2Ø 97 cc jsr $cc97      (FRMEVL), (GETBYT) Holt Laufwerknr.
. cb72  eØ Ø2     cpx #$Ø2      Laufwerk größer 1 ?
. cb74  bØ Øa     bcs $cb8Ø     Ja, dann Fehler
. cb76  8e 6f Ø2 stx $Ø26f      Schreibt Laufwerk 1
. cb79  8e 73 Ø2 stx $Ø273      Schreibt Laufwerk 2
. cb7c  a9 1Ø     lda #$1Ø      Setzt
. cb7e  dØ 4d     bne $cbcd     ... Laufwerk-Bit

. cb8Ø  4c 49 cc jmp $cc49      'ILLEGAL QUANTITY ERROR' ausgeben

     ID

. cb83  ad 7b Ø2 lda $Ø27b      ID-Flag gesetzt ?
. cb86  dØ d9     bne $cb61     Ja, dann Fehler
. cb88  2Ø 73 Ø4 jsr $Ø473      CHRGET-Routine
. cb8b  8d 79 Ø2 sta $Ø279      Schreibt ID-Byte 1
```

```
. cb8e  20 73 04 jsr $0473    CHRGET-Routine
. cb91  8d 7a 02 sta $027a    Schreibt ID-Byte 2
. cb94  a9 ff    lda #$ff     Setzt
. cb96  8d 7b 02 sta $027b    ... ID-Flag
. cb99  20 73 04 jsr $0473    CHRGET-Routine
. cb9c  4c d1 cb jmp $cbd1    Springt nach $cbd1

      ON

. cb9f  20 51 cc jsr $cc51    Gerätenummer auswerten
. cba2  4c cd cb jmp $cbcd    Springt immer nach $cbcd

      DATEINAME 1

. cba5  a9 01    lda #$01     Holt Dateiname-Bit 1
. cba7  20 69 cc jsr $cc69    Übernimmt Dateiname
. cbaa  8d 6e 02 sta $026e    Schreibt Länge des Dateinamens
. cbad  8d 5d 02 sta $025d    Schreibt Länge des Dateinamens
. cbb0  a9 5e    lda #$5e     Schreibt
. cbb2  8d 70 02 sta $0270    ... Adresse
. cbb5  85 24    sta $24      ... des
. cbb7  a9 02    lda #$02     ... Dateinamens
. cbb9  8d 71 02 sta $0271    ... in Puffer
. cbbc  85 25    sta $25      ... und Zielzeiger
. cbbe  a0 00    ldy #$00     Kopiert Dateiname
. cbc0  20 b0 04 jsr $04b0    ... (LDA($22),Y)
. cbc3  91 24    sta ($24),y  ... in Puffer...
. cbc5  c8       iny
. cbc6  cc 5d 02 cpy $025d    Ende des Dateinamens erreicht ?
. cbc9  90 f5    bcc $cbc0    Nein, dann weiterkopieren
. cbcb  a9 01    lda #$01     Setzt Bit
. cbcd  05 82    ora $82      ... für Dateiname 1
. cbcf  85 82    sta $82      ... im SYNTAX-Flag
. cbd1  20 79 04 jsr $0479    CHRGOT-Routine
. cbd4  d0 03    bne $cbd9    Keine Endmarke ?, dann $cbd9
. cbd6  4c 42 cb jmp $cb42    ENDE

. cbd9  c9 2c    cmp #$2c     Zeichen gleich Komma ',' ?
. cbdb  d0 06    bne $cbe3    Nein, dann $cbe3
. cbdd  20 73 04 jsr $0473    CHRGET-Routine
. cbe0  4c 49 cb jmp $cb49    Weiter in DOS-Kommandoanalyse

. cbe3  c9 91    cmp #$91     Zeichen gleich 'ON'-Token ?
. cbe5  f0 b8    beq $cb9f    Ja, dann $cb9f
. cbe7  c9 a4    cmp #$a4     Zeichen gleich 'TO'-Token ?
. cbe9  d0 5b    bne $cc46    Nein, dann Fehler
. cbeb  20 73 04 jsr $0473    CHRGET-Routine
. cbee  c9 44    cmp #$44     Zeichen gleich 'd' für Laufwerk ?
. cbf0  f0 10    beq $cc02    Ja, dann $cc02
. cbf2  c9 91    cmp #$91     Zeichen gleich 'ON'-Token ?
. cbf4  f0 1f    beq $cc15    Ja, dann $cc15
. cbf6  c9 55    cmp #$55     Zeichen gleich 'u' für Unit ?
```

```
. cbf8  f0 21     beq $cc1b     Ja, dann $cc1b
. cbfa  c9 22     cmp #$22      Anführungszeichen für Text ?
. cbfc  f0 23     beq $cc21     Ja, dann $cc21
. cbfe  c9 28     cmp #$28      Zeichen gleich '(' für Variable ?
. cc00  f0 1f     beq $cc21     Ja, dann $cc21

      LAUFWERK 2

. cc02  a9 20     lda #$20      Holt Laufwerk-Bit 2
. cc04  20 aa cc  jsr $ccaa     Fehler, wenn schon gesetzt
. cc07  20 97 cc  jsr $cc97     (FRMEVL), (GETBYT) Holt Laufwerknr.
. cc0a  e0 02     cpx #$02      Laufwerk größer 1 ?
. cc0c  b0 3b     bcs $cc49     Ja, dann Fehler
. cc0e  8e 73 02  stx $0273     Schreibt Laufwerk 2
. cc11  a9 20     lda #$20      Setzt
. cc13  d0 1c     bne $cc31     ... Laufwerk-Bit 2

      ON

. cc15  20 51 cc  jsr $cc51     Übernimmt 'u'
. cc18  4c 31 cc  jmp $cc31     Setzt SYNTAX-Flag

      GERÄTENUMMER

. cc1b  20 58 cc  jsr $cc58     Übernimmt Gerätenummer
. cc1e  4c 31 cc  jmp $cc31     Setzt SYNTAX-Flag

      DATEINAME 2

. cc21  a9 02     lda #$02      Holt Dateiname-Bit 2
. cc23  20 69 cc  jsr $cc69     Übernimmt Dateiname
. cc26  8d 72 02  sta $0272     Schreibt Länge des Dateinamens
. cc29  8e 74 02  stx $0274     Schreibt Adresse
. cc2c  8c 75 02  sty $0275     ... des Dateinamens
. cc2f  a9 02     lda #$02      Setzt
. cc31  05 82     ora $82       ... Dateiname-Bit 2
. cc33  85 82     sta $82       ... in SYNTAX-Flag
. cc35  20 79 04  jsr $0479     CHRGOT-Routine
. cc38  f0 9c     beq $cbd6     Folgt Endemarke ?, dann $cbd6
. cc3a  c9 2c     cmp #$2c      Zeichen gleich Komma ',' ?
. cc3c  f0 ad     beq $cbeb     Ja, dann $cbeb
. cc3e  c9 91     cmp #$91      Zeichen gleich 'ON'-Token
. cc40  f0 d3     beq $cc15     Ja, dann $cc15
. cc42  c9 55     cmp #$55      Zeichen gleich 'u' für Unit ?
. cc44  f0 d5     beq $cc1b     Ja, dann $cc1b
. cc46  a2 0b     ldx #$0b      Holt Fehlernummer 11 ('SYNTAX')

. cc48  2c        .byte $2c

. cc49  a2 0e     ldx #$0e      Holt Fehlernummer 14 ('ILLEGAL QUANTITY')

. cc4b  2c        .byte $2c
```

```
. cc4c  a2 17                Holt Fehlernummer 23 ('STRING TOO LONG')

. cc4e  4c 83 86 jmp $8683   Fehlerausgabe

     GERÄTENUMMER AUSWERTEN

. cc51  20 73 04 jsr $0473   CHRGET-Routine
. cc54  c9 55    cmp #$55    Zeichen gleich 'u' für Unit ?
. cc56  d0 ee    bne $cc46   Nein, dann Fehler
. cc58  20 97 cc jsr $cc97   (FRMEVL), (GETBYT) Holt Gerätenummer
. cc5b  e0 20    cpx #$20    Gerätenummer größer 31 ?
. cc5d  b0 ea    bcs $cc49   Ja, dann Fehler
. cc5f  e0 03    cpx #$03    Gerätenummer kleiner 3 ?
. cc61  90 e6    bcc $cc49   Ja, dann Fehler
. cc63  8e 77 02 stx $0277   Schreibt Geräteadresse
. cc66  a9 08    lda #$08    Holt Gerätenummer-Bit
. cc68  60       rts         ENDE

     DATEINAMEN ÜBERNEHMEN

. cc69  20 aa cc jsr $ccaa   Fehler, wenn schon gesetzt
. cc6c  20 48 9c jsr $9c48   (FRMEVL), (CHKSTR), (FRESTR)
. cc6f  aa       tax         Stringlänge nach X-Reg.
. cc70  f0 d7    beq $cc49   Stringlänge gleich Null ?, dann Fehler
. cc72  a0 00    ldy #$00
. cc74  20 b0 04 jsr $04b0   Holt 1. Zeichen aus Namen (LDA($22),Y)
. cc77  c9 40    cmp #$40    Zeichen gleich Klammeraffe ?
. cc79  d0 12    bne $cc8d   Nein, dann $cc8d
. cc7b  a9 80    lda #$80    Holt Replace-Bit
. cc7d  20 aa cc jsr $ccaa   Fehler, wenn schon gesetzt
. cc80  a5 82    lda $82     Setzt
. cc82  09 80    ora #$80    ... Replace-
. cc84  85 82    sta $82     ... Bit
. cc86  ca       dex         Erniedrigt Stringlänge um 1
. cc87  e6 22    inc $22     Erhöht
. cc89  d0 02    bne $cc8d   ... String-Anfangsadresse
. cc8b  e6 23    inc $23     ... um 1
. cc8d  8a       txa         Stringlänge nach Akku
. cc8e  c9 11    cmp #$11    Stringlänge größer 16 ?
. cc90  b0 ba    bcs $cc4c   Ja, dann Fehler
. cc92  a6 22    ldx $22     Holt
. cc94  a4 23    ldy $23     ... String-Anfangsadresse
. cc96  60       rts         ENDE

     BYTE/ADRESSE AUSWERTEN

. cc97  20 73 04 jsr $0473   CHRGET-Routine
. cc9a  f0 aa    beq $cc46   Endemarke ?, dann Fehler
. cc9c  90 09    bcc $cca7   Zeichen gleich Ziffer ?, dann $cca7
. cc9e  20 8e 94 jsr $948e   Prüft ob '(' folgt
. cca1  20 84 9d jsr $9d84   (FRMEVL), (GETBYT)
```

```
. cca4  4c 8b 94 jmp $948b    Prüft ob ')' folgt

. cca7  4c 84 9d jmp $9d84    (FRMEVL), (GETBYT)

. ccaa  25 82    and $82      SYNTAX-Flag
. ccac  dØ 98    bne $cc46    Nein, dann Fehler
. ccae  6Ø       rts          ENDE

. ccaf  29 e6    and #$e6     2. Dateiname, 2. Laufwerk, Geräteadresse ?
. ccb1  fØ Ø2    beq $ccb5    Nein, dann $ccb5
. ccb3  dØ 91    bne $cc46    Sonst: Fehler

. ccb5  a5 82    lda $82      Holt SYNTAX-Flag
. ccb7  29 Ø1    and #$Ø1     Dateiname
. ccb9  c9 Ø1    cmp #$Ø1     ... angegeben ?
. ccbb  dØ f6    bne $ccb3    Nein, dann Fehler
. ccbd  a5 82    lda $82      Holt SYNTAX-Flag
. ccbf  6Ø       rts          ENDE

. cccØ  29 c4    and #$c4     Geräteadresse oder Replace ?
. ccc2  dØ ef    bne $ccb3    Ja, dann Fehler
. ccc4  a5 82    lda $82      Holt SYNTAX-Flag
. ccc6  29 Ø3    and #$Ø3     2 Dateinamen
. ccc8  c9 Ø3    cmp #$Ø3     ... angegeben ?
. ccca  dØ e7    bne $ccb3    Nein, dann Fehler
. cccc  a5 82    lda $82      Holt SYNTAX-Flag
. ccce  6Ø       rts          ENDE

. cccf  :
. cccf  *** DS$ ÜBERNEHMEN ***
. cccf  :
. cccf  a5 79    lda $79      Holt Länge von DS$
. ccd1  dØ 11    bne $cce4    DS$ gültig ?, dann $cce4
. ccd3  a9 28    lda #$28     Setzt Länge
. ccd5  85 79    sta $79      ... auf 4Ø Zeichen
. ccd7  2Ø Ø6 a9 jsr $a9Ø6    Platz für String reservieren
. ccda  86 7a    stx $7a      Schreibt Adresse
. ccdc  84 7b    sty $7b      ... von DS$
. ccde  aØ 28    ldy #$28
-------------------------------------------------------------------------
. cceØ  a9 Ød    lda #$Ød     Schreibt $Ød in R-Zeiger ?? (Unsinn!)
. cce2  91 7a    sta ($7a),y      (Alte Betriebssystem-Version)
-------------------------------------------------------------------------
. cceØ  2Ø ba cd jsr $cdba    Schreibt Adresse $ØØ79 in R-Zeiger
. cce3  ea       nop              (Neue Betriebssystem-Version)
. cce4  ae 77 Ø2 ldx $Ø277    Holt Geräteadresse
. cce7  dØ Ø5    bne $ccee    Geräteadresse ungleich Null ?, dann $ccee
. cce9  a2 Ø8    ldx #$Ø8     Sonst: Geräteadresse
. cceb  8e 77 Ø2 stx $Ø277    ... mit 8 besetzen
. ccee  a9 ØØ    lda #$ØØ     Logische Adresse gleich Ø
. ccfØ  aØ 6f    ldy #$6f     Sekundäradresse gleich 15
. ccf2  2Ø ba ff jsr $ffba    (SETLFS) Logische Datei einrichten
```

```
. ccf5  a9 00     lda #$00      Flag für kein Dateiname
. ccf7  20 bd ff  jsr $ffbd     (SETNAM) Dateiname löschen
. ccfa  20 c0 ff  jsr $ffc0     (OPEN) Logische Datei öffnen
. ccfd  a2 00     ldx #$00      Kanal 0
. ccff  20 c6 ff  jsr $ffc6     (CHKIN)  öffnet Eingabekanal
. cd02  b0 1b     bcs $cd1f     Fehler ?, dann $cd1f
. cd04  a0 ff     ldy #$ff
. cd06  c8        iny
. cd07  20 cf ff  jsr $ffcf     (BASIN) Zeicheneingabe
. cd0a  c9 0d     cmp #$0d      Zeichen gleich CR-Code ?
. cd0c  f0 04     beq $cd12     Ja, dann fertig
. cd0e  91 7a     sta ($7a),y   Zeichen in DS$ kopieren
. cd10  d0 f4     bne $cd06     Springt immer nach $cd06

. cd12  a9 00     lda #$00      Schreibt $00
. cd14  91 7a     sta ($7a),y   ... als Endmarke hinter DS$
. cd16  20 cc ff  jsr $ffcc     (CLRCH) Schließt Ein-/Ausgabe-Kanäle
. cd19  a9 00     lda #$00
. cd1b  38        sec
. cd1c  4c c3 ff  jmp $ffc3     (CLOSE) Schließt logische Datei

. cd1f  48        pha           Fehlercode auf Stack
. cd20  20 12 cd  jsr $cd12     String abschließen
. cd23  20 57 cd  jsr $cd57     DS$ löschen
. cd26  68        pla           Holt Fehlercode vom Stack
. cd27  aa        tax           Fehlercode nach X-Reg.
. cd28  4c 83 86  jmp $8683     Fehlerausgabe

. cd2b  :
. cd2b  *** 'ARE YOU SURE ?' ***
. cd2b  :
. cd2b  24 81     bit $81       Direktmodus ?
. cd2d  30 25     bmi $cd54     Nein, dann keine Meldung
. cd2f  20 4f ff  jsr $ff4f     Meldung ausgeben

. cd32  41 52 45 20 59 4f 55 20 Meldung: 'ARE YOU SURE?'
. cd3a  53 55 52 45 3f
. cd3f  00                      .byte $00

. cd40  20 cc ff  jsr $ffcc     (CLRCH) Schließt Ein-/Ausgabe-Kanäle
. cd43  20 cf ff  jsr $ffcf     (BASIN) Zeicheneingabe
. cd46  48        pha           1. Zeichen auf Stack
. cd47  c9 0d     cmp #$0d      Zeichen gleich CR-Code ?
. cd49  f0 05     beq $cd50     Ja, dann $cd50
. cd4b  20 cf ff  jsr $ffcf     (BASIN) Zeicheneingabe
. cd4e  d0 f7     bne $cd47     Springt immer nach $cd47

. cd50  68        pla           Holt 1. eingegebenes Zeichen
. cd51  c9 59     cmp #$59      Zeichen gleich 'y' ?, dann Z-Flag setzen
. cd53  60        rts           ENDE

. cd54  a9 00     lda #$00      Z-Flag setzen (simuliert 'Yes')
```

```
. cd56  60       rts          ENDE

. cd57  :
. cd57  *** DS$ LÖSCHEN ***
. cd57  :
. cd57  98       tya
. cd58  48       pha
. cd59  a5 79    lda $79      Holt Länge von DS$
. cd5b  f0 0a    beq $cd67    Länge gleich 0 ?, dann $cd67
. cd5d  a0 28    ldy #$28     Macht
. cd5f  98       tya          ... DS$
. cd60  91 7a    sta ($7a),y  ... ungültig...
. cd62  c8       iny
. cd63  a9 ff    lda #$ff
. cd65  91 7a    sta ($7a),y
. cd67  a9 00    lda #$00     Setzt Länge von
. cd69  85 79    sta $79      ... DS$ auf Null
. cd6b  68       pla
. cd6c  a8       tay
. cd6d  60       rts          ENDE

. cd6e  :
. cd6e  *** RESTE VON KEY ***
. cd6e  :
. cd6e  2c 30 20 59 45 4b     .byte ',0 yek' := (KEY 0,) rückwärts

. cd74  aa       tax
. cd75  98       tya
. cd76  48       pha
. cd77  a9 00    lda #$00
. cd79  20 5f a4 jsr $a45f    Integerzahl (Akku, X-Reg.) ausgeben
. cd7c  68       pla
. cd7d  a8       tay
. cd7e  60       rts          ENDE

. cd7f  :
. cd7f  *** RUCKSACK AUS LOOP ***
. cd7f  :
. cd7f  85 3a    sta $3a      Schreibt Zeilennummer (High)
. cd81  88       dey
. cd82  aa       tax          Zeilennummer (High)
. cd83  e8       inx          ... gleich $ff ?
. cd84  d0 02    bne $cd88    Nein, dann $cd88
. cd86  86 81    stx $81      Setzt Direktmodus
. cd88  60       rts          ENDE

. cd89  :
. cd89  *** VERSCHLÜSSELTE AUTORENLISTE ***
. cd89  :

. cd89  d8 1b 14 0c 07 7b 01 d7        t.ryan
. cd91  d8 11 07 10 1d 7b 17           b.herd
```

```
. cd98  d8 Ø7 1Ø Ø5 1a 1a 16 7b 1f      j.cooper
. cda1  d8 1b 1Ø Ø2 1a 17 7b 13 47 d8   f.bowen

. cdab  :
. cdab  *** AUTOREN AUSGEBEN ***
. cdab  :
. cdab  aØ 21     ldy #$21     Offset in Autorenliste
. cdad  b9 89 cd lda $cd89,y  Holt Zeichen aus Autorenliste
. cdbØ  49 55     eor #$55     Entschlüsselt Zeichen
. cdb2  2Ø d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. cdb5  88        dey          Offset verringern
. cdb6  1Ø f5     bpl $cdad    Noch nicht Ø ?, dann $cdad
. cdb8  6Ø        rts          ENDE

. cdb9  ØØ        .byte $ØØ

--------------------------------------------------------------------------------
  ADRESSBEREICH $CDBA-$CDC2 IST MIT $FF BELEGT! (Alte Betriebssystem-Version)
--------------------------------------------------------------------------------
. cdba  :
. cdba  *** DS$ ÜBERNEHMEN (TEIL 2) ***
. cdba  :
. cdba  a9 79     lda #$79     Richtet            (Neue
. cdbc  91 7a     sta ($7a),y  ... R-Zeiger        Betriebssystem-
. cdbe  a9 ØØ     lda #$ØØ     ... auf            Version)
. cdcØ  c8        iny          ... Deskriptor
. cdc1  91 7a     sta ($7a),y
. cdc2  6Ø        rts          ENDE

      ADRESSBEREICH $CDC3-$CDFF IST MIT $FF BELEGT!!!

. ceØØ  :
. ceØØ  *** INTERRUPT ***
. ceØØ  :
. ceØØ  ba        tsx          Prüft
. ceØ1  bd Ø4 Ø1 lda $Ø1Ø4,x  ... auf
. ceØ4  29 1Ø     and #$1Ø     ... Hardware-Interrupt
. ceØ6  dØ Ø3     bne $ceØb    Kein Hardware-Interrupt ?, dann $ceØb
. ceØ8  6c 14 Ø3 jmp ($Ø314)  Interrupt

. ceØb  6c 16 Ø3 jmp ($Ø316)  Monitor Break-Interrupt

. ceØe  ad Ø9 ff lda $ffØ9    Holt Interrupt-Request-Register
. ce11  29 Ø2     and #$Ø2     Raster-Interrupt ?
. ce13  fØ Ø3     beq $ce18    Nein, dann $ce18
. ce15  2Ø 6Ø ce jsr $ce6Ø    Grafik bearbeiten
. ce18  2c d8 Ø7 bit $Ø7d8    RS-232 in Betrieb ?
. ce1b  1Ø Øe     bpl $ce2b    Nein, dann $ce2b
. ce1d  ad Ø1 fd lda $fdØ1    Holt RS-232-Status
. ce2Ø  8d d4 Ø7 sta $Ø7d4
. ce23  1Ø Ø6     bpl $ce2b    Kein RS-232-Interrupt ?, dann $ce2b
. ce25  2Ø 95 ea jsr $ea95    Datenbyte empfangen
```

```
. ce28  20 5b ea jsr $ea5b    RS-232-Routine
. ce2b  20 e4 e3 jsr $e3e4    Setzt Timer im Kassettenbetrieb
. ce2e  ad 09 ff lda $ff09    Holt Interrupt-Request-Register
. ce31  29 02    and #$02     Raster-Interrupt ?
. ce33  f0 28    beq $ce5d    Nein, dann $ce5d
. ce35  8d 09 ff sta $ff09    Schreibt Interrupt-Request-Register zurück
. ce38  2c 0b ff bit $ff0b    Raster-Zeile
. ce3b  a9 cc    lda #$cc     Nächster Raster-Interrupt
. ce3d  50 1b    bvc $ce5a    Interrupt war bei Raster 161
. ce3f  6c 12 03 jmp ($0312)  Interrupt (Uhr)

. ce42  20 bf cf jsr $cfbf    Datasette bedienen und Uhr aktualisieren
. ce45  20 cd ce jsr $cecd    Ton bearbeiten
. ce48  a5 fb    lda $fb      Holt aktuelle Bank-Konfiguration
. ce4a  48       pha
. ce4b  a9 00    lda #$00     Schaltet ROM
. ce4d  85 fb    sta $fb      ... ein
. ce4f  08       php
. ce50  58       cli
. ce51  20 11 db jsr $db11    (SCNKEY) Tastaturabfrage
. ce54  28       plp
. ce55  68       pla          Bank-Konfiguration
. ce56  85 fb    sta $fb      ... wiederherstellen
. ce58  a9 a1    lda #$a1     Nächster Raster-Interrupt bei 161
. ce5a  8d 0b ff sta $ff0b    Interrupt vorbereiten
. ce5d  4c be fc jmp $fcbe    Aktuelle Bank-Konfiguration einschalten

. ce60  ad 1c ff lda $ff1c    Holt Raster-Zeile (Bit 8)
. ce63  29 01    and #$01     Bit 8 gesetzt ?
. ce65  d0 39    bne $cea0    Ja, dann Raster nicht sichtbar
. ce67  ad 1d ff lda $ff1d    Holt Raster-Zeile (Bit 0- Bit 7)
. ce6a  c9 a3    cmp #$a3     Raster noch im Grafik-Bereich ?
. ce6c  b0 2e    bcs $ce9c    Nein, dann $ce9c
. ce6e  24 83    bit $83      Grafik und Text ?
. ce70  50 52    bvc $cec4    Nein, dann fertig
. ce72  a9 08    lda #$08     Bild- und Farbspeicher
. ce74  8d 14 ff sta $ff14    ... einschalten
. ce77  ad 06 ff lda $ff06    HIRES
. ce7a  29 df    and #$df     ... ausschalten
. ce7c  a8       tay          Byte merken
. ce7d  ad 07 ff lda $ff07    Multicolor
. ce80  29 ef    and #$ef     ... ausschalten
. ce82  aa       tax          Byte merken
. ce83  ad 12 ff lda $ff12    Daten aus RAM/ROM
. ce86  0d fa 07 ora $07fa    ROM-Maske für geteilten Bildschirm
. ce89  48       pha          Byte auf Stack retten
. ce8a  ad 1d ff lda $ff1d    Raster-Zeile
. ce8d  c9 a3    cmp #$a3     ... noch im Grafikbereich ?
. ce8f  90 f9    bcc $ce8a    Ja, dann warten
. ce91  68       pla          Schaltet
. ce92  8d 12 ff sta $ff12    ... Grafik-
. ce95  8c 06 ff sty $ff06    ... Modus
```

```
. ce98  8e Ø7 ff stx $ffØ7     ... aus
. ce9b  6Ø        rts          ENDE

. ce9c  c9 cc     cmp #$cc     Raster-Zeile noch im Textbereich ?
. ce9e  9Ø 24     bcc $cec4    Ja, dann fertig
. ceaØ  a6 83     ldx $83      Holt Grafikmodus
. cea2  fØ 2Ø     beq $cec4    Grafik ausgeschaltet ?, dann $cec4
. cea4  1Ø Ø8     bpl $ceae    Nicht Multicolor-Grafik ?, dann $ceae
. cea6  ad Ø7 ff lda $ffØ7     Setzt
. cea9  Ø9 1Ø     ora #$1Ø     ... Multicolor-
. ceab  8d Ø7 ff sta $ffØ7     ... Freigabe-Bit
. ceae  ad Ø6 ff lda $ffØ6     Setzt
. ceb1  Ø9 2Ø     ora #$2Ø     ... HIRES-
. ceb3  8d Ø6 ff sta $ffØ6     ... Freigabe-Bit
. ceb6  ad 12 ff lda $ff12     Setzt Flag
. ceb9  29 fb     and #$fb     ... für Daten
. cebb  8d 12 ff sta $ff12     ... aus RAM
. cebe  ad fb Ø7 lda $Ø7fb     Basis-Adresse
. cec1  8d 14 ff sta $ff14     ... für Bild- und Farbspeicher setzen
. cec4  6Ø        rts          ENDE

. cec5  ea        nop
. cec6  ea        nop
. cec7  ea        nop
. cec8  ea        nop
. cec9  ea        nop
. ceca  ea        nop
. cecb  ea        nop
. cecc  ea        nop

. cecd  :
. cecd  *** TONGENERATOREN BEARBEITEN ***
. cecd  :
. cecd  a2 Ø1     ldx #$Ø1     Index für Tongeneratoren (1 oder 2)
. cecf  bd fc Ø4 lda $Ø4fc,x   Zähler von
. ced2  1d fe Ø4 ora $Ø4fe,x   ... Tongenerator (x) abgelaufen ?
. ced5  fØ 13     beq $ceea    Ja, dann $ceea
. ced7  fe fc Ø4 inc $Ø4fc,x   Zähler
. ceda  dØ Øe     bne $ceea    ... um 1
. cedc  fe fe Ø4 inc $Ø4fe,x   ... verringern
. cedf  dØ Ø9     bne $ceea    Noch nicht Ø ?, dann $ceea
. cee1  bd ee ce lda $ceee,x   Schaltet
. cee4  2d 11 ff and $ff11     ... Tongenerator (x)
. cee7  8d 11 ff sta $ff11     ... aus
. ceea  ca        dex          Falls erst ein Tongenerator
. ceeb  1Ø e2     bpl $cecf    ... bearbeitet wurde, mit 2. fortfahren
. ceed  6Ø        rts          ENDE

. ceee  ef 9f     .byte $ef, $9f

. cefØ  :
. cefØ  *** UHR TAKT ***
```

```
. cefØ  :
. cefØ  e6 a5    inc $a5      Systemuhr
. cef2  dØ Ø6    bne $cefa    ... um
. cef4  e6 a4    inc $a4      ... 1
. cef6  dØ Ø2    bne $cefa    ... erhöhen...
. cef8  e6 a3    inc $a3
. cefa  38       sec          Uhrzeit
. cefb  a5 a5    lda $a5      ... 24:ØØ:ØØ
. cefd  e9 Ø1    sbc #$Ø1     ... erreicht ?...
. ceff  a5 a4    lda $a4
. cfØ1  e9 1a    sbc #$1a
. cfØ3  a5 a3    lda $a3
. cfØ5  e9 4f    sbc #$4f
. cfØ7  9Ø Ø8    bcc $cf11    Nein, dann $cf11
. cfØ9  a2 ØØ    ldx #$ØØ     Uhr
. cfØb  86 a3    stx $a3      ... auf
. cfØd  86 a4    stx $a4      ... ØØ:ØØ:ØØ
. cfØf  86 a5    stx $a5      ... zurücksetzen

     STOP-TASTE ÜBERPRÜFEN

. cf11  a9 7f    lda #$7f     Abfrage des
. cf13  2Ø 7Ø db jsr $db7Ø    ... Tastatur-Decoders
. cf16  85 ee    sta $ee      Ergebnis zwischenspeichern
. cf18  a9 7f    lda #$7f     Abfrage des
. cf1a  2Ø 7Ø db jsr $db7Ø    ... Tastatur-Decoders
. cf1d  c5 ee    cmp $ee      Ergebnis gleich ?
. cf1f  dØ fØ    bne $cf11    Nein, dann Abfrage wiederholen
. cf21  Ø9 7f    ora #$7f     Setzt($7f)/löscht($ff)
. cf23  85 91    sta $91      ... Flag für STOP-Taste
. cf25  6Ø       rts          ENDE

. cf26  :
. cf26  *** UHR LESEN ***
. cf26  :
. cf26  78       sei
. cf27  a5 a5    lda $a5      Übergibt
. cf29  a6 a4    ldx $a4      ... Systemzeit
. cf2b  a4 a3    ldy $a3      ... an Akku, X-Reg., Y-Reg.
. cf2d  :
. cf2d  *** UHR STELLEN ***
. cf2d  :
. cf2d  78       sei
. cf2e  85 a5    sta $a5      Übernimmt
. cf3Ø  86 a4    stx $a4      ... Systemzeit
. cf32  84 a3    sty $a3      ... von Akku, X-Reg., Y-Reg.
. cf34  58       cli
. cf35  6Ø       rts          ENDE

. cf36  :
. cf36  *** MONITOR-MELDUNG ***
. cf36  :
```

```
. cf36  .byte $0d 'monitor' $8d
. cf3f  .byte $0d 'break'
. cf45  .byte $0d '   pc  sr ac xr yr sp'
. cf5b  .byte $0d ';' $a0
. cf5e  .byte 'a' $a0
. cf60  .byte ' erroR'

. cf66  :
. cf66  *** MONITOR-MELDUNG AUSGEBEN ***
. cf66  :
. cf66  bd 36 cf lda $cf36,x  Holt Zeichen aus Text
. cf69  08       php
. cf6a  29 7f    and #$7f     Löscht Bit 7
. cf6c  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. cf6f  e8       inx          Zeiger um 1 erhöhen
. cf70  28       plp
. cf71  10 f3    bpl $cf66    Noch nicht letztes Zeichen ?, dann weiter
. cf73  60       rts          ENDE

. cf74  :
. cf74  *** ABSCHLUSS DER ZEILENÜBERNAHME ***
. cf74  :
. cf74  a9 0d    lda #$0d
. cf76  a6 98    ldx $98      Eingabekanal gleich
. cf78  e0 03    cpx #$03     ... Bildschirm ?
. cf7a  f0 06    beq $cf82    Ja, dann $cf82
. cf7c  a6 99    ldx $99      Ausgabekanal gleich
. cf7e  e0 03    cpx #$03     ... Bildschirm ?
. cf80  f0 03    beq $cf85    Ja, dann $cf85
. cf82  20 49 dc jsr $dc49    PRINT
. cf85  a9 0d    lda #$0d
. cf87  4c b0 d9 jmp $d9b0    Register wiederherstellen

. cf8a  :
. cf8a  *** DIVERSE PROGRAMMTEILE ***
. cf8a  :
. cf8a  bd 13 01 lda $0113,x  Holt Farbcodes im RAM
. cf8d  2c f9 07 bit $07f9    Flag für Farbcodes im ROM gesetzt ?
. cf90  10 03    bpl $cf95    Nein, dann $cf95
. cf92  bd 43 e1 lda $e143,x  Holt Farbcodes im ROM
. cf95  60       rts          ENDE

. cf96  2c f8 07 bit $07f8    Flag für Laden aus RAM gesetzt ?
. cf99  30 03    bmi $cf9e    Ja, dann $cf9e
. cf9b  b1 a1    lda ($a1),y  Lädt auf ROM
. cf9d  60       rts          ENDE

. cf9e  a9 a1    lda #$a1     Ladeadresse ($a1)
. cfa0  8d df 07 sta $07df    ... einsetzen
. cfa3  4c d9 07 jmp $07d9    LDA($a1),Y)

. cfa6  a9 09    lda #$09     Teil von Reset
```

```
. cfa8  8d 20 fd sta $fd20
. cfab  09 80    ora #$80
. cfad  8d 20 fd sta $fd20
. cfb0  4c 1e fc jmp $fc1e    Initialisiert Module

     DIESE ROUTINE WIRD AB ADRESSE $07d9 INS RAM KOPIERT !!!

. cfb3  08       php
. cfb4  78       sei
. cfb5  8d 3f ff sta $ff3f    RAM einschalten
. cfb8  b1 00    lda ($00),y  Adresse ist variabel
. cfba  8d 3e ff sta $ff3e    ROM einschalten
. cfbd  28       plp
. cfbe  60       rts          ENDE

. cfbf  :
. cfbf  *** DATASETTE UND UHR BEDIENEN ***
. cfbf  :
. cfbf  ad 10 fd lda $fd10    PLAY-Taste
. cfc2  29 04    and #$04     ... gedrückt ?
. cfc4  d0 1b    bne $cfe1    Nein, dann $cfe1
. cfc6  2c fc 07 bit $07fc    Läuft Motor ?
. cfc9  30 06    bmi $cfd1    Ja, dann $cfd1
. cfcb  a5 01    lda $01      Schaltet
. cfcd  29 f7    and #$f7     ... Motor
. cfcf  85 01    sta $01      ... ein
. cfd1  ce fd 07 dec $07fd    Zähler für Uhr Takt
. cfd4  10 08    bpl $cfde    Innerhalb von 5 Interrupts
. cfd6  a9 04    lda #$04     ... muß die Uhr 6mal getaktet werden
. cfd8  8d fd 07 sta $07fd
. cfdb  20 f0 ce jsr $cef0    Uhr Takt
. cfde  4c f0 ce jmp $cef0    Uhr Takt

. cfe1  8d fc 07 sta $07fc    Initialisiert Motor-Flag
. cfe4  20 b0 e3 jsr $e3b0    Schaltet Motor aus
. cfe7  4c d1 cf jmp $cfd1    Schaltet Uhr

. cfea  e8       inx
. cfeb  8e c4 fe stx $fec4
. cfee  8e c0 fe stx $fec0
. cff1  a9 80    lda #$80
. cff3  8d 11 ff sta $ff11
. cff6  60       rts          ENDE

     DER BEREICH $CFF7-$CFFF IST MIT $FF BELEGT
```

```
. d000  :
. d000  *** ZEICHENSATZ ***
. d000  :
. d000                                  Zeichen: 0 Klammeraffe
. d000  3c      .byte % 00111100
. d001  66      .byte % 01100110
. d002  6e      .byte % 01101110
. d003  6e      .byte % 01101110
. d004  60      .byte % 01100000
. d005  62      .byte % 01100010
. d006  3c      .byte % 00111100
. d007  00      .byte % 00000000
. d008                                  Zeichen: 1 'A'
. d008  18      .byte % 00011000
. d009  3c      .byte % 00111100
. d00a  66      .byte % 01100110
. d00b  7e      .byte % 01111110
. d00c  66      .byte % 01100110
. d00d  66      .byte % 01100110
. d00e  66      .byte % 01100110
. d00f  00      .byte % 00000000
. d010                                  Zeichen: 2 'B'
. d010  7c      .byte % 01111100
. d011  66      .byte % 01100110
. d012  66      .byte % 01100110
. d013  7c      .byte % 01111100
. d014  66      .byte % 01100110
. d015  66      .byte % 01100110
. d016  7c      .byte % 01111100
. d017  00      .byte % 00000000
. d018                                  Zeichen: 3 'C'
. d018  3c      .byte % 00111100
. d019  66      .byte % 01100110
. d01a  60      .byte % 01100000
. d01b  60      .byte % 01100000
. d01c  60      .byte % 01100000
. d01d  66      .byte % 01100110
. d01e  3c      .byte % 00111100
. d01f  00      .byte % 00000000
. d020                                  Zeichen: 4 'D'
. d020  78      .byte % 01111000
. d021  6c      .byte % 01101100
. d022  66      .byte % 01100110
. d023  66      .byte % 01100110
. d024  66      .byte % 01100110
. d025  6c      .byte % 01101100
. d026  78      .byte % 01111000
. d027  00      .byte % 00000000
. d028                                  Zeichen: 5 'E'
. d028  7e      .byte % 01111110
. d029  60      .byte % 01100000
. d02a  60      .byte % 01100000
```

```
. d02b  78      .byte % 01111000
. d02c  60      .byte % 01100000
. d02d  60      .byte % 01100000
. d02e  7e      .byte % 01111110
. d02f  00      .byte % 00000000
. d030                          Zeichen: 6 'F'
. d030  7e      .byte % 01111110
. d031  60      .byte % 01100000
. d032  60      .byte % 01100000
. d033  78      .byte % 01111000
. d034  60      .byte % 01100000
. d035  60      .byte % 01100000
. d036  60      .byte % 01100000
. d037  00      .byte % 00000000
. d038                          Zeichen: 7 'G'
. d038  3c      .byte % 00111100
. d039  66      .byte % 01100110
. d03a  60      .byte % 01100000
. d03b  6e      .byte % 01101110
. d03c  66      .byte % 01100110
. d03d  66      .byte % 01100110
. d03e  3c      .byte % 00111100
. d03f  00      .byte % 00000000
. d040                          Zeichen: 8 'H'
. d040  66      .byte % 01100110
. d041  66      .byte % 01100110
. d042  66      .byte % 01100110
. d043  7e      .byte % 01111110
. d044  66      .byte % 01100110
. d045  66      .byte % 01100110
. d046  66      .byte % 01100110
. d047  00      .byte % 00000000
. d048                          Zeichen: 9 'I'
. d048  3c      .byte % 00111100
. d049  18      .byte % 00011000
. d04a  18      .byte % 00011000
. d04b  18      .byte % 00011000
. d04c  18      .byte % 00011000
. d04d  18      .byte % 00011000
. d04e  3c      .byte % 00111100
. d04f  00      .byte % 00000000
. d050                          Zeichen: 10 'J'
. d050  1e      .byte % 00011110
. d051  0c      .byte % 00001100
. d052  0c      .byte % 00001100
. d053  0c      .byte % 00001100
. d054  0c      .byte % 00001100
. d055  6c      .byte % 01101100
. d056  38      .byte % 00111000
. d057  00      .byte % 00000000
. d058                          Zeichen: 11 'K'
. d058  66      .byte % 01100110
```

```
. d059  6c       .byte % 01101100
. d05a  78       .byte % 01111000
. d05b  70       .byte % 01110000
. d05c  78       .byte % 01111000
. d05d  6c       .byte % 01101100
. d05e  66       .byte % 01100110
. d05f  00       .byte % 00000000
. d060                             Zeichen: 12 'L'
. d060  60       .byte % 01100000
. d061  60       .byte % 01100000
. d062  60       .byte % 01100000
. d063  60       .byte % 01100000
. d064  60       .byte % 01100000
. d065  60       .byte % 01100000
. d066  7e       .byte % 01111110
. d067  00       .byte % 00000000
. d068                             Zeichen: 13 'M'
. d068  63       .byte % 01100011
. d069  77       .byte % 01110111
. d06a  7f       .byte % 01111111
. d06b  6b       .byte % 01101011
. d06c  63       .byte % 01100011
. d06d  63       .byte % 01100011
. d06e  63       .byte % 01100011
. d06f  00       .byte % 00000000
. d070                             Zeichen: 14 'N'
. d070  66       .byte % 01100110
. d071  76       .byte % 01110110
. d072  7e       .byte % 01111110
. d073  7e       .byte % 01111110
. d074  6e       .byte % 01101110
. d075  66       .byte % 01100110
. d076  66       .byte % 01100110
. d077  00       .byte % 00000000
. d078                             Zeichen: 15 'O'
. d078  3c       .byte % 00111100
. d079  66       .byte % 01100110
. d07a  66       .byte % 01100110
. d07b  66       .byte % 01100110
. d07c  66       .byte % 01100110
. d07d  66       .byte % 01100110
. d07e  3c       .byte % 00111100
. d07f  00       .byte % 00000000
. d080                             Zeichen: 16 'P'
. d080  7c       .byte % 01111100
. d081  66       .byte % 01100110
. d082  66       .byte % 01100110
. d083  7c       .byte % 01111100
. d084  60       .byte % 01100000
. d085  60       .byte % 01100000
. d086  60       .byte % 01100000
. d087  00       .byte % 00000000
```

```
. d088                                  Zeichen: 17 'Q'
. d088  3c      .byte % 00111100
. d089  66      .byte % 01100110
. d08a  66      .byte % 01100110
. d08b  66      .byte % 01100110
. d08c  66      .byte % 01100110
. d08d  3c      .byte % 00111100
. d08e  0e      .byte % 00001110
. d08f  00      .byte % 00000000
. d090                                  Zeichen: 18 'R'
. d090  7c      .byte % 01111100
. d091  66      .byte % 01100110
. d092  66      .byte % 01100110
. d093  7c      .byte % 01111100
. d094  78      .byte % 01111000
. d095  6c      .byte % 01101100
. d096  66      .byte % 01100110
. d097  00      .byte % 00000000
. d098                                  Zeichen: 19 'S'
. d098  3c      .byte % 00111100
. d099  66      .byte % 01100110
. d09a  60      .byte % 01100000
. d09b  3c      .byte % 00111100
. d09c  06      .byte % 00000110
. d09d  66      .byte % 01100110
. d09e  3c      .byte % 00111100
. d09f  00      .byte % 00000000
. d0a0                                  Zeichen: 20 'T'
. d0a0  7e      .byte % 01111110
. d0a1  18      .byte % 00011000
. d0a2  18      .byte % 00011000
. d0a3  18      .byte % 00011000
. d0a4  18      .byte % 00011000
. d0a5  18      .byte % 00011000
. d0a6  18      .byte % 00011000
. d0a7  00      .byte % 00000000
. d0a8                                  Zeichen: 21 'U'
. d0a8  66      .byte % 01100110
. d0a9  66      .byte % 01100110
. d0aa  66      .byte % 01100110
. d0ab  66      .byte % 01100110
. d0ac  66      .byte % 01100110
. d0ad  66      .byte % 01100110
. d0ae  3c      .byte % 00111100
. d0af  00      .byte % 00000000
. d0b0                                  Zeichen: 22 'V'
. d0b0  66      .byte % 01100110
. d0b1  66      .byte % 01100110
. d0b2  66      .byte % 01100110
. d0b3  66      .byte % 01100110
. d0b4  66      .byte % 01100110
. d0b5  3c      .byte % 00111100
```

```
. d0b6  18      .byte % 00011000
. d0b7  00      .byte % 00000000
. d0b8                          Zeichen: 23 'W'
. d0b8  63      .byte % 01100011
. d0b9  63      .byte % 01100011
. d0ba  63      .byte % 01100011
. d0bb  6b      .byte % 01101011
. d0bc  7f      .byte % 01111111
. d0bd  77      .byte % 01110111
. d0be  63      .byte % 01100011
. d0bf  00      .byte % 00000000
. d0c0                          Zeichen: 24 'X'
. d0c0  66      .byte % 01100110
. d0c1  66      .byte % 01100110
. d0c2  3c      .byte % 00111100
. d0c3  18      .byte % 00011000
. d0c4  3c      .byte % 00111100
. d0c5  66      .byte % 01100110
. d0c6  66      .byte % 01100110
. d0c7  00      .byte % 00000000
. d0c8                          Zeichen: 25 'Y'
. d0c8  66      .byte % 01100110
. d0c9  66      .byte % 01100110
. d0ca  66      .byte % 01100110
. d0cb  3c      .byte % 00111100
. d0cc  18      .byte % 00011000
. d0cd  18      .byte % 00011000
. d0ce  18      .byte % 00011000
. d0cf  00      .byte % 00000000
. d0d0                          Zeichen: 26 'Z'
. d0d0  7e      .byte % 01111110
. d0d1  06      .byte % 00000110
. d0d2  0c      .byte % 00001100
. d0d3  18      .byte % 00011000
. d0d4  30      .byte % 00110000
. d0d5  60      .byte % 01100000
. d0d6  7e      .byte % 01111110
. d0d7  00      .byte % 00000000
. d0d8                          Zeichen: 27 Eckige Klammer
. d0d8  3c      .byte % 00111100
. d0d9  30      .byte % 00110000
. d0da  30      .byte % 00110000
. d0db  30      .byte % 00110000
. d0dc  30      .byte % 00110000
. d0dd  30      .byte % 00110000
. d0de  3c      .byte % 00111100
. d0df  00      .byte % 00000000
. d0e0                          Zeichen: 28 Pfund
. d0e0  0c      .byte % 00001100
. d0e1  12      .byte % 00010010
. d0e2  30      .byte % 00110000
. d0e3  7c      .byte % 01111100
```

```
. d0e4  30      .byte %00110000
. d0e5  62      .byte %01100010
. d0e6  fc      .byte %11111100
. d0e7  00      .byte %00000000
. d0e8                          Zeichen: 29 Eckige Klammer
. d0e8  3c      .byte %00111100
. d0e9  0c      .byte %00001100
. d0ea  0c      .byte %00001100
. d0eb  0c      .byte %00001100
. d0ec  0c      .byte %00001100
. d0ed  0c      .byte %00001100
. d0ee  3c      .byte %00111100
. d0ef  00      .byte %00000000
. d0f0                          Zeichen: 30 Pfeil
. d0f0  00      .byte %00000000
. d0f1  18      .byte %00011000
. d0f2  3c      .byte %00111100
. d0f3  7e      .byte %01111110
. d0f4  18      .byte %00011000
. d0f5  18      .byte %00011000
. d0f6  18      .byte %00011000
. d0f7  18      .byte %00011000
. d0f8                          Zeichen: 31 ''
. d0f8  00      .byte %00000000
. d0f9  10      .byte %00010000
. d0fa  30      .byte %00110000
. d0fb  7f      .byte %01111111
. d0fc  7f      .byte %01111111
. d0fd  30      .byte %00110000
. d0fe  10      .byte %00010000
. d0ff  00      .byte %00000000
. d100                          Zeichen: 32 SPACE
. d100  00      .byte %00000000
. d101  00      .byte %00000000
. d102  00      .byte %00000000
. d103  00      .byte %00000000
. d104  00      .byte %00000000
. d105  00      .byte %00000000
. d106  00      .byte %00000000
. d107  00      .byte %00000000
. d108                          Zeichen: 33 '!'
. d108  18      .byte %00011000
. d109  18      .byte %00011000
. d10a  18      .byte %00011000
. d10b  18      .byte %00011000
. d10c  00      .byte %00000000
. d10d  00      .byte %00000000
. d10e  18      .byte %00011000
. d10f  00      .byte %00000000
. d110                          Zeichen: 34 '"'
. d110  66      .byte %01100110
. d111  66      .byte %01100110
```

```
. d112  66       .byte % 01100110
. d113  00       .byte % 00000000
. d114  00       .byte % 00000000
. d115  00       .byte % 00000000
. d116  00       .byte % 00000000
. d117  00       .byte % 00000000
. d118                           Zeichen: 35 '#'
. d118  66       .byte % 01100110
. d119  66       .byte % 01100110
. d11a  ff       .byte % 11111111
. d11b  66       .byte % 01100110
. d11c  ff       .byte % 11111111
. d11d  66       .byte % 01100110
. d11e  66       .byte % 01100110
. d11f  00       .byte % 00000000
. d120                           Zeichen: 36 '$'
. d120  18       .byte % 00011000
. d121  3e       .byte % 00111110
. d122  60       .byte % 01100000
. d123  3c       .byte % 00111100
. d124  06       .byte % 00000110
. d125  7c       .byte % 01111100
. d126  18       .byte % 00011000
. d127  00       .byte % 00000000
. d128                           Zeichen: 37 '%'
. d128  62       .byte % 01100010
. d129  66       .byte % 01100110
. d12a  0c       .byte % 00001100
. d12b  18       .byte % 00011000
. d12c  30       .byte % 00110000
. d12d  66       .byte % 01100110
. d12e  46       .byte % 01000110
. d12f  00       .byte % 00000000
. d130                           Zeichen: 38 '&'
. d130  3c       .byte % 00111100
. d131  66       .byte % 01100110
. d132  3c       .byte % 00111100
. d133  38       .byte % 00111000
. d134  67       .byte % 01100111
. d135  66       .byte % 01100110
. d136  3f       .byte % 00111111
. d137  00       .byte % 00000000
. d138                           Zeichen: 39 '''
. d138  06       .byte % 00000110
. d139  0c       .byte % 00001100
. d13a  18       .byte % 00011000
. d13b  00       .byte % 00000000
. d13c  00       .byte % 00000000
. d13d  00       .byte % 00000000
. d13e  00       .byte % 00000000
. d13f  00       .byte % 00000000
. d140                           Zeichen: 40 '('
```

```
. d140  0c      .byte % 00001100
. d141  18      .byte % 00011000
. d142  30      .byte % 00110000
. d143  30      .byte % 00110000
. d144  30      .byte % 00110000
. d145  18      .byte % 00011000
. d146  0c      .byte % 00001100
. d147  00      .byte % 00000000
. d148                          Zeichen: 41 ')'
. d148  30      .byte % 00110000
. d149  18      .byte % 00011000
. d14a  0c      .byte % 00001100
. d14b  0c      .byte % 00001100
. d14c  0c      .byte % 00001100
. d14d  18      .byte % 00011000
. d14e  30      .byte % 00110000
. d14f  00      .byte % 00000000
. d150                          Zeichen: 42 '*'
. d150  00      .byte % 00000000
. d151  66      .byte % 01100110
. d152  3c      .byte % 00111100
. d153  ff      .byte % 11111111
. d154  3c      .byte % 00111100
. d155  66      .byte % 01100110
. d156  00      .byte % 00000000
. d157  00      .byte % 00000000
. d158                          Zeichen: 43 '+'
. d158  00      .byte % 00000000
. d159  18      .byte % 00011000
. d15a  18      .byte % 00011000
. d15b  7e      .byte % 01111110
. d15c  18      .byte % 00011000
. d15d  18      .byte % 00011000
. d15e  00      .byte % 00000000
. d15f  00      .byte % 00000000
. d160                          Zeichen: 44 ','
. d160  00      .byte % 00000000
. d161  00      .byte % 00000000
. d162  00      .byte % 00000000
. d163  00      .byte % 00000000
. d164  00      .byte % 00000000
. d165  18      .byte % 00011000
. d166  18      .byte % 00011000
. d167  30      .byte % 00110000
. d168                          Zeichen: 45 '-'
. d168  00      .byte % 00000000
. d169  00      .byte % 00000000
. d16a  00      .byte % 00000000
. d16b  7e      .byte % 01111110
. d16c  00      .byte % 00000000
. d16d  00      .byte % 00000000
. d16e  00      .byte % 00000000
```

```
. d16f  00      .byte % 00000000
. d170                          Zeichen: 46 '.'
. d170  00      .byte % 00000000
. d171  00      .byte % 00000000
. d172  00      .byte % 00000000
. d173  00      .byte % 00000000
. d174  00      .byte % 00000000
. d175  18      .byte % 00011000
. d176  18      .byte % 00011000
. d177  00      .byte % 00000000
. d178                          Zeichen: 47 '/'
. d178  00      .byte % 00000000
. d179  03      .byte % 00000011
. d17a  06      .byte % 00000110
. d17b  0c      .byte % 00001100
. d17c  18      .byte % 00011000
. d17d  30      .byte % 00110000
. d17e  60      .byte % 01100000
. d17f  00      .byte % 00000000
. d180                          Zeichen: 48 '0'
. d180  3c      .byte % 00111100
. d181  66      .byte % 01100110
. d182  6e      .byte % 01101110
. d183  76      .byte % 01110110
. d184  66      .byte % 01100110
. d185  66      .byte % 01100110
. d186  3c      .byte % 00111100
. d187  00      .byte % 00000000
. d188                          Zeichen: 49 '1'
. d188  18      .byte % 00011000
. d189  18      .byte % 00011000
. d18a  38      .byte % 00111000
. d18b  18      .byte % 00011000
. d18c  18      .byte % 00011000
. d18d  18      .byte % 00011000
. d18e  7e      .byte % 01111110
. d18f  00      .byte % 00000000
. d190                          Zeichen: 50 '2'
. d190  3c      .byte % 00111100
. d191  66      .byte % 01100110
. d192  06      .byte % 00000110
. d193  0c      .byte % 00001100
. d194  30      .byte % 00110000
. d195  60      .byte % 01100000
. d196  7e      .byte % 01111110
. d197  00      .byte % 00000000
. d198                          Zeichen: 51 '3'
. d198  3c      .byte % 00111100
. d199  66      .byte % 01100110
. d19a  06      .byte % 00000110
. d19b  1c      .byte % 00011100
. d19c  06      .byte % 00000110
```

```
. d19d  66        .byte % Ø11ØØ11Ø
. d19e  3c        .byte % ØØ1111ØØ
. d19f  ØØ        .byte % ØØØØØØØØ
. d1aØ                              Zeichen: 52 '4'
. d1aØ  Ø6        .byte % ØØØØØ11Ø
. d1a1  Øe        .byte % ØØØØ111Ø
. d1a2  1e        .byte % ØØØ1111Ø
. d1a3  66        .byte % Ø11ØØ11Ø
. d1a4  7f        .byte % Ø1111111
. d1a5  Ø6        .byte % ØØØØØ11Ø
. d1a6  Ø6        .byte % ØØØØØ11Ø
. d1a7  ØØ        .byte % ØØØØØØØØ
. d1a8                              Zeichen: 53 '5'
. d1a8  7e        .byte % Ø1111110
. d1a9  6Ø        .byte % Ø11ØØØØØ
. d1aa  7c        .byte % Ø11111ØØ
. d1ab  Ø6        .byte % ØØØØØ11Ø
. d1ac  Ø6        .byte % ØØØØØ11Ø
. d1ad  66        .byte % Ø11ØØ11Ø
. d1ae  3c        .byte % ØØ1111ØØ
. d1af  ØØ        .byte % ØØØØØØØØ
. d1bØ                              Zeichen: 54 '6'
. d1bØ  3c        .byte % ØØ1111ØØ
. d1b1  66        .byte % Ø11ØØ11Ø
. d1b2  6Ø        .byte % Ø11ØØØØØ
. d1b3  7c        .byte % Ø11111ØØ
. d1b4  66        .byte % Ø11ØØ11Ø
. d1b5  66        .byte % Ø11ØØ11Ø
. d1b6  3c        .byte % ØØ1111ØØ
. d1b7  ØØ        .byte % ØØØØØØØØ
. d1b8                              Zeichen: 55 '7'
. d1b8  7e        .byte % Ø1111110
. d1b9  66        .byte % Ø11ØØ11Ø
. d1ba  Øc        .byte % ØØØØ11ØØ
. d1bb  18        .byte % ØØØ11ØØØ
. d1bc  18        .byte % ØØØ11ØØØ
. d1bd  18        .byte % ØØØ11ØØØ
. d1be  18        .byte % ØØØ11ØØØ
. d1bf  ØØ        .byte % ØØØØØØØØ
. d1cØ                              Zeichen: 56 '8'
. d1cØ  3c        .byte % ØØ1111ØØ
. d1c1  66        .byte % Ø11ØØ11Ø
. d1c2  66        .byte % Ø11ØØ11Ø
. d1c3  3c        .byte % ØØ1111ØØ
. d1c4  66        .byte % Ø11ØØ11Ø
. d1c5  66        .byte % Ø11ØØ11Ø
. d1c6  3c        .byte % ØØ1111ØØ
. d1c7  ØØ        .byte % ØØØØØØØØ
. d1c8                              Zeichen: 57 '9'
. d1c8  3c        .byte % ØØ1111ØØ
. d1c9  66        .byte % Ø11ØØ11Ø
. d1ca  66        .byte % Ø11ØØ11Ø
```

```
. d1cb  3e      .byte % 00111110
. d1cc  06      .byte % 00000110
. d1cd  66      .byte % 01100110
. d1ce  3c      .byte % 00111100
. d1cf  00      .byte % 00000000
. d1d0                          Zeichen: 58 ':'
. d1d0  00      .byte % 00000000
. d1d1  00      .byte % 00000000
. d1d2  18      .byte % 00011000
. d1d3  00      .byte % 00000000
. d1d4  00      .byte % 00000000
. d1d5  18      .byte % 00011000
. d1d6  00      .byte % 00000000
. d1d7  00      .byte % 00000000
. d1d8                          Zeichen: 59 ';'
. d1d8  00      .byte % 00000000
. d1d9  00      .byte % 00000000
. d1da  18      .byte % 00011000
. d1db  00      .byte % 00000000
. d1dc  00      .byte % 00000000
. d1dd  18      .byte % 00011000
. d1de  18      .byte % 00011000
. d1df  30      .byte % 00110000
. d1e0                          Zeichen: 60 <
. d1e0  0e      .byte % 00001110
. d1e1  18      .byte % 00011000
. d1e2  30      .byte % 00110000
. d1e3  60      .byte % 01100000
. d1e4  30      .byte % 00110000
. d1e5  18      .byte % 00011000
. d1e6  0e      .byte % 00001110
. d1e7  00      .byte % 00000000
. d1e8                          Zeichen: 61 '='
. d1e8  00      .byte % 00000000
. d1e9  00      .byte % 00000000
. d1ea  7e      .byte % 01111110
. d1eb  00      .byte % 00000000
. d1ec  7e      .byte % 01111110
. d1ed  00      .byte % 00000000
. d1ee  00      .byte % 00000000
. d1ef  00      .byte % 00000000
. d1f0                          Zeichen: 62 >
. d1f0  70      .byte % 01110000
. d1f1  18      .byte % 00011000
. d1f2  0c      .byte % 00001100
. d1f3  06      .byte % 00000110
. d1f4  0c      .byte % 00001100
. d1f5  18      .byte % 00011000
. d1f6  70      .byte % 01110000
. d1f7  00      .byte % 00000000
. d1f8                          Zeichen: 63 '?'
. d1f8  3c      .byte % 00111100
```

```
. d1f9  66      .byte % 01100110
. d1fa  06      .byte % 00000110
. d1fb  0c      .byte % 00001100
. d1fc  18      .byte % 00011000
. d1fd  00      .byte % 00000000
. d1fe  18      .byte % 00011000
. d1ff  00      .byte % 00000000
. d200                          Zeichen: 64
. d200  00      .byte % 00000000
. d201  00      .byte % 00000000
. d202  00      .byte % 00000000
. d203  ff      .byte % 11111111
. d204  ff      .byte % 11111111
. d205  00      .byte % 00000000
. d206  00      .byte % 00000000
. d207  00      .byte % 00000000
. d208                          Zeichen: 65
. d208  08      .byte % 00001000
. d209  1c      .byte % 00011100
. d20a  3e      .byte % 00111110
. d20b  7f      .byte % 01111111
. d20c  7f      .byte % 01111111
. d20d  1c      .byte % 00011100
. d20e  3e      .byte % 00111110
. d20f  00      .byte % 00000000
. d210                          Zeichen: 66
. d210  18      .byte % 00011000
. d211  18      .byte % 00011000
. d212  18      .byte % 00011000
. d213  18      .byte % 00011000
. d214  18      .byte % 00011000
. d215  18      .byte % 00011000
. d216  18      .byte % 00011000
. d217  18      .byte % 00011000
. d218                          Zeichen: 67
. d218  00      .byte % 00000000
. d219  00      .byte % 00000000
. d21a  00      .byte % 00000000
. d21b  ff      .byte % 11111111
. d21c  ff      .byte % 11111111
. d21d  00      .byte % 00000000
. d21e  00      .byte % 00000000
. d21f  00      .byte % 00000000
. d220                          Zeichen: 68
. d220  00      .byte % 00000000
. d221  00      .byte % 00000000
. d222  ff      .byte % 11111111
. d223  ff      .byte % 11111111
. d224  00      .byte % 00000000
. d225  00      .byte % 00000000
. d226  00      .byte % 00000000
. d227  00      .byte % 00000000
```

```
. d228                                  Zeichen: 69
. d228  ØØ      .byte % ØØØØØØØØ
. d229  ff      .byte % 11111111
. d22a  ff      .byte % 11111111
. d22b  ØØ      .byte % ØØØØØØØØ
. d22c  ØØ      .byte % ØØØØØØØØ
. d22d  ØØ      .byte % ØØØØØØØØ
. d22e  ØØ      .byte % ØØØØØØØØ
. d22f  ØØ      .byte % ØØØØØØØØ
. d23Ø                                  Zeichen: 7Ø
. d23Ø  ØØ      .byte % ØØØØØØØØ
. d231  ØØ      .byte % ØØØØØØØØ
. d232  ØØ      .byte % ØØØØØØØØ
. d233  ØØ      .byte % ØØØØØØØØ
. d234  ff      .byte % 11111111
. d235  ff      .byte % 11111111
. d236  ØØ      .byte % ØØØØØØØØ
. d237  ØØ      .byte % ØØØØØØØØ
. d238                                  Zeichen: 71
. d238  3Ø      .byte % ØØ11ØØØØ
. d239  3Ø      .byte % ØØ11ØØØØ
. d23a  3Ø      .byte % ØØ11ØØØØ
. d23b  3Ø      .byte % ØØ11ØØØØ
. d23c  3Ø      .byte % ØØ11ØØØØ
. d23d  3Ø      .byte % ØØ11ØØØØ
. d23e  3Ø      .byte % ØØ11ØØØØ
. d23f  3Ø      .byte % ØØ11ØØØØ
. d24Ø                                  Zeichen: 72
. d24Ø  Øc      .byte % ØØØØ11ØØ
. d241  Øc      .byte % ØØØØ11ØØ
. d242  Øc      .byte % ØØØØ11ØØ
. d243  Øc      .byte % ØØØØ11ØØ
. d244  Øc      .byte % ØØØØ11ØØ
. d245  Øc      .byte % ØØØØ11ØØ
. d246  Øc      .byte % ØØØØ11ØØ
. d247  Øc      .byte % ØØØØ11ØØ
. d248                                  Zeichen: 73
. d248  ØØ      .byte % ØØØØØØØØ
. d249  ØØ      .byte % ØØØØØØØØ
. d24a  ØØ      .byte % ØØØØØØØØ
. d24b  eØ      .byte % 111ØØØØØ
. d24c  fØ      .byte % 1111ØØØØ
. d24d  38      .byte % ØØ111ØØØ
. d24e  18      .byte % ØØØ11ØØØ
. d24f  18      .byte % ØØØ11ØØØ
. d25Ø                                  Zeichen: 74
. d25Ø  18      .byte % ØØØ11ØØØ
. d251  18      .byte % ØØØ11ØØØ
. d252  1c      .byte % ØØØ111ØØ
. d253  Øf      .byte % ØØØØ1111
. d254  Ø7      .byte % ØØØØ0111
. d255  ØØ      .byte % ØØØØØØØØ
```

```
. d256  00      .byte % 00000000
. d257  00      .byte % 00000000
. d258                          Zeichen: 75
. d258  18      .byte % 00011000
. d259  18      .byte % 00011000
. d25a  38      .byte % 00111000
. d25b  f0      .byte % 11110000
. d25c  e0      .byte % 11100000
. d25d  00      .byte % 00000000
. d25e  00      .byte % 00000000
. d25f  00      .byte % 00000000
. d260                          Zeichen: 76
. d260  c0      .byte % 11000000
. d261  c0      .byte % 11000000
. d262  c0      .byte % 11000000
. d263  c0      .byte % 11000000
. d264  c0      .byte % 11000000
. d265  c0      .byte % 11000000
. d266  ff      .byte % 11111111
. d267  ff      .byte % 11111111
. d268                          Zeichen: 77
. d268  c0      .byte % 11000000
. d269  e0      .byte % 11100000
. d26a  70      .byte % 01110000
. d26b  38      .byte % 00111000
. d26c  1c      .byte % 00011100
. d26d  0e      .byte % 00001110
. d26e  07      .byte % 00000111
. d26f  03      .byte % 00000011
. d270                          Zeichen: 78
. d270  03      .byte % 00000011
. d271  07      .byte % 00000111
. d272  0e      .byte % 00001110
. d273  1c      .byte % 00011100
. d274  38      .byte % 00111000
. d275  70      .byte % 01110000
. d276  e0      .byte % 11100000
. d277  c0      .byte % 11000000
. d278                          Zeichen: 79
. d278  ff      .byte % 11111111
. d279  ff      .byte % 11111111
. d27a  c0      .byte % 11000000
. d27b  c0      .byte % 11000000
. d27c  c0      .byte % 11000000
. d27d  c0      .byte % 11000000
. d27e  c0      .byte % 11000000
. d27f  c0      .byte % 11000000
. d280                          Zeichen: 80
. d280  ff      .byte % 11111111
. d281  ff      .byte % 11111111
. d282  03      .byte % 00000011
. d283  03      .byte % 00000011
```

```
. d284  03      .byte % 00000011
. d285  03      .byte % 00000011
. d286  03      .byte % 00000011
. d287  03      .byte % 00000011
. d288                          Zeichen: 81
. d288  00      .byte % 00000000
. d289  3c      .byte % 00111100
. d28a  7e      .byte % 01111110
. d28b  7e      .byte % 01111110
. d28c  7e      .byte % 01111110
. d28d  7e      .byte % 01111110
. d28e  3c      .byte % 00111100
. d28f  00      .byte % 00000000
. d290                          Zeichen: 82
. d290  00      .byte % 00000000
. d291  00      .byte % 00000000
. d292  00      .byte % 00000000
. d293  00      .byte % 00000000
. d294  00      .byte % 00000000
. d295  ff      .byte % 11111111
. d296  ff      .byte % 11111111
. d297  00      .byte % 00000000
. d298                          Zeichen: 83
. d298  36      .byte % 00110110
. d299  7f      .byte % 01111111
. d29a  7f      .byte % 01111111
. d29b  7f      .byte % 01111111
. d29c  3e      .byte % 00111110
. d29d  1c      .byte % 00011100
. d29e  08      .byte % 00001000
. d29f  00      .byte % 00000000
. d2a0                          Zeichen: 84
. d2a0  60      .byte % 01100000
. d2a1  60      .byte % 01100000
. d2a2  60      .byte % 01100000
. d2a3  60      .byte % 01100000
. d2a4  60      .byte % 01100000
. d2a5  60      .byte % 01100000
. d2a6  60      .byte % 01100000
. d2a7  60      .byte % 01100000
. d2a8                          Zeichen: 85
. d2a8  00      .byte % 00000000
. d2a9  00      .byte % 00000000
. d2aa  00      .byte % 00000000
. d2ab  07      .byte % 00000111
. d2ac  0f      .byte % 00001111
. d2ad  1c      .byte % 00011100
. d2ae  18      .byte % 00011000
. d2af  18      .byte % 00011000
. d2b0                          Zeichen: 86
. d2b0  c3      .byte % 11000011
. d2b1  e7      .byte % 11100111
```

```
. d2b2  7e      .byte %01111110
. d2b3  3c      .byte %00111100
. d2b4  3c      .byte %00111100
. d2b5  7e      .byte %01111110
. d2b6  e7      .byte %11100111
. d2b7  c3      .byte %11000011
. d2b8                          Zeichen: 87
. d2b8  00      .byte %00000000
. d2b9  3c      .byte %00111100
. d2ba  7e      .byte %01111110
. d2bb  66      .byte %01100110
. d2bc  66      .byte %01100110
. d2bd  7e      .byte %01111110
. d2be  3c      .byte %00111100
. d2bf  00      .byte %00000000
. d2c0                          Zeichen: 88
. d2c0  18      .byte %00011000
. d2c1  18      .byte %00011000
. d2c2  66      .byte %01100110
. d2c3  66      .byte %01100110
. d2c4  18      .byte %00011000
. d2c5  18      .byte %00011000
. d2c6  3c      .byte %00111100
. d2c7  00      .byte %00000000
. d2c8                          Zeichen: 89
. d2c8  06      .byte %00000110
. d2c9  06      .byte %00000110
. d2ca  06      .byte %00000110
. d2cb  06      .byte %00000110
. d2cc  06      .byte %00000110
. d2cd  06      .byte %00000110
. d2ce  06      .byte %00000110
. d2cf  06      .byte %00000110
. d2d0                          Zeichen: 90
. d2d0  08      .byte %00001000
. d2d1  1c      .byte %00011100
. d2d2  3e      .byte %00111110
. d2d3  7f      .byte %01111111
. d2d4  3e      .byte %00111110
. d2d5  1c      .byte %00011100
. d2d6  08      .byte %00001000
. d2d7  00      .byte %00000000
. d2d8                          Zeichen: 91
. d2d8  18      .byte %00011000
. d2d9  18      .byte %00011000
. d2da  18      .byte %00011000
. d2db  ff      .byte %11111111
. d2dc  ff      .byte %11111111
. d2dd  18      .byte %00011000
. d2de  18      .byte %00011000
. d2df  18      .byte %00011000
. d2e0                          Zeichen: 92
```

```
. d2e0  c0      .byte % 11000000
. d2e1  c0      .byte % 11000000
. d2e2  30      .byte % 00110000
. d2e3  30      .byte % 00110000
. d2e4  c0      .byte % 11000000
. d2e5  c0      .byte % 11000000
. d2e6  30      .byte % 00110000
. d2e7  30      .byte % 00110000
. d2e8                          Zeichen: 93
. d2e8  18      .byte % 00011000
. d2e9  18      .byte % 00011000
. d2ea  18      .byte % 00011000
. d2eb  18      .byte % 00011000
. d2ec  18      .byte % 00011000
. d2ed  18      .byte % 00011000
. d2ee  18      .byte % 00011000
. d2ef  18      .byte % 00011000
. d2f0                          Zeichen: 94
. d2f0  00      .byte % 00000000
. d2f1  00      .byte % 00000000
. d2f2  03      .byte % 00000011
. d2f3  3e      .byte % 00111110
. d2f4  76      .byte % 01110110
. d2f5  36      .byte % 00110110
. d2f6  36      .byte % 00110110
. d2f7  00      .byte % 00000000
. d2f8                          Zeichen: 95
. d2f8  ff      .byte % 11111111
. d2f9  7f      .byte % 01111111
. d2fa  3f      .byte % 00111111
. d2fb  1f      .byte % 00011111
. d2fc  0f      .byte % 00001111
. d2fd  07      .byte % 00000111
. d2fe  03      .byte % 00000011
. d2ff  01      .byte % 00000001
. d300                          Zeichen: 96
. d300  00      .byte % 00000000
. d301  00      .byte % 00000000
. d302  00      .byte % 00000000
. d303  00      .byte % 00000000
. d304  00      .byte % 00000000
. d305  00      .byte % 00000000
. d306  00      .byte % 00000000
. d307  00      .byte % 00000000
. d308                          Zeichen: 97
. d308  f0      .byte % 11110000
. d309  f0      .byte % 11110000
. d30a  f0      .byte % 11110000
. d30b  f0      .byte % 11110000
. d30c  f0      .byte % 11110000
. d30d  f0      .byte % 11110000
. d30e  f0      .byte % 11110000
```

```
. d30f  f0        .byte %11110000
. d310                              Zeichen: 98
. d310  00        .byte %00000000
. d311  00        .byte %00000000
. d312  00        .byte %00000000
. d313  00        .byte %00000000
. d314  ff        .byte %11111111
. d315  ff        .byte %11111111
. d316  ff        .byte %11111111
. d317  ff        .byte %11111111
. d318                              Zeichen: 99
. d318  ff        .byte %11111111
. d319  00        .byte %00000000
. d31a  00        .byte %00000000
. d31b  00        .byte %00000000
. d31c  00        .byte %00000000
. d31d  00        .byte %00000000
. d31e  00        .byte %00000000
. d31f  00        .byte %00000000
. d320                              Zeichen: 100
. d320  00        .byte %00000000
. d321  00        .byte %00000000
. d322  00        .byte %00000000
. d323  00        .byte %00000000
. d324  00        .byte %00000000
. d325  00        .byte %00000000
. d326  00        .byte %00000000
. d327  ff        .byte %11111111
. d328                              Zeichen: 101
. d328  c0        .byte %11000000
. d329  c0        .byte %11000000
. d32a  c0        .byte %11000000
. d32b  c0        .byte %11000000
. d32c  c0        .byte %11000000
. d32d  c0        .byte %11000000
. d32e  c0        .byte %11000000
. d32f  c0        .byte %11000000
. d330                              Zeichen: 102
. d330  cc        .byte %11001100
. d331  cc        .byte %11001100
. d332  33        .byte %00110011
. d333  33        .byte %00110011
. d334  cc        .byte %11001100
. d335  cc        .byte %11001100
. d336  33        .byte %00110011
. d337  33        .byte %00110011
. d338                              Zeichen: 103
. d338  03        .byte %00000011
. d339  03        .byte %00000011
. d33a  03        .byte %00000011
. d33b  03        .byte %00000011
. d33c  03        .byte %00000011
```

```
. d33d  03      .byte % 00000011
. d33e  03      .byte % 00000011
. d33f  03      .byte % 00000011
. d340                          Zeichen: 104
. d340  00      .byte % 00000000
. d341  00      .byte % 00000000
. d342  00      .byte % 00000000
. d343  00      .byte % 00000000
. d344  cc      .byte % 11001100
. d345  cc      .byte % 11001100
. d346  33      .byte % 00110011
. d347  33      .byte % 00110011
. d348                          Zeichen: 105
. d348  ff      .byte % 11111111
. d349  fe      .byte % 11111110
. d34a  fc      .byte % 11111100
. d34b  f8      .byte % 11111000
. d34c  f0      .byte % 11110000
. d34d  e0      .byte % 11100000
. d34e  c0      .byte % 11000000
. d34f  80      .byte % 10000000
. d350                          Zeichen: 106
. d350  03      .byte % 00000011
. d351  03      .byte % 00000011
. d352  03      .byte % 00000011
. d353  03      .byte % 00000011
. d354  03      .byte % 00000011
. d355  03      .byte % 00000011
. d356  03      .byte % 00000011
. d357  03      .byte % 00000011
. d358                          Zeichen: 107
. d358  18      .byte % 00011000
. d359  18      .byte % 00011000
. d35a  18      .byte % 00011000
. d35b  1f      .byte % 00011111
. d35c  1f      .byte % 00011111
. d35d  18      .byte % 00011000
. d35e  18      .byte % 00011000
. d35f  18      .byte % 00011000
. d360                          Zeichen: 108
. d360  00      .byte % 00000000
. d361  00      .byte % 00000000
. d362  00      .byte % 00000000
. d363  00      .byte % 00000000
. d364  0f      .byte % 00001111
. d365  0f      .byte % 00001111
. d366  0f      .byte % 00001111
. d367  0f      .byte % 00001111
. d368                          Zeichen: 109
. d368  18      .byte % 00011000
. d369  18      .byte % 00011000
. d36a  18      .byte % 00011000
```

```
. d36b  1f        .byte % 00011111
. d36c  1f        .byte % 00011111
. d36d  00        .byte % 00000000
. d36e  00        .byte % 00000000
. d36f  00        .byte % 00000000
. d370                              Zeichen: 110
. d370  00        .byte % 00000000
. d371  00        .byte % 00000000
. d372  00        .byte % 00000000
. d373  f8        .byte % 11111000
. d374  f8        .byte % 11111000
. d375  18        .byte % 00011000
. d376  18        .byte % 00011000
. d377  18        .byte % 00011000
. d378                              Zeichen: 111
. d378  00        .byte % 00000000
. d379  00        .byte % 00000000
. d37a  00        .byte % 00000000
. d37b  00        .byte % 00000000
. d37c  00        .byte % 00000000
. d37d  00        .byte % 00000000
. d37e  ff        .byte % 11111111
. d37f  ff        .byte % 11111111
. d380                              Zeichen: 112
. d380  00        .byte % 00000000
. d381  00        .byte % 00000000
. d382  00        .byte % 00000000
. d383  1f        .byte % 00011111
. d384  1f        .byte % 00011111
. d385  18        .byte % 00011000
. d386  18        .byte % 00011000
. d387  18        .byte % 00011000
. d388                              Zeichen: 113
. d388  18        .byte % 00011000
. d389  18        .byte % 00011000
. d38a  18        .byte % 00011000
. d38b  ff        .byte % 11111111
. d38c  ff        .byte % 11111111
. d38d  00        .byte % 00000000
. d38e  00        .byte % 00000000
. d38f  00        .byte % 00000000
. d390                              Zeichen: 114
. d390  00        .byte % 00000000
. d391  00        .byte % 00000000
. d392  00        .byte % 00000000
. d393  ff        .byte % 11111111
. d394  ff        .byte % 11111111
. d395  18        .byte % 00011000
. d396  18        .byte % 00011000
. d397  18        .byte % 00011000
. d398                              Zeichen: 115
. d398  18        .byte % 00011000
```

```
. d399  18       .byte % 00011000
. d39a  18       .byte % 00011000
. d39b  f8       .byte % 11111000
. d39c  f8       .byte % 11111000
. d39d  18       .byte % 00011000
. d39e  18       .byte % 00011000
. d39f  18       .byte % 00011000
. d3a0                             Zeichen: 116
. d3a0  c0       .byte % 11000000
. d3a1  c0       .byte % 11000000
. d3a2  c0       .byte % 11000000
. d3a3  c0       .byte % 11000000
. d3a4  c0       .byte % 11000000
. d3a5  c0       .byte % 11000000
. d3a6  c0       .byte % 11000000
. d3a7  c0       .byte % 11000000
. d3a8                             Zeichen: 117
. d3a8  e0       .byte % 11100000
. d3a9  e0       .byte % 11100000
. d3aa  e0       .byte % 11100000
. d3ab  e0       .byte % 11100000
. d3ac  e0       .byte % 11100000
. d3ad  e0       .byte % 11100000
. d3ae  e0       .byte % 11100000
. d3af  e0       .byte % 11100000
. d3b0                             Zeichen: 118
. d3b0  07       .byte % 00000111
. d3b1  07       .byte % 00000111
. d3b2  07       .byte % 00000111
. d3b3  07       .byte % 00000111
. d3b4  07       .byte % 00000111
. d3b5  07       .byte % 00000111
. d3b6  07       .byte % 00000111
. d3b7  07       .byte % 00000111
. d3b8                             Zeichen: 119
. d3b8  ff       .byte % 11111111
. d3b9  ff       .byte % 11111111
. d3ba  00       .byte % 00000000
. d3bb  00       .byte % 00000000
. d3bc  00       .byte % 00000000
. d3bd  00       .byte % 00000000
. d3be  00       .byte % 00000000
. d3bf  00       .byte % 00000000
. d3c0                             Zeichen: 120
. d3c0  ff       .byte % 11111111
. d3c1  ff       .byte % 11111111
. d3c2  ff       .byte % 11111111
. d3c3  00       .byte % 00000000
. d3c4  00       .byte % 00000000
. d3c5  00       .byte % 00000000
. d3c6  00       .byte % 00000000
. d3c7  00       .byte % 00000000
```

```
. d3c8                                  Zeichen: 121
. d3c8  00      .byte % 00000000
. d3c9  00      .byte % 00000000
. d3ca  00      .byte % 00000000
. d3cb  00      .byte % 00000000
. d3cc  00      .byte % 00000000
. d3cd  ff      .byte % 11111111
. d3ce  ff      .byte % 11111111
. d3cf  ff      .byte % 11111111
. d3d0                                  Zeichen: 122
. d3d0  03      .byte % 00000011
. d3d1  03      .byte % 00000011
. d3d2  03      .byte % 00000011
. d3d3  03      .byte % 00000011
. d3d4  03      .byte % 00000011
. d3d5  03      .byte % 00000011
. d3d6  ff      .byte % 11111111
. d3d7  ff      .byte % 11111111
. d3d8                                  Zeichen: 123
. d3d8  00      .byte % 00000000
. d3d9  00      .byte % 00000000
. d3da  00      .byte % 00000000
. d3db  00      .byte % 00000000
. d3dc  f0      .byte % 11110000
. d3dd  f0      .byte % 11110000
. d3de  f0      .byte % 11110000
. d3df  f0      .byte % 11110000
. d3e0                                  Zeichen: 124
. d3e0  0f      .byte % 00001111
. d3e1  0f      .byte % 00001111
. d3e2  0f      .byte % 00001111
. d3e3  0f      .byte % 00001111
. d3e4  00      .byte % 00000000
. d3e5  00      .byte % 00000000
. d3e6  00      .byte % 00000000
. d3e7  00      .byte % 00000000
. d3e8                                  Zeichen: 125
. d3e8  18      .byte % 00011000
. d3e9  18      .byte % 00011000
. d3ea  18      .byte % 00011000
. d3eb  f8      .byte % 11111000
. d3ec  f8      .byte % 11111000
. d3ed  00      .byte % 00000000
. d3ee  00      .byte % 00000000
. d3ef  00      .byte % 00000000
. d3f0                                  Zeichen: 126
. d3f0  f0      .byte % 11110000
. d3f1  f0      .byte % 11110000
. d3f2  f0      .byte % 11110000
. d3f3  f0      .byte % 11110000
. d3f4  00      .byte % 00000000
. d3f5  00      .byte % 00000000
```

```
. d3f6  00      .byte % 00000000
. d3f7  00      .byte % 00000000
. d3f8                          Zeichen: 127
. d3f8  f0      .byte % 11110000
. d3f9  f0      .byte % 11110000
. d3fa  f0      .byte % 11110000
. d3fb  f0      .byte % 11110000
. d3fc  0f      .byte % 00001111
. d3fd  0f      .byte % 00001111
. d3fe  0f      .byte % 00001111
. d3ff  0f      .byte % 00001111
. d400                          Zeichen: 128 Klammeraffe
. d400  3c      .byte % 00111100
. d401  66      .byte % 01100110
. d402  6e      .byte % 01101110
. d403  6e      .byte % 01101110
. d404  60      .byte % 01100000
. d405  62      .byte % 01100010
. d406  3c      .byte % 00111100
. d407  00      .byte % 00000000
. d408                          Zeichen: 129 'a'
. d408  00      .byte % 00000000
. d409  00      .byte % 00000000
. d40a  3c      .byte % 00111100
. d40b  06      .byte % 00000110
. d40c  3e      .byte % 00111110
. d40d  66      .byte % 01100110
. d40e  3e      .byte % 00111110
. d40f  00      .byte % 00000000
. d410                          Zeichen: 130 'b'
. d410  60      .byte % 01100000
. d411  60      .byte % 01100000
. d412  7c      .byte % 01111100
. d413  66      .byte % 01100110
. d414  66      .byte % 01100110
. d415  66      .byte % 01100110
. d416  7c      .byte % 01111100
. d417  00      .byte % 00000000
. d418                          Zeichen: 131 'c'
. d418  00      .byte % 00000000
. d419  00      .byte % 00000000
. d41a  3c      .byte % 00111100
. d41b  66      .byte % 01100110
. d41c  60      .byte % 01100000
. d41d  66      .byte % 01100110
. d41e  3c      .byte % 00111100
. d41f  00      .byte % 00000000
. d420                          Zeichen: 132 'd'
. d420  06      .byte % 00000110
. d421  06      .byte % 00000110
. d422  3e      .byte % 00111110
. d423  66      .byte % 01100110
```

```
. d424  66       .byte % 01100110
. d425  66       .byte % 01100110
. d426  3e       .byte % 00111110
. d427  00       .byte % 00000000
. d428                           Zeichen: 133 'e'
. d428  00       .byte % 00000000
. d429  00       .byte % 00000000
. d42a  3c       .byte % 00111100
. d42b  66       .byte % 01100110
. d42c  7e       .byte % 01111110
. d42d  60       .byte % 01100000
. d42e  3e       .byte % 00111110
. d42f  00       .byte % 00000000
. d430                           Zeichen: 134 'f'
. d430  1c       .byte % 00011100
. d431  36       .byte % 00110110
. d432  30       .byte % 00110000
. d433  78       .byte % 01111000
. d434  30       .byte % 00110000
. d435  30       .byte % 00110000
. d436  30       .byte % 00110000
. d437  00       .byte % 00000000
. d438                           Zeichen: 135 'g'
. d438  00       .byte % 00000000
. d439  00       .byte % 00000000
. d43a  3e       .byte % 00111110
. d43b  66       .byte % 01100110
. d43c  66       .byte % 01100110
. d43d  3e       .byte % 00111110
. d43e  06       .byte % 00000110
. d43f  7c       .byte % 01111100
. d440                           Zeichen: 136 'h'
. d440  60       .byte % 01100000
. d441  60       .byte % 01100000
. d442  7c       .byte % 01111100
. d443  66       .byte % 01100110
. d444  66       .byte % 01100110
. d445  66       .byte % 01100110
. d446  66       .byte % 01100110
. d447  00       .byte % 00000000
. d448                           Zeichen: 137 'i'
. d448  18       .byte % 00011000
. d449  00       .byte % 00000000
. d44a  18       .byte % 00011000
. d44b  18       .byte % 00011000
. d44c  18       .byte % 00011000
. d44d  18       .byte % 00011000
. d44e  18       .byte % 00011000
. d44f  00       .byte % 00000000
. d450                           Zeichen: 138 'j'
. d450  06       .byte % 00000110
. d451  00       .byte % 00000000
```

```
. d452  06      .byte % 00000110
. d453  06      .byte % 00000110
. d454  06      .byte % 00000110
. d455  06      .byte % 00000110
. d456  66      .byte % 01100110
. d457  3c      .byte % 00111100
. d458                          Zeichen: 139 'k'
. d458  60      .byte % 01100000
. d459  60      .byte % 01100000
. d45a  66      .byte % 01100110
. d45b  6c      .byte % 01101100
. d45c  78      .byte % 01111000
. d45d  7c      .byte % 01111100
. d45e  66      .byte % 01100110
. d45f  00      .byte % 00000000
. d460                          Zeichen: 140 'l'
. d460  38      .byte % 00111000
. d461  18      .byte % 00011000
. d462  18      .byte % 00011000
. d463  18      .byte % 00011000
. d464  18      .byte % 00011000
. d465  18      .byte % 00011000
. d466  3c      .byte % 00111100
. d467  00      .byte % 00000000
. d468                          Zeichen: 141 'm'
. d468  00      .byte % 00000000
. d469  00      .byte % 00000000
. d46a  6b      .byte % 01101011
. d46b  7f      .byte % 01111111
. d46c  7f      .byte % 01111111
. d46d  63      .byte % 01100011
. d46e  63      .byte % 01100011
. d46f  00      .byte % 00000000
. d470                          Zeichen: 142 'n'
. d470  00      .byte % 00000000
. d471  00      .byte % 00000000
. d472  7c      .byte % 01111100
. d473  66      .byte % 01100110
. d474  66      .byte % 01100110
. d475  66      .byte % 01100110
. d476  66      .byte % 01100110
. d477  00      .byte % 00000000
. d478                          Zeichen: 143 'o'
. d478  00      .byte % 00000000
. d479  00      .byte % 00000000
. d47a  3c      .byte % 00111100
. d47b  66      .byte % 01100110
. d47c  66      .byte % 01100110
. d47d  66      .byte % 01100110
. d47e  3c      .byte % 00111100
. d47f  00      .byte % 00000000
. d480                          Zeichen: 144 'p'
```

```
. d480  00        .byte % 00000000
. d481  00        .byte % 00000000
. d482  7c        .byte % 01111100
. d483  66        .byte % 01100110
. d484  66        .byte % 01100110
. d485  7c        .byte % 01111100
. d486  60        .byte % 01100000
. d487  60        .byte % 01100000
. d488                              Zeichen: 145 'q'
. d488  00        .byte % 00000000
. d489  00        .byte % 00000000
. d48a  3e        .byte % 00111110
. d48b  66        .byte % 01100110
. d48c  66        .byte % 01100110
. d48d  3e        .byte % 00111110
. d48e  06        .byte % 00000110
. d48f  06        .byte % 00000110
. d490                              Zeichen: 146 'r'
. d490  00        .byte % 00000000
. d491  00        .byte % 00000000
. d492  7c        .byte % 01111100
. d493  66        .byte % 01100110
. d494  60        .byte % 01100000
. d495  60        .byte % 01100000
. d496  60        .byte % 01100000
. d497  00        .byte % 00000000
. d498                              Zeichen: 147 's'
. d498  00        .byte % 00000000
. d499  00        .byte % 00000000
. d49a  3c        .byte % 00111100
. d49b  60        .byte % 01100000
. d49c  3c        .byte % 00111100
. d49d  06        .byte % 00000110
. d49e  7c        .byte % 01111100
. d49f  00        .byte % 00000000
. d4a0                              Zeichen: 148 't'
. d4a0  30        .byte % 00110000
. d4a1  30        .byte % 00110000
. d4a2  fc        .byte % 11111100
. d4a3  30        .byte % 00110000
. d4a4  30        .byte % 00110000
. d4a5  36        .byte % 00110110
. d4a6  1c        .byte % 00011100
. d4a7  00        .byte % 00000000
. d4a8                              Zeichen: 149 'u'
. d4a8  00        .byte % 00000000
. d4a9  00        .byte % 00000000
. d4aa  66        .byte % 01100110
. d4ab  66        .byte % 01100110
. d4ac  66        .byte % 01100110
. d4ad  66        .byte % 01100110
. d4ae  3c        .byte % 00111100
```

```
. d4af  00      .byte % 00000000
. d4b0                          Zeichen: 150 'v'
. d4b0  00      .byte % 00000000
. d4b1  00      .byte % 00000000
. d4b2  66      .byte % 01100110
. d4b3  66      .byte % 01100110
. d4b4  66      .byte % 01100110
. d4b5  3c      .byte % 00111100
. d4b6  18      .byte % 00011000
. d4b7  00      .byte % 00000000
. d4b8                          Zeichen: 151 'w'
. d4b8  00      .byte % 00000000
. d4b9  00      .byte % 00000000
. d4ba  63      .byte % 01100011
. d4bb  6b      .byte % 01101011
. d4bc  7f      .byte % 01111111
. d4bd  36      .byte % 00110110
. d4be  22      .byte % 00100010
. d4bf  00      .byte % 00000000
. d4c0                          Zeichen: 152 'x'
. d4c0  00      .byte % 00000000
. d4c1  00      .byte % 00000000
. d4c2  66      .byte % 01100110
. d4c3  3c      .byte % 00111100
. d4c4  18      .byte % 00011000
. d4c5  3c      .byte % 00111100
. d4c6  66      .byte % 01100110
. d4c7  00      .byte % 00000000
. d4c8                          Zeichen: 153 'y'
. d4c8  00      .byte % 00000000
. d4c9  00      .byte % 00000000
. d4ca  66      .byte % 01100110
. d4cb  66      .byte % 01100110
. d4cc  66      .byte % 01100110
. d4cd  3e      .byte % 00111110
. d4ce  06      .byte % 00000110
. d4cf  7c      .byte % 01111100
. d4d0                          Zeichen: 154 'z'
. d4d0  00      .byte % 00000000
. d4d1  00      .byte % 00000000
. d4d2  7e      .byte % 01111110
. d4d3  0c      .byte % 00001100
. d4d4  18      .byte % 00011000
. d4d5  30      .byte % 00110000
. d4d6  7e      .byte % 01111110
. d4d7  00      .byte % 00000000
. d4d8                          Zeichen: 155 Eckige Klammer
. d4d8  3c      .byte % 00111100
. d4d9  30      .byte % 00110000
. d4da  30      .byte % 00110000
. d4db  30      .byte % 00110000
. d4dc  30      .byte % 00110000
```

```
. d4dd  30      .byte % 00110000
. d4de  3c      .byte % 00111100
. d4df  00      .byte % 00000000
. d4e0                          Zeichen: 156 Pfund
. d4e0  0c      .byte % 00001100
. d4e1  12      .byte % 00010010
. d4e2  30      .byte % 00110000
. d4e3  7c      .byte % 01111100
. d4e4  30      .byte % 00110000
. d4e5  62      .byte % 01100010
. d4e6  fc      .byte % 11111100
. d4e7  00      .byte % 00000000
. d4e8                          Zeichen: 157 Eckige Klammer
. d4e8  3c      .byte % 00111100
. d4e9  0c      .byte % 00001100
. d4ea  0c      .byte % 00001100
. d4eb  0c      .byte % 00001100
. d4ec  0c      .byte % 00001100
. d4ed  0c      .byte % 00001100
. d4ee  3c      .byte % 00111100
. d4ef  00      .byte % 00000000
. d4f0                          Zeichen: 158 Pfeil
. d4f0  00      .byte % 00000000
. d4f1  18      .byte % 00011000
. d4f2  3c      .byte % 00111100
. d4f3  7e      .byte % 01111110
. d4f4  18      .byte % 00011000
. d4f5  18      .byte % 00011000
. d4f6  18      .byte % 00011000
. d4f7  18      .byte % 00011000
. d4f8                          Zeichen: 159
. d4f8  00      .byte % 00000000
. d4f9  10      .byte % 00010000
. d4fa  30      .byte % 00110000
. d4fb  7f      .byte % 01111111
. d4fc  7f      .byte % 01111111
. d4fd  30      .byte % 00110000
. d4fe  10      .byte % 00010000
. d4ff  00      .byte % 00000000
. d500                          Zeichen: 160 SPACE
. d500  00      .byte % 00000000
. d501  00      .byte % 00000000
. d502  00      .byte % 00000000
. d503  00      .byte % 00000000
. d504  00      .byte % 00000000
. d505  00      .byte % 00000000
. d506  00      .byte % 00000000
. d507  00      .byte % 00000000
. d508                          Zeichen: 161 '!'
. d508  18      .byte % 00011000
. d509  18      .byte % 00011000
. d50a  18      .byte % 00011000
```

```
. d50b  18      .byte % 00011000
. d50c  00      .byte % 00000000
. d50d  00      .byte % 00000000
. d50e  18      .byte % 00011000
. d50f  00      .byte % 00000000
. d510                          Zeichen: 162 '"'
. d510  66      .byte % 01100110
. d511  66      .byte % 01100110
. d512  66      .byte % 01100110
. d513  00      .byte % 00000000
. d514  00      .byte % 00000000
. d515  00      .byte % 00000000
. d516  00      .byte % 00000000
. d517  00      .byte % 00000000
. d518                          Zeichen: 163 '#'
. d518  66      .byte % 01100110
. d519  66      .byte % 01100110
. d51a  ff      .byte % 11111111
. d51b  66      .byte % 01100110
. d51c  ff      .byte % 11111111
. d51d  66      .byte % 01100110
. d51e  66      .byte % 01100110
. d51f  00      .byte % 00000000
. d520                          Zeichen: 164 '$'
. d520  18      .byte % 00011000
. d521  3e      .byte % 00111110
. d522  60      .byte % 01100000
. d523  3c      .byte % 00111100
. d524  06      .byte % 00000110
. d525  7c      .byte % 01111100
. d526  18      .byte % 00011000
. d527  00      .byte % 00000000
. d528                          Zeichen: 165 '%'
. d528  62      .byte % 01100010
. d529  66      .byte % 01100110
. d52a  0c      .byte % 00001100
. d52b  18      .byte % 00011000
. d52c  30      .byte % 00110000
. d52d  66      .byte % 01100110
. d52e  46      .byte % 01000110
. d52f  00      .byte % 00000000
. d530                          Zeichen: 166 '&'
. d530  3c      .byte % 00111100
. d531  66      .byte % 01100110
. d532  3c      .byte % 00111100
. d533  38      .byte % 00111000
. d534  67      .byte % 01100111
. d535  66      .byte % 01100110
. d536  3f      .byte % 00111111
. d537  00      .byte % 00000000
. d538                          Zeichen: 167 '''
. d538  06      .byte % 00000110
```

```
. d539  0c      .byte % 00001100
. d53a  18      .byte % 00011000
. d53b  00      .byte % 00000000
. d53c  00      .byte % 00000000
. d53d  00      .byte % 00000000
. d53e  00      .byte % 00000000
. d53f  00      .byte % 00000000
. d540                          Zeichen: 168 '('
. d540  0c      .byte % 00001100
. d541  18      .byte % 00011000
. d542  30      .byte % 00110000
. d543  30      .byte % 00110000
. d544  30      .byte % 00110000
. d545  18      .byte % 00011000
. d546  0c      .byte % 00001100
. d547  00      .byte % 00000000
. d548                          Zeichen: 169 ')'
. d548  30      .byte % 00110000
. d549  18      .byte % 00011000
. d54a  0c      .byte % 00001100
. d54b  0c      .byte % 00001100
. d54c  0c      .byte % 00001100
. d54d  18      .byte % 00011000
. d54e  30      .byte % 00110000
. d54f  00      .byte % 00000000
. d550                          Zeichen: 170 '*'
. d550  00      .byte % 00000000
. d551  66      .byte % 01100110
. d552  3c      .byte % 00111100
. d553  ff      .byte % 11111111
. d554  3c      .byte % 00111100
. d555  66      .byte % 01100110
. d556  00      .byte % 00000000
. d557  00      .byte % 00000000
. d558                          Zeichen: 171 '+'
. d558  00      .byte % 00000000
. d559  18      .byte % 00011000
. d55a  18      .byte % 00011000
. d55b  7e      .byte % 01111110
. d55c  18      .byte % 00011000
. d55d  18      .byte % 00011000
. d55e  00      .byte % 00000000
. d55f  00      .byte % 00000000
. d560                          Zeichen: 172 ','
. d560  00      .byte % 00000000
. d561  00      .byte % 00000000
. d562  00      .byte % 00000000
. d563  00      .byte % 00000000
. d564  00      .byte % 00000000
. d565  18      .byte % 00011000
. d566  18      .byte % 00011000
. d567  30      .byte % 00110000
```

```
. d568                                  Zeichen: 173 '-'
. d568  00      .byte % 00000000
. d569  00      .byte % 00000000
. d56a  00      .byte % 00000000
. d56b  7e      .byte % 01111110
. d56c  00      .byte % 00000000
. d56d  00      .byte % 00000000
. d56e  00      .byte % 00000000
. d56f  00      .byte % 00000000
. d570                                  Zeichen: 174 '.'
. d570  00      .byte % 00000000
. d571  00      .byte % 00000000
. d572  00      .byte % 00000000
. d573  00      .byte % 00000000
. d574  00      .byte % 00000000
. d575  18      .byte % 00011000
. d576  18      .byte % 00011000
. d577  00      .byte % 00000000
. d578                                  Zeichen: 175 '/'
. d578  00      .byte % 00000000
. d579  03      .byte % 00000011
. d57a  06      .byte % 00000110
. d57b  0c      .byte % 00001100
. d57c  18      .byte % 00011000
. d57d  30      .byte % 00110000
. d57e  60      .byte % 01100000
. d57f  00      .byte % 00000000
. d580                                  Zeichen: 176 '0'
. d580  3c      .byte % 00111100
. d581  66      .byte % 01100110
. d582  6e      .byte % 01101110
. d583  76      .byte % 01110110
. d584  66      .byte % 01100110
. d585  66      .byte % 01100110
. d586  3c      .byte % 00111100
. d587  00      .byte % 00000000
. d588                                  Zeichen: 177 '1'
. d588  18      .byte % 00011000
. d589  18      .byte % 00011000
. d58a  38      .byte % 00111000
. d58b  18      .byte % 00011000
. d58c  18      .byte % 00011000
. d58d  18      .byte % 00011000
. d58e  7e      .byte % 01111110
. d58f  00      .byte % 00000000
. d590                                  Zeichen: 178 '2'
. d590  3c      .byte % 00111100
. d591  66      .byte % 01100110
. d592  06      .byte % 00000110
. d593  0c      .byte % 00001100
. d594  30      .byte % 00110000
. d595  60      .byte % 01100000
```

```
. d596  7e       .byte % 01111110
. d597  00       .byte % 00000000
. d598                           Zeichen: 179 '3'
. d598  3c       .byte % 00111100
. d599  66       .byte % 01100110
. d59a  06       .byte % 00000110
. d59b  1c       .byte % 00011100
. d59c  06       .byte % 00000110
. d59d  66       .byte % 01100110
. d59e  3c       .byte % 00111100
. d59f  00       .byte % 00000000
. d5a0                           Zeichen: 180 '4'
. d5a0  06       .byte % 00000110
. d5a1  0e       .byte % 00001110
. d5a2  1e       .byte % 00011110
. d5a3  66       .byte % 01100110
. d5a4  7f       .byte % 01111111
. d5a5  06       .byte % 00000110
. d5a6  06       .byte % 00000110
. d5a7  00       .byte % 00000000
. d5a8                           Zeichen: 181 '5'
. d5a8  7e       .byte % 01111110
. d5a9  60       .byte % 01100000
. d5aa  7c       .byte % 01111100
. d5ab  06       .byte % 00000110
. d5ac  06       .byte % 00000110
. d5ad  66       .byte % 01100110
. d5ae  3c       .byte % 00111100
. d5af  00       .byte % 00000000
. d5b0                           Zeichen: 182 '6'
. d5b0  3c       .byte % 00111100
. d5b1  66       .byte % 01100110
. d5b2  60       .byte % 01100000
. d5b3  7c       .byte % 01111100
. d5b4  66       .byte % 01100110
. d5b5  66       .byte % 01100110
. d5b6  3c       .byte % 00111100
. d5b7  00       .byte % 00000000
. d5b8                           Zeichen: 183 '7'
. d5b8  7e       .byte % 01111110
. d5b9  66       .byte % 01100110
. d5ba  0c       .byte % 00001100
. d5bb  18       .byte % 00011000
. d5bc  18       .byte % 00011000
. d5bd  18       .byte % 00011000
. d5be  18       .byte % 00011000
. d5bf  00       .byte % 00000000
. d5c0                           Zeichen: 184 '8'
. d5c0  3c       .byte % 00111100
. d5c1  66       .byte % 01100110
. d5c2  66       .byte % 01100110
. d5c3  3c       .byte % 00111100
```

```
. d5c4  66      .byte % 01100110
. d5c5  66      .byte % 01100110
. d5c6  3c      .byte % 00111100
. d5c7  00      .byte % 00000000
. d5c8                          Zeichen: 185 '9'
. d5c8  3c      .byte % 00111100
. d5c9  66      .byte % 01100110
. d5ca  66      .byte % 01100110
. d5cb  3e      .byte % 00111110
. d5cc  06      .byte % 00000110
. d5cd  66      .byte % 01100110
. d5ce  3c      .byte % 00111100
. d5cf  00      .byte % 00000000
. d5d0                          Zeichen: 186 ':'
. d5d0  00      .byte % 00000000
. d5d1  00      .byte % 00000000
. d5d2  18      .byte % 00011000
. d5d3  00      .byte % 00000000
. d5d4  00      .byte % 00000000
. d5d5  18      .byte % 00011000
. d5d6  00      .byte % 00000000
. d5d7  00      .byte % 00000000
. d5d8                          Zeichen: 187 ';'
. d5d8  00      .byte % 00000000
. d5d9  00      .byte % 00000000
. d5da  18      .byte % 00011000
. d5db  00      .byte % 00000000
. d5dc  00      .byte % 00000000
. d5dd  18      .byte % 00011000
. d5de  18      .byte % 00011000
. d5df  30      .byte % 00110000
. d5e0                          Zeichen: 188 <
. d5e0  0e      .byte % 00001110
. d5e1  18      .byte % 00011000
. d5e2  30      .byte % 00110000
. d5e3  60      .byte % 01100000
. d5e4  30      .byte % 00110000
. d5e5  18      .byte % 00011000
. d5e6  0e      .byte % 00001110
. d5e7  00      .byte % 00000000
. d5e8                          Zeichen: 189 =
. d5e8  00      .byte % 00000000
. d5e9  00      .byte % 00000000
. d5ea  7e      .byte % 01111110
. d5eb  00      .byte % 00000000
. d5ec  7e      .byte % 01111110
. d5ed  00      .byte % 00000000
. d5ee  00      .byte % 00000000
. d5ef  00      .byte % 00000000
. d5f0                          Zeichen: 190 >
. d5f0  70      .byte % 01110000
. d5f1  18      .byte % 00011000
```

```
. d5f2  0c      .byte % 00001100
. d5f3  06      .byte % 00000110
. d5f4  0c      .byte % 00001100
. d5f5  18      .byte % 00011000
. d5f6  70      .byte % 01110000
. d5f7  00      .byte % 00000000
. d5f8                          Zeichen: 191 '?'
. d5f8  3c      .byte % 00111100
. d5f9  66      .byte % 01100110
. d5fa  06      .byte % 00000110
. d5fb  0c      .byte % 00001100
. d5fc  18      .byte % 00011000
. d5fd  00      .byte % 00000000
. d5fe  18      .byte % 00011000
. d5ff  00      .byte % 00000000
. d600                          Zeichen: 192
. d600  00      .byte % 00000000
. d601  00      .byte % 00000000
. d602  00      .byte % 00000000
. d603  ff      .byte % 11111111
. d604  ff      .byte % 11111111
. d605  00      .byte % 00000000
. d606  00      .byte % 00000000
. d607  00      .byte % 00000000
. d608                          Zeichen: 193 'A'
. d608  18      .byte % 00011000
. d609  3c      .byte % 00111100
. d60a  66      .byte % 01100110
. d60b  7e      .byte % 01111110
. d60c  66      .byte % 01100110
. d60d  66      .byte % 01100110
. d60e  66      .byte % 01100110
. d60f  00      .byte % 00000000
. d610                          Zeichen: 194 'B'
. d610  7c      .byte % 01111100
. d611  66      .byte % 01100110
. d612  66      .byte % 01100110
. d613  7c      .byte % 01111100
. d614  66      .byte % 01100110
. d615  66      .byte % 01100110
. d616  7c      .byte % 01111100
. d617  00      .byte % 00000000
. d618                          Zeichen: 195 'C'
. d618  3c      .byte % 00111100
. d619  66      .byte % 01100110
. d61a  60      .byte % 01100000
. d61b  60      .byte % 01100000
. d61c  60      .byte % 01100000
. d61d  66      .byte % 01100110
. d61e  3c      .byte % 00111100
. d61f  00      .byte % 00000000
. d620                          Zeichen: 196 'D'
```

```
. d620  78        .byte % 01111000
. d621  6c        .byte % 01101100
. d622  66        .byte % 01100110
. d623  66        .byte % 01100110
. d624  66        .byte % 01100110
. d625  6c        .byte % 01101100
. d626  78        .byte % 01111000
. d627  00        .byte % 00000000
. d628                              Zeichen: 197 'E'
. d628  7e        .byte % 01111110
. d629  60        .byte % 01100000
. d62a  60        .byte % 01100000
. d62b  78        .byte % 01111000
. d62c  60        .byte % 01100000
. d62d  60        .byte % 01100000
. d62e  7e        .byte % 01111110
. d62f  00        .byte % 00000000
. d630                              Zeichen: 198 'F'
. d630  7e        .byte % 01111110
. d631  60        .byte % 01100000
. d632  60        .byte % 01100000
. d633  78        .byte % 01111000
. d634  60        .byte % 01100000
. d635  60        .byte % 01100000
. d636  60        .byte % 01100000
. d637  00        .byte % 00000000
. d638                              Zeichen: 199 'G'
. d638  3c        .byte % 00111100
. d639  66        .byte % 01100110
. d63a  60        .byte % 01100000
. d63b  6e        .byte % 01101110
. d63c  66        .byte % 01100110
. d63d  66        .byte % 01100110
. d63e  3c        .byte % 00111100
. d63f  00        .byte % 00000000
. d640                              Zeichen: 200 'H'
. d640  66        .byte % 01100110
. d641  66        .byte % 01100110
. d642  66        .byte % 01100110
. d643  7e        .byte % 01111110
. d644  66        .byte % 01100110
. d645  66        .byte % 01100110
. d646  66        .byte % 01100110
. d647  00        .byte % 00000000
. d648                              Zeichen: 201 'I'
. d648  3c        .byte % 00111100
. d649  18        .byte % 00011000
. d64a  18        .byte % 00011000
. d64b  18        .byte % 00011000
. d64c  18        .byte % 00011000
. d64d  18        .byte % 00011000
. d64e  3c        .byte % 00111100
```

```
. d64f  00      .byte % 00000000
. d650                          Zeichen: 202 'J'
. d650  1e      .byte % 00011110
. d651  0c      .byte % 00001100
. d652  0c      .byte % 00001100
. d653  0c      .byte % 00001100
. d654  0c      .byte % 00001100
. d655  6c      .byte % 01101100
. d656  38      .byte % 00111000
. d657  00      .byte % 00000000
. d658                          Zeichen: 203 'K'
. d658  66      .byte % 01100110
. d659  6c      .byte % 01101100
. d65a  78      .byte % 01111000
. d65b  70      .byte % 01110000
. d65c  78      .byte % 01111000
. d65d  6c      .byte % 01101100
. d65e  66      .byte % 01100110
. d65f  00      .byte % 00000000
. d660                          Zeichen: 204 'L'
. d660  60      .byte % 01100000
. d661  60      .byte % 01100000
. d662  60      .byte % 01100000
. d663  60      .byte % 01100000
. d664  60      .byte % 01100000
. d665  60      .byte % 01100000
. d666  7e      .byte % 01111110
. d667  00      .byte % 00000000
. d668                          Zeichen: 205 'M'
. d668  63      .byte % 01100011
. d669  77      .byte % 01110111
. d66a  7f      .byte % 01111111
. d66b  6b      .byte % 01101011
. d66c  63      .byte % 01100011
. d66d  63      .byte % 01100011
. d66e  63      .byte % 01100011
. d66f  00      .byte % 00000000
. d670                          Zeichen: 206 'N'
. d670  66      .byte % 01100110
. d671  76      .byte % 01110110
. d672  7e      .byte % 01111110
. d673  7e      .byte % 01111110
. d674  6e      .byte % 01101110
. d675  66      .byte % 01100110
. d676  66      .byte % 01100110
. d677  00      .byte % 00000000
. d678                          Zeichen: 207 'O'
. d678  3c      .byte % 00111100
. d679  66      .byte % 01100110
. d67a  66      .byte % 01100110
. d67b  66      .byte % 01100110
. d67c  66      .byte % 01100110
```

```
. d67d  66       .byte % 01100110
. d67e  3c       .byte % 00111100
. d67f  00       .byte % 00000000
. d680                           Zeichen: 208 'P'
. d680  7c       .byte % 01111100
. d681  66       .byte % 01100110
. d682  66       .byte % 01100110
. d683  7c       .byte % 01111100
. d684  60       .byte % 01100000
. d685  60       .byte % 01100000
. d686  60       .byte % 01100000
. d687  00       .byte % 00000000
. d688                           Zeichen: 209 'Q'
. d688  3c       .byte % 00111100
. d689  66       .byte % 01100110
. d68a  66       .byte % 01100110
. d68b  66       .byte % 01100110
. d68c  66       .byte % 01100110
. d68d  3c       .byte % 00111100
. d68e  0e       .byte % 00001110
. d68f  00       .byte % 00000000
. d690                           Zeichen: 210 'R'
. d690  7c       .byte % 01111100
. d691  66       .byte % 01100110
. d692  66       .byte % 01100110
. d693  7c       .byte % 01111100
. d694  78       .byte % 01111000
. d695  6c       .byte % 01101100
. d696  66       .byte % 01100110
. d697  00       .byte % 00000000
. d698                           Zeichen: 211 'S'
. d698  3c       .byte % 00111100
. d699  66       .byte % 01100110
. d69a  60       .byte % 01100000
. d69b  3c       .byte % 00111100
. d69c  06       .byte % 00000110
. d69d  66       .byte % 01100110
. d69e  3c       .byte % 00111100
. d69f  00       .byte % 00000000
. d6a0                           Zeichen: 212 'T'
. d6a0  7e       .byte % 01111110
. d6a1  18       .byte % 00011000
. d6a2  18       .byte % 00011000
. d6a3  18       .byte % 00011000
. d6a4  18       .byte % 00011000
. d6a5  18       .byte % 00011000
. d6a6  18       .byte % 00011000
. d6a7  00       .byte % 00000000
. d6a8                           Zeichen: 213 'U'
. d6a8  66       .byte % 01100110
. d6a9  66       .byte % 01100110
. d6aa  66       .byte % 01100110
```

```
. d6ab  66       .byte % 01100110
. d6ac  66       .byte % 01100110
. d6ad  66       .byte % 01100110
. d6ae  3c       .byte % 00111100
. d6af  00       .byte % 00000000
. d6b0                             Zeichen: 214 'V'
. d6b0  66       .byte % 01100110
. d6b1  66       .byte % 01100110
. d6b2  66       .byte % 01100110
. d6b3  66       .byte % 01100110
. d6b4  66       .byte % 01100110
. d6b5  3c       .byte % 00111100
. d6b6  18       .byte % 00011000
. d6b7  00       .byte % 00000000
. d6b8                             Zeichen: 215 'W'
. d6b8  63       .byte % 01100011
. d6b9  63       .byte % 01100011
. d6ba  63       .byte % 01100011
. d6bb  6b       .byte % 01101011
. d6bc  7f       .byte % 01111111
. d6bd  77       .byte % 01110111
. d6be  63       .byte % 01100011
. d6bf  00       .byte % 00000000
. d6c0                             Zeichen: 216 'X'
. d6c0  66       .byte % 01100110
. d6c1  66       .byte % 01100110
. d6c2  3c       .byte % 00111100
. d6c3  18       .byte % 00011000
. d6c4  3c       .byte % 00111100
. d6c5  66       .byte % 01100110
. d6c6  66       .byte % 01100110
. d6c7  00       .byte % 00000000
. d6c8                             Zeichen: 217 'Y'
. d6c8  66       .byte % 01100110
. d6c9  66       .byte % 01100110
. d6ca  66       .byte % 01100110
. d6cb  3c       .byte % 00111100
. d6cc  18       .byte % 00011000
. d6cd  18       .byte % 00011000
. d6ce  18       .byte % 00011000
. d6cf  00       .byte % 00000000
. d6d0                             Zeichen: 218 'Z'
. d6d0  7e       .byte % 01111110
. d6d1  06       .byte % 00000110
. d6d2  0c       .byte % 00001100
. d6d3  18       .byte % 00011000
. d6d4  30       .byte % 00110000
. d6d5  60       .byte % 01100000
. d6d6  7e       .byte % 01111110
. d6d7  00       .byte % 00000000
. d6d8                             Zeichen: 219
. d6d8  18       .byte % 00011000
```

```
. d6d9  18      .byte %00011000
. d6da  18      .byte %00011000
. d6db  ff      .byte %11111111
. d6dc  ff      .byte %11111111
. d6dd  18      .byte %00011000
. d6de  18      .byte %00011000
. d6df  18      .byte %00011000
. d6e0                          Zeichen: 220
. d6e0  c0      .byte %11000000
. d6e1  c0      .byte %11000000
. d6e2  30      .byte %00110000
. d6e3  30      .byte %00110000
. d6e4  c0      .byte %11000000
. d6e5  c0      .byte %11000000
. d6e6  30      .byte %00110000
. d6e7  30      .byte %00110000
. d6e8                          Zeichen: 221
. d6e8  18      .byte %00011000
. d6e9  18      .byte %00011000
. d6ea  18      .byte %00011000
. d6eb  18      .byte %00011000
. d6ec  18      .byte %00011000
. d6ed  18      .byte %00011000
. d6ee  18      .byte %00011000
. d6ef  18      .byte %00011000
. d6f0                          Zeichen: 222
. d6f0  33      .byte %00110011
. d6f1  33      .byte %00110011
. d6f2  cc      .byte %11001100
. d6f3  cc      .byte %11001100
. d6f4  33      .byte %00110011
. d6f5  33      .byte %00110011
. d6f6  cc      .byte %11001100
. d6f7  cc      .byte %11001100
. d6f8                          Zeichen: 223
. d6f8  33      .byte %00110011
. d6f9  99      .byte %10011001
. d6fa  cc      .byte %11001100
. d6fb  66      .byte %01100110
. d6fc  33      .byte %00110011
. d6fd  99      .byte %10011001
. d6fe  cc      .byte %11001100
. d6ff  66      .byte %01100110
. d700                          Zeichen: 224 SPACE
. d700  00      .byte %00000000
. d701  00      .byte %00000000
. d702  00      .byte %00000000
. d703  00      .byte %00000000
. d704  00      .byte %00000000
. d705  00      .byte %00000000
. d706  00      .byte %00000000
. d707  00      .byte %00000000
```

```
. d708                                  Zeichen: 225
. d708  f0      .byte % 11110000
. d709  f0      .byte % 11110000
. d70a  f0      .byte % 11110000
. d70b  f0      .byte % 11110000
. d70c  f0      .byte % 11110000
. d70d  f0      .byte % 11110000
. d70e  f0      .byte % 11110000
. d70f  f0      .byte % 11110000
. d710                                  Zeichen: 226
. d710  00      .byte % 00000000
. d711  00      .byte % 00000000
. d712  00      .byte % 00000000
. d713  00      .byte % 00000000
. d714  ff      .byte % 11111111
. d715  ff      .byte % 11111111
. d716  ff      .byte % 11111111
. d717  ff      .byte % 11111111
. d718                                  Zeichen: 227
. d718  ff      .byte % 11111111
. d719  00      .byte % 00000000
. d71a  00      .byte % 00000000
. d71b  00      .byte % 00000000
. d71c  00      .byte % 00000000
. d71d  00      .byte % 00000000
. d71e  00      .byte % 00000000
. d71f  00      .byte % 00000000
. d720                                  Zeichen: 228
. d720  00      .byte % 00000000
. d721  00      .byte % 00000000
. d722  00      .byte % 00000000
. d723  00      .byte % 00000000
. d724  00      .byte % 00000000
. d725  00      .byte % 00000000
. d726  00      .byte % 00000000
. d727  ff      .byte % 11111111
. d728                                  Zeichen: 229
. d728  c0      .byte % 11000000
. d729  c0      .byte % 11000000
. d72a  c0      .byte % 11000000
. d72b  c0      .byte % 11000000
. d72c  c0      .byte % 11000000
. d72d  c0      .byte % 11000000
. d72e  c0      .byte % 11000000
. d72f  c0      .byte % 11000000
. d730                                  Zeichen: 230
. d730  cc      .byte % 11001100
. d731  cc      .byte % 11001100
. d732  33      .byte % 00110011
. d733  33      .byte % 00110011
. d734  cc      .byte % 11001100
. d735  cc      .byte % 11001100
```

```
. d736  33      .byte % 00110011
. d737  33      .byte % 00110011
. d738                          Zeichen: 231
. d738  03      .byte % 00000011
. d739  03      .byte % 00000011
. d73a  03      .byte % 00000011
. d73b  03      .byte % 00000011
. d73c  03      .byte % 00000011
. d73d  03      .byte % 00000011
. d73e  03      .byte % 00000011
. d73f  03      .byte % 00000011
. d740                          Zeichen: 232
. d740  00      .byte % 00000000
. d741  00      .byte % 00000000
. d742  00      .byte % 00000000
. d743  00      .byte % 00000000
. d744  cc      .byte % 11001100
. d745  cc      .byte % 11001100
. d746  33      .byte % 00110011
. d747  33      .byte % 00110011
. d748                          Zeichen: 233
. d748  cc      .byte % 11001100
. d749  99      .byte % 10011001
. d74a  33      .byte % 00110011
. d74b  66      .byte % 01100110
. d74c  cc      .byte % 11001100
. d74d  99      .byte % 10011001
. d74e  33      .byte % 00110011
. d74f  66      .byte % 01100110
. d750                          Zeichen: 234
. d750  03      .byte % 00000011
. d751  03      .byte % 00000011
. d752  03      .byte % 00000011
. d753  03      .byte % 00000011
. d754  03      .byte % 00000011
. d755  03      .byte % 00000011
. d756  03      .byte % 00000011
. d757  03      .byte % 00000011
. d758                          Zeichen: 235
. d758  18      .byte % 00011000
. d759  18      .byte % 00011000
. d75a  18      .byte % 00011000
. d75b  1f      .byte % 00011111
. d75c  1f      .byte % 00011111
. d75d  18      .byte % 00011000
. d75e  18      .byte % 00011000
. d75f  18      .byte % 00011000
. d760                          Zeichen: 236
. d760  00      .byte % 00000000
. d761  00      .byte % 00000000
. d762  00      .byte % 00000000
. d763  00      .byte % 00000000
```

```
. d764  0f      .byte % 00001111
. d765  0f      .byte % 00001111
. d766  0f      .byte % 00001111
. d767  0f      .byte % 00001111
. d768                          Zeichen: 237
. d768  18      .byte % 00011000
. d769  18      .byte % 00011000
. d76a  18      .byte % 00011000
. d76b  1f      .byte % 00011111
. d76c  1f      .byte % 00011111
. d76d  00      .byte % 00000000
. d76e  00      .byte % 00000000
. d76f  00      .byte % 00000000
. d770                          Zeichen: 238
. d770  00      .byte % 00000000
. d771  00      .byte % 00000000
. d772  00      .byte % 00000000
. d773  f8      .byte % 11111000
. d774  f8      .byte % 11111000
. d775  18      .byte % 00011000
. d776  18      .byte % 00011000
. d777  18      .byte % 00011000
. d778                          Zeichen: 239
. d778  00      .byte % 00000000
. d779  00      .byte % 00000000
. d77a  00      .byte % 00000000
. d77b  00      .byte % 00000000
. d77c  00      .byte % 00000000
. d77d  00      .byte % 00000000
. d77e  ff      .byte % 11111111
. d77f  ff      .byte % 11111111
. d780                          Zeichen: 240
. d780  00      .byte % 00000000
. d781  00      .byte % 00000000
. d782  00      .byte % 00000000
. d783  1f      .byte % 00011111
. d784  1f      .byte % 00011111
. d785  18      .byte % 00011000
. d786  18      .byte % 00011000
. d787  18      .byte % 00011000
. d788                          Zeichen: 241
. d788  18      .byte % 00011000
. d789  18      .byte % 00011000
. d78a  18      .byte % 00011000
. d78b  ff      .byte % 11111111
. d78c  ff      .byte % 11111111
. d78d  00      .byte % 00000000
. d78e  00      .byte % 00000000
. d78f  00      .byte % 00000000
. d790                          Zeichen: 242
. d790  00      .byte % 00000000
. d791  00      .byte % 00000000
```

```
. d792  ØØ      .byte % ØØØØØØØØ
. d793  ff      .byte % 11111111
. d794  ff      .byte % 11111111
. d795  18      .byte % ØØØ11ØØØ
. d796  18      .byte % ØØØ11ØØØ
. d797  18      .byte % ØØØ11ØØØ
. d798                          Zeichen: 243
. d798  18      .byte % ØØØ11ØØØ
. d799  18      .byte % ØØØ11ØØØ
. d79a  18      .byte % ØØØ11ØØØ
. d79b  f8      .byte % 11111ØØØ
. d79c  f8      .byte % 11111ØØØ
. d79d  18      .byte % ØØØ11ØØØ
. d79e  18      .byte % ØØØ11ØØØ
. d79f  18      .byte % ØØØ11ØØØ
. d7aØ                          Zeichen: 244
. d7aØ  cØ      .byte % 11ØØØØØØ
. d7a1  cØ      .byte % 11ØØØØØØ
. d7a2  cØ      .byte % 11ØØØØØØ
. d7a3  cØ      .byte % 11ØØØØØØ
. d7a4  cØ      .byte % 11ØØØØØØ
. d7a5  cØ      .byte % 11ØØØØØØ
. d7a6  cØ      .byte % 11ØØØØØØ
. d7a7  cØ      .byte % 11ØØØØØØ
. d7a8                          Zeichen: 245
. d7a8  eØ      .byte % 111ØØØØØ
. d7a9  eØ      .byte % 111ØØØØØ
. d7aa  eØ      .byte % 111ØØØØØ
. d7ab  eØ      .byte % 111ØØØØØ
. d7ac  eØ      .byte % 111ØØØØØ
. d7ad  eØ      .byte % 111ØØØØØ
. d7ae  eØ      .byte % 111ØØØØØ
. d7af  eØ      .byte % 111ØØØØØ
. d7bØ                          Zeichen: 246
. d7bØ  Ø7      .byte % ØØØØØ111
. d7b1  Ø7      .byte % ØØØØØ111
. d7b2  Ø7      .byte % ØØØØØ111
. d7b3  Ø7      .byte % ØØØØØ111
. d7b4  Ø7      .byte % ØØØØØ111
. d7b5  Ø7      .byte % ØØØØØ111
. d7b6  Ø7      .byte % ØØØØØ111
. d7b7  Ø7      .byte % ØØØØØ111
. d7b8                          Zeichen: 247
. d7b8  ff      .byte % 11111111
. d7b9  ff      .byte % 11111111
. d7ba  ØØ      .byte % ØØØØØØØØ
. d7bb  ØØ      .byte % ØØØØØØØØ
. d7bc  ØØ      .byte % ØØØØØØØØ
. d7bd  ØØ      .byte % ØØØØØØØØ
. d7be  ØØ      .byte % ØØØØØØØØ
. d7bf  ØØ      .byte % ØØØØØØØØ
. d7cØ                          Zeichen: 248
```

```
. d7cØ  ff      .byte % 11111111
. d7c1  ff      .byte % 11111111
. d7c2  ff      .byte % 11111111
. d7c3  ØØ      .byte % ØØØØØØØØ
. d7c4  ØØ      .byte % ØØØØØØØØ
. d7c5  ØØ      .byte % ØØØØØØØØ
. d7c6  ØØ      .byte % ØØØØØØØØ
. d7c7  ØØ      .byte % ØØØØØØØØ
. d7c8                          Zeichen: 249
. d7c8  ØØ      .byte % ØØØØØØØØ
. d7c9  ØØ      .byte % ØØØØØØØØ
. d7ca  ØØ      .byte % ØØØØØØØØ
. d7cb  ØØ      .byte % ØØØØØØØØ
. d7cc  ØØ      .byte % ØØØØØØØØ
. d7cd  ff      .byte % 11111111
. d7ce  ff      .byte % 11111111
. d7cf  ff      .byte % 11111111
. d7dØ                          Zeichen: 25Ø
. d7dØ  Ø1      .byte % ØØØØØØØ1
. d7d1  Ø3      .byte % ØØØØØ11
. d7d2  Ø6      .byte % ØØØØ11Ø
. d7d3  6c      .byte % Ø11Ø11ØØ
. d7d4  78      .byte % Ø1111ØØØ
. d7d5  7Ø      .byte % Ø1111ØØØ
. d7d6  6Ø      .byte % Ø11ØØØØØ
. d7d7  ØØ      .byte % ØØØØØØØØ
. d7d8                          Zeichen: 251
. d7d8  ØØ      .byte % ØØØØØØØØ
. d7d9  ØØ      .byte % ØØØØØØØØ
. d7da  ØØ      .byte % ØØØØØØØØ
. d7db  ØØ      .byte % ØØØØØØØØ
. d7dc  fØ      .byte % 1111ØØØØ
. d7dd  fØ      .byte % 1111ØØØØ
. d7de  fØ      .byte % 1111ØØØØ
. d7df  fØ      .byte % 1111ØØØØ
. d7eØ                          Zeichen: 252
. d7eØ  Øf      .byte % ØØØØ1111
. d7e1  Øf      .byte % ØØØØ1111
. d7e2  Øf      .byte % ØØØØ1111
. d7e3  Øf      .byte % ØØØØ1111
. d7e4  ØØ      .byte % ØØØØØØØØ
. d7e5  ØØ      .byte % ØØØØØØØØ
. d7e6  ØØ      .byte % ØØØØØØØØ
. d7e7  ØØ      .byte % ØØØØØØØØ
. d7e8                          Zeichen: 253
. d7e8  18      .byte % ØØØ11ØØØ
. d7e9  18      .byte % ØØØ11ØØØ
. d7ea  18      .byte % ØØØ11ØØØ
. d7eb  f8      .byte % 11111ØØØ
. d7ec  f8      .byte % 11111ØØØ
. d7ed  ØØ      .byte % ØØØØØØØØ
. d7ee  ØØ      .byte % ØØØØØØØØ
```

```
. d7ef  ØØ       .byte % ØØØØØØØØ
. d7fØ                            Zeichen: 254
. d7fØ  fØ       .byte % 11110000
. d7f1  fØ       .byte % 11110000
. d7f2  fØ       .byte % 11110000
. d7f3  fØ       .byte % 11110000
. d7f4  ØØ       .byte % ØØØØØØØØ
. d7f5  ØØ       .byte % ØØØØØØØØ
. d7f6  ØØ       .byte % ØØØØØØØØ
. d7f7  ØØ       .byte % ØØØØØØØØ
. d7f8                            Zeichen: 255
. d7f8  fØ       .byte % 11110000
. d7f9  fØ       .byte % 11110000
. d7fa  fØ       .byte % 11110000
. d7fb  fØ       .byte % 11110000
. d7fc  Øf       .byte % ØØØØ1111
. d7fd  Øf       .byte % ØØØØ1111
. d7fe  Øf       .byte % ØØØØ1111
. d7ff  Øf       .byte % ØØØØ1111

. d8ØØ  Ø8 Ø9    .byte $Ø8 $Ø9  (Neue Betriebssystem-Version)
-------------------------------------------------------------------------
. d8ØØ  Ø4 18    .byte $Ø4 $18  (Alte Betriebssystem-Version)
-------------------------------------------------------------------------
. d8Ø2  :
. d8Ø2  *** BILDSCHIRM-ZEILENADRESSEN ***
. d8Ø2  :
. d8Ø2  ØØ 28 5Ø 78 aØ c8 fØ 18  Low-Bytes
. d8Øa  4Ø 68 9Ø b8 eØ Ø8 3Ø 58
. d812  8Ø a8 dØ f8 2Ø 48 7Ø 98
. d81a  cØ
. d81b  Øc Øc Øc Øc Øc Øc Øc Ød  High-Bytes
. d823  Ød Ød Ød Ød Ød Øe Øe Øe
. d82b  Øe Øe Øe Øe Øf Øf Øf Øf
. d833  Øf

. d834  :
. d834  *** SCREEN ***
. d834  :
. d834  a2 28    ldx #$28      Spaltenzahl 4Ø nach X-Reg.
. d836  aØ 19    ldy #$19      Zeilenzahl 25 nach Y-Reg.
. d838  6Ø       rts           ENDE

. d839  :
. d839  *** PLOT ***
. d839  :
. d839  bØ Øe    bcs $d849     Carry=1 ?, dann $d849
. d83b  86 cd    stx $cd       Schreibt Zeile, in der sich der Cursor befindet
. d83d  86 c4    stx $c4       Schreibt X-Position des Cursors für Eingabe
. d83f  84 ca    sty $ca       Schreibt Cursorspalte der aktuellen Zeile
. d841  84 c5    sty $c5       Schreibt Y-Position des Cursors für Eingabe
. d843  2Ø 7Ø de jsr $de7Ø     Stellt vollen Bildschirm her
```

```
. d846  20 a8 d8 jsr $d8a8    Setzt Cursorzeiger
. d849  a6 cd    ldx $cd      Holt Cursorzeile nach X-Reg.
. d84b  a4 ca    ldy $ca      Holt Cursorspalte nach Y-Reg.
. d84d  60       rts          ENDE

. d84e  :
. d84e  *** EDITOR RESET ***
. d84e  :
. d84e  a9 0c    lda #$0c     Setzt Video-RAM-Anfang
. d850  8d 3e 05 sta $053e    ... auf Page 12
. d853  a9 03    lda #$03     Ausgabekanal
. d855  85 99    sta $99      ... gleich 3 (Bildschirm)
. d857  a9 00    lda #$00     Eingabekanal
. d859  85 98    sta $98      ... gleich 0 (Tastatur)
. d85b  8d 47 05 sta $0547    Shift und C= wirksam
. d85e  85 83    sta $83      Textmodus einschalten
. d860  85 ef    sta $ef      Tastaturpuffer auf 0 Zeichen setzen
. d862  85 f0    sta $f0      Print-Warteflag initialisieren
. d864  a9 7a    lda #$7a     Initialisiert
. d866  8d 45 05 sta $0545    ... Keylog-Vektor
. d869  a9 db    lda #$db     ... ($db7a)...
. d86b  8d 46 05 sta $0546
. d86e  a9 0a    lda #$0a     Akku gleich 10
. d870  8d 3f 05 sta $053f    Schreibt Maximalwert für Tastaturpuffer
. d873  8d 4a 05 sta $054a    Schreibt Cursorzähler
. d876  8d 42 05 sta $0542    Schreibt Anlaufverzögerung für Repeat
. d879  a9 80    lda #$80     Akku gleich 128
. d87b  8d 40 05 sta $0540    Schreibt Flag für Repeat
. d87e  a9 10    lda #$10     Akku gleich 16
. d880  8d 3b 05 sta $053b    Zeichenfarbe gleich schwarz
. d883  a9 04    lda #$04     Schreibt Geschwindigkeit
. d885  8d 41 05 sta $0541    ... für Repeat
. d888  20 70 de jsr $de70    Stellt vollen Bildschirm her
. d88b  :
. d88b  *** CLEAR SCREEN ***
. d88b  :
. d88b  20 9a d8 jsr $d89a    HOME
. d88e  20 aa d8 jsr $d8aa    Setzt Cursorzeiger auf Zeile (X-Reg.)
. d891  20 f7 da jsr $daf7    Löscht Zeile (X-Reg.)
. d894  ec e5 07 cpx $07e5    Letzte
. d897  e8       inx          ... Zeile erreicht ?
. d898  90 f4    bcc $d88e    Nein, dann $d88e
. d89a  :
. d89a  *** HOME ***
. d89a  :
. d89a  ae e6 07 ldx $07e6    Holt oberste Zeile
. d89d  86 cd    stx $cd      Schreibt Zeile, in der sich der Cursor befindet
. d89f  86 c4    stx $c4      Schreibt X-Position des Cursors für Eingabe
. d8a1  ac e7 07 ldy $07e7    Holt linken Rand
. d8a4  84 ca    sty $ca      Schreibt Cursorspalte der aktuellen Zeile
. d8a6  84 c5    sty $c5      Schreibt Y-Position des Cursors für Eingabe
. d8a8  a6 cd    ldx $cd      Holt Zeile in der sich der Cursor befindet
```

```
. d8aa  bd Ø2 d8 lda $d8Ø2,x  Holt Zeilenadresse (Low) aus Zeilentabelle
. d8ad  85 c8    sta $c8      Schreibt Zeilenanfangsadresse (Low)
. d8af  bd 1b d8 lda $d81b,x  Holt Zeilenadresse (High) aus Zeilentabelle
. d8b2  85 c9    sta $c9      Schreibt Zeilenanfangsadresse (High)
. d8b4  a5 c8    lda $c8      Holt Zeilenanfangsadresse (Low)
. d8b6  85 ea    sta $ea      Schreibt Adresse des Bildschirm-Farb-RAM (Low)
. d8b8  a5 c9    lda $c9      Holt Zeilenanfangsadresse (High)
. d8ba  29 Ø3    and #$Ø3
. d8bc  Ø9 Ø8    ora #$Ø8
. d8be  85 eb    sta $eb      Schreibt Adresse des Bildschirm-Farb-RAM (Low)
. d8cØ  6Ø       rts          ENDE
```

```
. d8c1  :
. d8c1  *** CODE AUS TASTENPUFFER HOLEN ***  (Alte Betriebssystem-Version)
. d8c1  :
. d8c1  ac 5d Ø5 ldy $Ø55d    Holt Funktionstasten-Index
. d8c4  fØ Øe    beq $d8d4    Keine Funktionstaste gedrückt ?, dann $d8d5
. d8c6  ac 5e Ø5 ldy $Ø55e    Holt Zeiger in Textpuffer
. d8c9  b9 67 Ø5 lda $Ø567,y  Holt Zeichen aus Text
. d8cc  ce 5d Ø5 dec $Ø55d    Erniedrigt Zeichenindex
. d8cf  ee 5e Ø5 inc $Ø55e    Erhöht Zeiger in Textpuffer
. d8d2  58       cli
. d8d3  6Ø       rts          ENDE

. d8d4  ac 27 Ø5 ldy $Ø527    Holt 1. Code aus Textpuffer
. d8d7  a2 ØØ    ldx #$ØØ     Offset gleich Ø
. d8d9  bd 28 Ø5 lda $Ø528,x  Schiebt
. d8dc  9d 27 Ø5 sta $Ø527,x  ... Pufferinhalt
. d8df  e8       inx          ... nach...
. d8eØ  e4 ef    cpx $ef      Gesamten Puffer verschoben ?
. d8e2  dØ f5    bne $d8d9    Nein, dann $d8d9
. d8e4  c6 ef    dec $ef      Erniedrigt Index um 1
. d8e6  98       tya          Holt 1. Code nach Akku
. d8e7  58       cli
. d8e8  18       clc
. d8e9  6Ø       rts          ENDE
```

```
. d8c1  :
. d8c1  *** CODE AUS TASTENPUFFER HOLEN ***  (Neue Betriebssystem-Version)
. d8c1  :
. d8c1  ae 5d Ø5 ldx $Ø55d    Holt Funktionstasten-Index
. d8c4  fØ Øf    beq $d8d5    Keine Funktionstaste gedrückt ?, dann $d8d5
. d8c6  ac 5e Ø5 ldy $Ø55e    Holt Zeiger in Textpuffer
. d8c9  b9 67 Ø5 lda $Ø567,y  Holt Zeichen aus Text
. d8cc  ce 5d Ø5 dec $Ø55d    Erniedrigt Zeichenindex
. d8cf  ee 5e Ø5 inc $Ø55e    Erhöht Zeiger in Textpuffer
. d8d2  58       cli
. d8d3  18       clc
. d8d4  6Ø       rts          ENDE
```

```
. d8d5  ac 27 Ø5 ldy $Ø527    Holt 1. Code aus Textpuffer
. d8d8  ea       nop
. d8d9  bd 28 Ø5 lda $Ø528,x  Schiebt
. d8dc  9d 27 Ø5 sta $Ø527,x  ... Pufferinhalt
. d8df  e8       inx          ... nach...
. d8eØ  e4 ef    cpx $ef      Gesamten Puffer verschoben ?
. d8e2  dØ f5    bne $d8d9    Nein, dann $d8d9
. d8e4  c6 ef    dec $ef      Erniedrigt Index um 1
. d8e6  98       tya          Holt 1. Code nach Akku
. d8e7  58       cli
. d8e8  18       clc
. d8e9  6Ø       rts          ENDE

. d8ea  :
. d8ea  *** EINGABE VOM BILDSCHIRM ***
. d8ea  :
. d8ea  2Ø 49 dc jsr $dc49    PRINT
. d8ed  2Ø b4 d8 jsr $d8b4    Setzt Zeiger in Farb-RAM
. d8fØ  a4 ca    ldy $ca      Holt Cursorspalte
. d8f2  b1 ea    lda ($ea),y  Holt Zeichen aus aktueller Bildschirmzeile
. d8f4  48       pha          Zeichen auf Stack retten
. d8f5  ad 3b Ø5 lda $Ø53b    Holt Zeichenfarbe, Helligkeit und Blink-Flag
. d8f8  91 ea    sta ($ea),y  In aktuelle Zeile schreiben
. d8fa  98       tya          Holt Cursorspalte,
. d8fb  18       clc          ... addiert
. d8fc  65 c8    adc $c8      ... Zeilenanfangsadresse,
. d8fe  8d Ød ff sta $ffØd    ... subtrahiert
. d9Ø1  a5 c9    lda $c9      ... $Øb ØØ (2816)
. d9Ø3  69 ØØ    adc #$ØØ     ... und schreibt
. d9Ø5  e9 Øb    sbc #$Øb     ... aktuelle Cursoradresse
. d9Ø7  8d Øc ff sta $ffØc    ... in TED-Register
. d9Øa  a5 ef    lda $ef      Holt Anzahl der Zeichen im Tastaturpuffer
. d9Øc  Ød 5d Ø5 ora $Ø55d    Oder-Verknüpfung mit Funktionstextlänge
. d9Øf  fØ f9    beq $d9Øa    Beide gleich Ø ?, dann warten...
. d911  68       pla          Holt Code vom Stack
. d912  91 ea    sta ($ea),y  In aktuelle Zeile schreiben
. d914  a9 ff    lda #$ff     Schaltet
. d916  8d Øc ff sta $ffØc    ... Cursor
. d919  8d Ød ff sta $ffØd    ... ab
. d91c  2Ø c1 d8 jsr $d8c1    Holt Code aus Tastenpuffer
. d91f  c9 83    cmp #$83     'RUN'-Taste ?
. d921  dØ 1Ø    bne $d933    Nein, dann $d933
. d923  a2 Ø9    ldx #$Ø9     Reserviert
. d925  78       sei          ... 9 Zeichen
. d926  86 ef    stx $ef      ... im Tastaturpuffer
. d928  bd 29 e1 lda $e129,x  Holt Text für 'RUN'-Taste
. d92b  9d 26 Ø5 sta $Ø526,x  Schreibt Text in Tastaturpuffer
. d92e  ca       dex          Schon alle Zeichen kopiert ?
. d92f  dØ f7    bne $d928    Nein, dann $d928
. d931  fØ ba    beq $d8ed    Zum Schleifenanfang

. d933  c9 Ød    cmp #$Ød     'RETURN'-Taste ?
```

```
. d935  d0 b3     bne $d8ea     Nein, dann $d8ea
. d937  85 c7     sta $c7       Schreibt Flag für Zeilenübernahme
. d939  20 95 df  jsr $df95     Setzt Cursor an Zeilenende
. d93c  8e 49 05  stx $0549     Bildschirmzeile
. d93f  20 87 df  jsr $df87     Setzt Cursor an Zeilenanfang
. d942  a9 00     lda #$00      Kein
. d944  85 cb     sta $cb       ... Anführungszeichen-Modus
. d946  ac e7 07  ldy $07e7     Holt linken Rand
. d949  a5 c4     lda $c4       Holt letzen Zeilenwert
. d94b  30 13     bmi $d960     Flag für 'ganze Zeile' gesetz ?, dann $d960
. d94d  c5 cd     cmp $cd       Vergleich mit Cursorzeile
. d94f  90 0f     bcc $d960     Zeilenwert kleiner Cursorzeile ?, dann $d960
. d951  a4 c5     ldy $c5       Holt letzten Spaltenwert
. d953  cd 49 05  cmp $0549     Spaltenwert gleich logische Endzeile ?
. d956  d0 04     bne $d95c     Nein, dann $d95c
. d958  c4 c3     cpy $c3       Spaltenwert gleich Zeilenende ?
. d95a  f0 02     beq $d95e     Ja, dann $d95e
. d95c  b0 11     bcs $d96f     Spaltenwert größer ?, dann $d96f
. d95e  85 cd     sta $cd       Schreibt Cursorzeile
. d960  84 ca     sty $ca       Schreibt Cursorspalte
. d962  4c 77 d9  jmp $d977     Zeile übernehmen

. d965  :
. d965  *** ZEICHEN VOM BILDSCHIRM HOLEN ***
. d965  :
. d965  98        tya           X-Reg.
. d966  48        pha           ... und
. d967  8a        txa           ... Y-Reg.
. d968  48        pha           ... auf Stack retten
. d969  a5 c7     lda $c7       Holt Flag für Zeilenübernahme
. d96b  f0 c4     beq $d931     Flag nicht gesetzt ?, dann Zeileneingabe
. d96d  10 08     bpl $d977     Flag größer $80 ?, dann Zeilenübernahme
. d96f  a9 00     lda #$00      Flag für Zeilenübernahme
. d971  85 c7     sta $c7       ... auf $00 setzen
. d973  4c 74 cf  jmp $cf74     Zeilenübernahme abschließen

. d976  ea        nop

. d977  20 a8 d8  jsr $d8a8     Setzt Cursorzeiger
. d97a  20 2f df  jsr $df2f     Liest Zeichen vom Bildschirm
. d97d  85 ce     sta $ce       Zeichen zwischenspeichern
. d97f  29 3f     and #$3f
. d981  06 ce     asl $ce       Bit 6
. d983  24 ce     bit $ce       ... gesetzt ?
. d985  10 02     bpl $d989     Ja, dann $d989
. d987  09 80     ora #$80      Bit 7 gesetzt ?
. d989  90 04     bcc $d98f     Nein, dann $d98f
. d98b  a6 cb     ldx $cb       Anführungszeichen-Modus ?
. d98d  d0 04     bne $d993     Ja, dann $d993
. d98f  70 02     bvs $d993     Bit 5 gesetzt ?, dann $d993
. d991  09 40     ora #$40
. d993  20 ba d9  jsr $d9ba     Anführungszeichen-Modus falls '"'
```

```
. d996  a4 cd     ldy $cd       Holt Cursorzeile
. d998  cc 49 05  cpy $0549     Cursorzeile kleiner logische Endzeile ?
. d99b  90 0a     bcc $d9a7     Ja, dann $d9a7
. d99d  a4 ca     ldy $ca       Holt Cursorspalte
. d99f  c4 c3     cpy $c3       Cursorspalte kleiner Zeilenende ?
. d9a1  90 04     bcc $d9a7     Ja, dann $d9a7
. d9a3  66 c7     ror $c7       Flag für Zeilenübernahme
. d9a5  30 03     bmi $d9aa     Zeilenübernahme abschließen

. d9a7  20 bf df  jsr $dfbf     Cursor um 1 Zeichen nach rechts verschieben
. d9aa  c9 de     cmp #$de      Zeichen für PI ?
. d9ac  d0 02     bne $d9b0     Nein, dann $d9b0
. d9ae  a9 ff     lda #$ff      Durch $ff ersetzen
. d9b0  85 ce     sta $ce       Übernommenes Zeichen zwischenspeichern
. d9b2  68        pla           X-Reg.
. d9b3  aa        tax           ... und
. d9b4  68        pla           ... Y-Reg.
. d9b5  a8        tay           ... Vom Stack holen
. d9b6  a5 ce     lda $ce       Übernommenes Zeichen holen
. d9b8  18        clc           Carry gleich 0
. d9b9  60        rts           ENDE

. d9ba  :
. d9ba  *** ANFÜHRUNGSZEICHEN-MODUS SETZEN/LÖSCHEN ***
. d9ba  :
. d9ba  c9 22     cmp #$22      Zeichen gleich Anführungszeichen (") ?
. d9bc  d0 08     bne $d9c6     Nein, dann $d9c6
. d9be  a5 cb     lda $cb       Holt Flag für Anführungszeichen-Modus
. d9c0  49 01     eor #$01      Flag setzen/löschen
. d9c2  85 cb     sta $cb       Schreibt Flag für Anführungszeichen-Modus
. d9c4  a9 22     lda #$22      Holt Anführungszeichen (")
. d9c6  60        rts           ENDE

. d9c7  :
. d9c7  *** ABSCHLUSS VON PRINT ***
. d9c7  :
. d9c7  a5 ce     lda $ce       Holt ausgegebenes Zeichen
. d9c9  8d eb 07  sta $07eb     Schreibt letztes ausgegebenes Zeichen
. d9cc  68        pla           Holt Y-Reg.
. d9cd  a8        tay           ... vom Stack
. d9ce  a5 cf     lda $cf       Holt Zähler für Insert
. d9d0  f0 02     beq $d9d4     Zähler gleich Null ?, dann $d9d4
. d9d2  46 cb     lsr $cb       Anführungszeichen-Modus löschen
. d9d4  68        pla           Holt X-Reg.
. d9d5  aa        tax           ... und
. d9d6  68        pla           ... Akku vom Stack
. d9d7  18        clc           ... Carry = 0
. d9d8  60        rts           ENDE

. d9d9  09 40     ora #$40
. d9db  a6 c2     ldx $c2       Reverse-Flag gesetzt ?
. d9dd  f0 02     beq $d9e1     Nein, dann $d9e1
```

```
. d9df  09 80     ora #$80      Setzt Bit 7
. d9e1  a6 cf     ldx $cf       Holt Zähler für Insert
. d9e3  f0 02     beq $d9e7     Zähler gleich Null ?, dann $d9e7
. d9e5  c6 cf     dec $cf       Insert-Zähler um 1 erniedrigen
. d9e7  2c ea 07  bit $07ea     Auto-Insert aktiv ?
. d9ea  10 09     bpl $d9f5     Nein, dann $d9f5
. d9ec  48        pha           Zeichen auf Stack retten
. d9ed  20 ce dd  jsr $ddce     Zeichen einfügen
. d9f0  a2 00     ldx #$00      Initialisiert
. d9f2  86 cf     stx $cf       ... Insert-Zähler
. d9f4  68        pla           Zeichen vom Stack holen
. d9f5  20 01 e0  jsr $e001     Zeichen auf Bildschirm ausgeben
. d9f8  cc e8 07  cpy $07e8     Rechter Rand erreicht ?
. d9fb  90 0c     bcc $da09     Nein, dann $da09
. d9fd  a6 cd     ldx $cd       Cursorzeile gleich
. d9ff  ec e5 07  cpx $07e5     ... unterste Bildschirmzeile ?
. da02  90 05     bcc $da09     Nein, dann $da09
. da04  2c e9 07  bit $07e9     Scrolling erlaubt ?
. da07  30 17     bmi $da20     Nein, dann fertig
. da09  20 a8 d8  jsr $d8a8     Setzt Cursorzeiger
. da0c  20 bf df  jsr $dfbf     Cursor um 1 Zeichen nach rechts verschieben
. da0f  90 0f     bcc $da20
. da11  20 39 df  jsr $df39     (GETBIT) Zeilentabelle
. da14  b0 09     bcs $da1f
. da16  38        sec
. da17  2c e9 07  bit $07e9     Scrolling erlaubt ?
. da1a  70 04     bvs $da20     Ja, dann fertig
. da1c  20 5e da  jsr $da5e     Fügt Leerzeile ein
. da1f  18        clc           Carry = 0
. da20  60        rts           ENDE

. da21  :
. da21  *** SCROLL AUF ***
. da21  :
. da21  a6 cd     ldx $cd       Holt Cursorzeile
. da23  ec e5 07  cpx $07e5     Cursorzeile kleiner unterste Bildschirmzeile ?
. da26  90 10     bcc $da38     Ja, dann $da38
. da28  2c e9 07  bit $07e9     Scrolling erlaubt ?
. da2b  10 07     bpl $da34     Ja, dann $da34
. da2d  ad e6 07  lda $07e6     Holt oberste Bildschirmzeile
. da30  85 cd     sta $cd       Schreibt Cursorzeile
. da32  b0 06     bcs $da3a     Springt immer nach $da3a

. da34  20 89 da  jsr $da89     Scrollt auf
. da37  18        clc
. da38  e6 cd     inc $cd       Cursorzeile um 1 erniedrigen
. da3a  4c a8 d8  jmp $d8a8     Setzt Cursorzeiger

. da3d  :
. da3d  *** BILDSCHIRMZEILE UMKOPIEREN ***
. da3d  :
. da3d  bd 02 d8  lda $d802,x   Holt Adresse der Zeile (Low)
```

```
. da4Ø  85 a9     sta $a9       Schreibt Zeiger in Farb-RAM (Low)
. da42  85 cØ     sta $cØ       Schreibt Zeiger in Zeichen-RAM (Low)
. da44  bd 1b d8  lda $d81b,x   Holt Adresse der Zeile (High)
. da47  85 c1     sta $c1       Schreibt Zeiger in Zeichen-RAM (High)
. da49  29 Ø3     and #$Ø3
. da4b  Ø9 Ø8     ora #$Ø8
. da4d  85 aa     sta $aa       Schreibt Zeiger in Farb-RAM (High)
. da4f  b1 cØ     lda ($cØ),y   Holt Code im Zeichen-RAM
. da51  91 c8     sta ($c8),y   Schreibt Code in Cursorzeile
. da53  b1 a9     lda ($a9),y   Holt Code in Farb-RAM
. da55  91 ea     sta ($ea),y   Schreibt Code in Cursorzeile
. da57  cc e8 Ø7  cpy $Ø7e8     Rechter Rand erreicht ?
. da5a  c8        iny
. da5b  9Ø f2     bcc $da4f     Nein, dann $da4f
. da5d  6Ø        rts           ENDE

. da5e  :
. da5e  *** LEERZEILE EINFÜGEN ***
. da5e  :
. da5e  a6 c4     ldx $c4       Holt letzten Zeilenwert
. da6Ø  3Ø Ø6     bmi $da68     Flag für 'Ganze Zeile' gesetzt ?, dann $da68
. da62  e4 cd     cpx $cd       Letzter Zeilenwert kleiner Cursorzeile ?
. da64  9Ø Ø2     bcc $da68     Ja, dann $da68
. da66  e6 c4     inc $c4       Letzten Zeilenwert um 1 erhöhen
. da68  ae e5 Ø7  ldx $Ø7e5     Setzt Zeilenzeiger auf unterste Zeile
. da6b  2Ø aa d8  jsr $d8aa     Setzt Cursorzeile auf Zeile (X-Reg.)
. da6e  ac e7 Ø7  ldy $Ø7e7     Holt linken Rand
. da71  e4 cd     cpx $cd       Cursorzeile erreicht ?
. da73  fØ Øe     beq $da83     Ja, dann fertig
. da75  ca        dex           Cursorzeile um 1 erniedrigen
. da76  2Ø 3b df  jsr $df3b     (GETBIT) Zeilentabelle
. da79  e8        inx           Cursorzeile um 1 erhöhen
. da7a  2Ø 48 df  jsr $df48     (CLRBIT) Zeilentabelle
. da7d  ca        dex           Cursorzeile um 1 erniedrigen
. da7e  2Ø 3d da  jsr $da3d     Kopiert Zeile (X-Reg.) nach Zeile (X-Reg.+1)
. da81  bØ e8     bcs $da6b     Springt immer nach $da6b

. da83  2Ø f7 da  jsr $daf7     Löscht Zeile (X-Reg.)
. da86  4c 59 df  jmp $df59     (SETBIT) Zeilentabelle

. da89  ae e6 Ø7  ldx $Ø7e6     Holt oberste Bildschirmzeile
. da8c  e8        inx
. da8d  2Ø 3b df  jsr $df3b     (GETBIT) Zeilentabelle
. da9Ø  9Ø Øc     bcc $da9e     Bit nicht gesetzt ?, dann $da9e
. da92  ec e5 Ø7  cpx $Ø7e5     Vergleicht mit unterster Bildschirmzeile
. da95  9Ø f5     bcc $da8c     Unterste Bildschirmzeile größer ?, dann $da8c
. da97  ae e6 Ø7  ldx $Ø7e6     Holt oberste Bildschirmzeile
. da9a  e8        inx           Oberste Bildschirmzeile um 1 erhöhen
. da9b  2Ø 4a df  jsr $df4a     (CLRBIT) Zeilentabelle
. da9e  c6 cd     dec $cd       Erniedrigt Cursorzeile um 1
. daaØ  24 c4     bit $c4       Letzter Zeilenwert Ø ?
. daa2  3Ø Ø2     bmi $daa6     Ja, dann $daa6
```

```
. daa4  c6 c4     dec $c4       Letzten Zeilenwert um 1 erniedrigen
. daa6  ae e6 Ø7  ldx $Ø7e6     Holt oberste Bildschirmzeile
. daa9  e4 fe     cpx $fe       Vergleich mit $fe
. daab  bØ Ø2     bcs $daaf     Oberste Bildschirmzeile größer ?, dann $daaf
. daad  c6 fe     dec $fe
. daaf  2Ø c5 da  jsr $dac5     Ab Zeile (X-Reg.) aufwärts scrollen
. dab2  ae e6 Ø7  ldx $Ø7e6     Holt oberste Bildschirmzeile
. dab5  2Ø 3b df  jsr $df3b     (GETBIT) Zeilentabelle
. dab8  Ø8        php
. dab9  2Ø 4a df  jsr $df4a     (CLRBIT) Zeilentabelle
. dabc  28        plp
. dabd  9Ø Ø5     bcc $dac4
. dabf  2c ec Ø7  bit $Ø7ec     Scroll-Sperre gesetzt ?
. dac2  3Ø c5     bmi $da89     Ja, dann $da89
. dac4  6Ø        rts           ENDE

. dac5  2Ø aa d8  jsr $d8aa     Setzt Cursorzeiger auf Zeile (X-Reg.)
. dac8  ac e7 Ø7  ldy $Ø7e7     Holt linken Rand
. dacb  ec e5 Ø7  cpx $Ø7e5     Unterste Bildschirmzeile erreicht ?
. dace  bØ Øe     bcs $dade     Ja, dann $dade
. dadØ  e8        inx           Cursorzeile um 1 erhöhen
. dad1  2Ø 3b df  jsr $df3b     (GETBIT) Zeilentabelle
. dad4  ca        dex           Cursorzeile um 1 erniedrigen
. dad5  2Ø 48 df  jsr $df48     (CLRBIT) Zeilentabelle
. dad8  e8        inx           Cursorzeile um 1 erhöhen
. dad9  2Ø 3d da  jsr $da3d     Kopiert Zeile (X-Reg.) nach Zeile (X-Reg.+1)
. dadc  bØ e7     bcs $dac5     Springt immer nach $dac5

. dade  2Ø f7 da  jsr $daf7     Löscht Zeile (X-Reg.)
. dae1  a9 7f     lda #$7f      Akku gleich %Ø1111111
. dae3  2Ø 7Ø db  jsr $db7Ø     Fragt Dekoder ab
. dae6  c9 df     cmp #$df      Commodore-Taste gedrückt ?
. dae8  dØ Ø9     bne $daf3     Nein, dann $daf3
. daea  aØ ØØ     ldy #$ØØ      Warteschleife
. daec  ea        nop           ... für
. daed  ca        dex           ... zirka
. daee  dØ fc     bne $daec     ... Ø.5 Sekunden...
. dafØ  88        dey
. daf1  dØ f9     bne $daec
. daf3  6Ø        rts           ENDE

. daf4  ea        nop
. daf5  ea        nop
. daf6  ea        nop

. daf7  ac e7 Ø7  ldy $Ø7e7     Holt linken Rand
. dafa  2Ø 4a df  jsr $df4a     (CLRBIT) Zeilentabelle
. dafd  2Ø aa d8  jsr $d8aa     Cursorzeiger auf Zeile (X-Reg.) setzen
. dbØØ  88        dey           Linken Rand um 1 erniedrigen
. dbØ1  c8        iny           Linken Rand um 1 erhöhen
. dbØ2  a9 2Ø     lda #$2Ø      Holt Leerzeichen
. dbØ4  91 c8     sta ($c8),y   Schreibt Leerzeichen an Cursorposition
```

```
. db06  ad 3b 05 lda $053b    Holt Farbe und Helligkeit
. db09  91 ea    sta ($ea),y  Schreibt Farb-Parameter in Farb-RAM
. db0b  cc e8 07 cpy $07e8    Rechter Rand erreicht ?
. db0e  d0 f1    bne $db01    Nein, dann $db01
. db10  60       rts          ENDE

. db11  :
. db11  *** TASTATURABFRAGE ***
. db11  :
. db11  a9 00    lda #$00     Initialisiert
. db13  8d 43 05 sta $0543    ... SHIFT-Flag
. db16  a0 40    ldy #$40     Setzt Offset
. db18  84 c6    sty $c6      ... in Tabelle auf $40
. db1a  20 70 db jsr $db70    Fragt Dekoder ab
. db1d  aa       tax          Taste nach X-Reg.
. db1e  e0 ff    cpx #$ff     Taste gedrückt ?
. db20  d0 03    bne $db25    Ja, dann $db25
. db22  4c 01 dc jmp $dc01    Springt nach $dc01

. db25  a0 00    ldy #$00     Initialisiert Offset
. db27  a9 26    lda #$26     Schreibt Adresse
. db29  85 ec    sta $ec      ... der Codetabelle
. db2b  a9 e0    lda #$e0     ... in
. db2d  85 ed    sta $ed      ... Zeiger
. db2f  a9 fe    lda #$fe     Code (%11111110) für Dekoder-Eingang
. db31  a2 08    ldx #$08     Holt Bitzähler
. db33  48       pha
. db34  68       pla
. db35  48       pha
. db36  20 70 db jsr $db70    Fragt Dekoder ab
. db39  85 ee    sta $ee      Ergebnis zwischenspeichern
. db3b  68       pla
. db3c  48       pha
. db3d  20 70 db jsr $db70    Fragt Dekoder ab
. db40  c5 ee    cmp $ee      Gleiches Ergebnis ?
. db42  d0 f0    bne $db34    Nein, dann erneut abfragen
. db44  4a       lsr          Schiebt Bit 7 ins Carry
. db45  b0 16    bcs $db5d    Bit gesetzt ?, dann $db5d
. db47  48       pha          Akku auf Stack retten
. db48  b1 ec    lda ($ec),y  Holt Code aus Tabelle
. db4a  c9 05    cmp #$05     Code größer als 4 ?
. db4c  b0 0c    bcs $db5a    Ja, dann $db5a
. db4e  c9 03    cmp #$03     Code gleich STOP-Code ?
. db50  f0 08    beq $db5a    Ja, dann $db5a
. db52  0d 43 05 ora $0543    Code mit SHIFT-Flag ODER-Verknüpfen
. db55  8d 43 05 sta $0543    Schreibt 1 für SHIFT, 2 für C= oder 4 für CTRL
. db58  10 02    bpl $db5c    Springt immer nach $db5c

. db5a  84 c6    sty $c6      Schreibt Offset in Code-Tabelle
. db5c  68       pla          Holt Akku vom Stack
. db5d  c8       iny          Erhöht Offset um 1
. db5e  c0 41    cpy #$41     Alle Tasten abgefragt ?
```

```
. db6Ø  bØ Ø8     bcs $db6a     Ja, dann $db6a
. db62  ca        dex           Erniedrigt Bitzähler um 1
. db63  dØ df     bne $db44     Noch nicht Ø ?, dann nächstes Bit bearbeiten
. db65  38        sec           Carry gleich 1
. db66  68        pla           Holt Dekoder-Eingang
. db67  2a        rol           Dekoder-Eingang nach links rotieren
. db68  dØ c7     bne $db31     Springt immer nach $db31

. db6a  68        pla           Korrigiert Stack
. db6b  a5 c6     lda $c6       Holt Offset in Codetabelle
. db6d  6c 45 Ø5  jmp ($Ø545)   KEYLOG

. db7Ø  :
. db7Ø  *** DEKODER-ABFRAGE ***
. db7Ø  :
. db7Ø  8d 3Ø fd  sta $fd3Ø     Tastaturdekoder Eingang
. db73  8d Ø8 ff  sta $ffØ8     Tastaturdekoder Ausgang
. db76  ad Ø8 ff  lda $ffØ8     Holt Code
. db79  6Ø        rts           ENDE

. db7a  :
. db7a  *** KEYLOG ***
. db7a  :
. db7a  ad 43 Ø5  lda $Ø543     Holt SHIFT-Flag
. db7d  c9 Ø3     cmp #$Ø3      SHIFT und Commodore-Taste gedrückt ?
. db7f  dØ 19     bne $db9a     Nein, dann $db9a
. db81  ad 47 Ø5  lda $Ø547     SHIFT-C= gesperrt ?
. db84  3Ø 34     bmi $dbba     Ja, dann $dbba
. db86  ad 44 Ø5  lda $Ø544     Wiederholungsverzögerung abgelaufen ?
. db89  dØ 2f     bne $dbba     Nein, dann $dbba
. db8b  ad 13 ff  lda $ff13     Basisadresse
. db8e  49 Ø4     eor #$Ø4      ... des Zeichengenerators
. db9Ø  8d 13 ff  sta $ff13     ... ändern
. db93  a9 Ø8     lda #$Ø8      Setzt
. db95  8d 44 Ø5  sta $Ø544     ... Wiederholungsverzögerung
. db98  dØ 2Ø     bne $dbba     Springt immer nach $dbba

. db9a  Øa        asl
. db9b  c9 Ø8     cmp #$Ø8      CTRL-Taste ?
. db9d  9Ø 1Ø     bcc $dbaf     Nein, dann $dbaf
. db9f  a9 Ø6     lda #$Ø6      CTRL-S
. dba1  ae f7 Ø7  ldx $Ø7f7     ... gesperrt ?
. dba4  dØ Ø9     bne $dbaf     Ja, dann $dbaf
. dba6  a6 c6     ldx $c6       Holt Offset in Code-Tabelle
. dba8  eØ Ød     cpx #$Ød      CTRL-S ?
. dbaa  dØ Ø3     bne $dbaf     Nein, dann $dbaf
. dbac  86 fØ     stx $fØ       Setzt PRINT-Warteflag
. dbae  6Ø        rts           ENDE

. dbaf  aa        tax           Schreibt
. dbbØ  bd 1e eØ  lda $eØ1e,x   ... Adresse der
. dbb3  85 ec     sta $ec       ... der Codetabelle
```

```
. dbb5  bd 1f e0 lda $e01f,x  ... in
. dbb8  85 ed    sta $ed      ... Zeiger ($ec-$ed)
. dbba  a4 c6    ldy $c6      Holt Offset in Tabelle
. dbbc  b1 ec    lda ($ec),y  Holt Code aus Tabelle
. dbbe  aa       tax
. dbbf  cc f6 07 cpy $07f6    Selber Code wie beim letzten Mal ?
. dbc2  f0 07    beq $dbcb    Ja, dann $dbcb
. dbc4  a0 10    ldy #$10     Setzt Anlaufverzögerung
. dbc6  8c 42 05 sty $0542    ... für Repeat
. dbc9  d0 36    bne $dc01    Springt immer nach $dc01

. dbcb  29 7f    and #$7f
. dbcd  2c 40 05 bit $0540    Repeat-Flag gesetzt ?
. dbd0  30 16    bmi $dbe8    Ja, dann $dbe8
. dbd2  70 57    bvs $dc2b    Repeat abgeschaltet ?, dann $dc2b
. dbd4  c9 7f    cmp #$7f     Taste aktiv ?
. dbd6  f0 29    beq $dc01    Nein, dann $dc01
. dbd8  c9 14    cmp #$14     'DELETE'-Taste ?
. dbda  f0 0c    beq $dbe8    Ja, dann $dbe8
. dbdc  c9 20    cmp #$20     'SPACE'-Taste ?
. dbde  f0 08    beq $dbe8    Ja, dann $dbe8
. dbe0  c9 1d    cmp #$1d     'Cursor right'-Taste ?
. dbe2  f0 04    beq $dbe8    Ja, dann $dbe8
. dbe4  c9 11    cmp #$11     'Cursor down'-Taste ?
. dbe6  d0 43    bne $dc2b    Nein, dann $dc2b
. dbe8  ac 42 05 ldy $0542    Holt Repeat-Verzögerung
. dbeb  f0 05    beq $dbf2    Schon Null ?, dann $dbf2
. dbed  ce 42 05 dec $0542    Anlaufverzögerung runterzählen
. dbf0  d0 39    bne $dc2b    Noch immer nicht Null ?, dann $dc2b
. dbf2  ce 41 05 dec $0541    Holt Repeat-Geschwindigkeit
. dbf5  d0 34    bne $dc2b    Noch nicht abgelaufen ?, dann $dc2b
. dbf7  a0 04    ldy #$04     Setzt Repeat-Geschwindigkeit
. dbf9  8c 41 05 sty $0541    ... auf 4
. dbfc  a4 ef    ldy $ef      Holt Tastaturpuffer-Index
. dbfe  88       dey
. dbff  10 2a    bpl $dc2b

. dc01  ea       nop
. dc02  ea       nop

. dc03  4e 44 05 lsr $0544    Verschiebt letztes SHIFT-Muster der Tastatur
. dc06  a4 c6    ldy $c6      Holt Offset in Tabelle
. dc08  8c f6 07 sty $07f6    Schreibt letzten Offset
. dc0b  e0 ff    cpx #$ff     Noch eine Taste gedrückt ?
. dc0d  f0 1c    beq $dc2b    Nein, dann $dc2b
. dc0f  8a       txa          Code nach Akku
. dc10  a2 00    ldx #$00     Initialisiert
. dc12  86 f0    stx $f0      ... Print-Warteflag
. dc14  a2 07    ldx #$07     Taste gleich
. dc16  dd 41 dc cmp $dc41,x  ... Funktionstaste ?
. dc19  f0 11    beq $dc2c    Ja, dann $dc2c
. dc1b  ca       dex
```

```
. dc1c  10 f8     bpl $dc16
. dc1e  a6 ef     ldx $ef       Holt Tastaturpuffer-Index
. dc20  ec 3f 05  cpx $053f     Maximale Größe des Tastaturpuffers erreicht ?
. dc23  b0 06     bcs $dc2b     Ja, dann $dc2b
. dc25  9d 27 05  sta $0527,x   Schreibt Code in Tastaturpuffer
. dc28  e8        inx           Erhöht Index
. dc29  86 ef     stx $ef       ... des Tastaturpuffers
. dc2b  60        rts           ENDE

. dc2c  bd 5f 05  lda $055f,x   Holt Länge des Funktionstasten-Textes
. dc2f  8d 5d 05  sta $055d     Schreibt Länge in Zeichenzähler
. dc32  a9 00     lda #$00      Initialisiert Zeiger im Puffer
. dc34  ca        dex           Erniedrigt Code um 1
. dc35  30 06     bmi $dc3d     Kleiner Null ?, dann fertig
. dc37  18        clc           Addiert
. dc38  7d 5f 05  adc $055f,x   ... Längen
. dc3b  90 f7     bcc $dc34     Springt immer nach $dc34

. dc3d  8d 5e 05  sta $055e     Setzt Zeiger auf Textanfang
. dc40  60        rts           ENDE

. dc41  :
. dc41  *** CODES FÜR FUNKTIONSTASTEN ***
. dc41  :
. dc41  85 89 86 8a             .byte $85 $89 $86 $8a
. dc45  87 8b 88 8c             .byte $87 $8b $88 $8c

. dc49  :
. dc49  *** PRINT ***
. dc49  :
. dc49  85 ce     sta $ce       Speichert Code
. dc4b  48        pha           Rettet
. dc4c  8a        txa           ... X-Reg.
. dc4d  48        pha           ... Y-Reg.
. dc4e  98        tya           ... und Akku
. dc4f  48        pha           ... auf Stack
. dc50  a5 f0     lda $f0       CRTL-S aktiv ?
. dc52  d0 fc     bne $dc50     Ja, dann warten
. dc54  85 c7     sta $c7       Initialisiert Übernahme-Flag
. dc56  a9 d9     lda #$d9      Legt
. dc58  48        pha           ... Adresse des
. dc59  a9 c6     lda #$c6      ... Print-Abschluß
. dc5b  48        pha           ... auf Stack
. dc5c  a4 ca     ldy $ca       Holt Cursorspalte
. dc5e  a5 ce     lda $ce       Holt Code
. dc60  c9 0d     cmp #$0d      Code gleich 'RETURN' ?
. dc62  f0 28     beq $dc8c     Ja, dann $dc8c
. dc64  c9 8d     cmp #$8d      Code gleich 'SHIFT-RETURN' ?
. dc66  f0 24     beq $dc8c     Ja, dann $dc8c
. dc68  ae eb 07  ldx $07eb     Holt letzten ausgegebenen Code
. dc6b  e0 1b     cpx #$1b      Letzter Code gleich 'ESC' ?
. dc6d  d0 03     bne $dc72     Nein, dann $dc72
```

```
. dc6f  4c Ø6 de jmp $deØ6    'ESC'-Auswertung

. dc72  aa       tax          Code nach X-Reg.
. dc73  3Ø 14    bmi $dc89    SHIFT-Zeichen ?, dann $dc89
. dc75  c9 2Ø    cmp #$2Ø     Steuercode ?
. dc77  9Ø 2e    bcc $dca7    Ja, dann $dca7
. dc79  c9 6Ø    cmp #$6Ø     Code darstellbar ?
. dc7b  9Ø Ø4    bcc $dc81    Ja, dann $dc81
. dc7d  29 df    and #$df     Subtrahiert $2Ø vom Code
. dc7f  dØ Ø2    bne $dc83    Springt immer nach $dc83

. dc81  29 3f    and #$3f     Subtrahiert $4Ø vom Code
. dc83  2Ø ba d9 jsr $d9ba    Schaltet ggf. Anführungszeichen-Modus ein
. dc86  4c db d9 jmp $d9db    Zeichenausgabe & Abschluß

. dc89  4c 47 dd jmp $dd47    Bearbeitet SHIFT-Zeichen

. dc8c  :
. dc8c  *** RETURN-CODE ***
. dc8c  :
. dc8c  2Ø 95 df jsr $df95    Setzt Cursor ans Zeilenende
. dc8f  e8       inx          Erhöht Zeilennummer um 1
. dc9Ø  2Ø 4a df jsr $df4a    (CLRBIT) Zeilentabelle
. dc93  ac e7 Ø7 ldy $Ø7e7    Holt linken Rand
. dc96  84 ca    sty $ca      Schreibt Cursorzeiger
. dc98  2Ø 21 da jsr $da21    Scroll auf

. dc9b  :
. dc9b  *** ESC-O (FLAGS LÖSCHEN) ***
. dc9b  :
. dc9b  a9 ØØ    lda #$ØØ    Initialisiert
. dc9d  85 cf    sta $cf      ... Insert-Flag
. dc9f  85 c2    sta $c2      ... Reverse-Flag
. dca1  85 cb    sta $cb      ... Anführungszeichen-Modus
. dca3  8d 3c Ø5 sta $Ø53c    ... und Blink-Flag
. dca6  6Ø       rts          ENDE

. dca7  :
. dca7  *** STEUERCODES ***
. dca7  :
. dca7  c9 1b    cmp #$1b     'ESC'-Code ?
. dca9  fØ 4e    beq $dcf9    Ja, dann $dcf9
. dcab  a6 cf    ldx $cf      Insert-Modus aktiv ?
. dcad  fØ Ø3    beq $dcb2    Nein, dann $dcb2
. dcaf  4c df d9 jmp $d9df    Zeichen reverse ausgeben

. dcb2  c9 14    cmp #$14     'DELETE'-Code ?
. dcb4  dØ Ø3    bne $dcb9    Nein, dann $dcb9
. dcb6  4c 99 dd jmp $dd99    DELETE

. dcb9  a6 cb    ldx $cb      Anführungszeichen-Modus gesetzt ?
. dcbb  dØ f2    bne $dcaf    Ja, dann $dcaf
```

```
. dcbd  c9 12     cmp #$12      'RVS ON'-Code ?
. dcbf  d0 02     bne $dcc3     Nein, dann $dcc3
. dcc1  85 c2     sta $c2       Setzt Reverse-Flag
. dcc3  c9 13     cmp #$13      'HOME'-Code ?
. dcc5  d0 0b     bne $dcd2     Nein, dann $dcd2
. dcc7  cd eb 07  cmp $07eb     Letzter Code ebenfalls 'HOME'-Code ?
. dcca  d0 03     bne $dccf     Nein, dann $dccf
. dccc  20 70 de  jsr $de70     Window löschen
. dccf  4c 9a d8  jmp $d89a     Setzt Cursor in HOME-Position

. dcd2  c9 1d     cmp #$1d      'CURSOR right'-Code ?
. dcd4  f0 24     beq $dcfa     Ja, dann $dcfa
. dcd6  c9 11     cmp #$11      'CURSOR down'-Code ?
. dcd8  f0 26     beq $dd00     Ja, dann $dd00
. dcda  c9 0e     cmp #$0e      'TEXT'-Modus ?
. dcdc  f0 49     beq $dd27     Ja, dann $dd27
. dcde  c9 08     cmp #$08      'SHIFT-C=' aus ?
. dce0  f0 4c     beq $dd2e     Ja, dann $dd2e
. dce2  c9 09     cmp #$09      'SHIFT-C=' ein ?
. dce4  f0 4f     beq $dd35     Ja, dann $dd35
. dce6  a2 0f     ldx #$0f      Code
. dce8  dd 33 e1  cmp $e133,x   ... gleich Farbcode ?
. dceb  f0 04     beq $dcf1     Ja, dann $dcf1
. dced  ca        dex
. dcee  10 f8     bpl $dce8
. dcf0  60        rts           ENDE

. dcf1  48        pha           Akku auf Stack retten
. dcf2  20 8a cf  jsr $cf8a     Holt Farbcode aus Tabelle
. dcf5  8d 3b 05  sta $053b     Setzt neue Zeichenfarbe
. dcf8  68        pla           Holt Akku vom Stack
. dcf9  60        rts           ENDE

. dcfa  :
. dcfa  *** CURSOR RIGHT ***
. dcfa  :
. dcfa  20 bf df  jsr $dfbf     Cursor nach rechts
. dcfd  b0 04     bcs $dd03     Rand überschritten ?, dann neue Zeile
. dcff  60        rts           ENDE

. dd00  :
. dd00  *** CURSOR DOWN ***
. dd00  :
. dd00  20 21 da  jsr $da21     Zeilennummer erhöhen und ggf. Scroll auf
. dd03  20 39 df  jsr $df39     (GETBIT) Zeilentabelle
. dd06  b0 03     bcs $dd0b     Bit gesetzt ?, dann $dd0b
. dd08  38        sec           Setzt Flag
. dd09  66 c4     ror $c4       ... für 'ganze Zeile'
. dd0b  18        clc
. dd0c  60        rts           ENDE

. dd0d  :
```

```
. dd0d  *** CURSOR UP ***
. dd0d  :
. dd0d  ae e6 07 ldx $07e6    Holt 1. Bildschirmzeile
. dd10  e4 cd    cpx $cd      Cursor bereits in 1. Zeile ?
. dd12  b0 f8    bcs $dd0c    Ja, dann fertig
. dd14  20 03 dd jsr $dd03    (GETBIT) Zeilentabelle (Cursor down)
. dd17  c6 cd    dec $cd      Erniedrigt Zeilennummer um 1
. dd19  4c a8 d8 jmp $d8a8    Setzt Cursorzeiger

. dd1c  :
. dd1c  *** CURSOR LEFT ***
. dd1c  :
. dd1c  20 d4 df jsr $dfd4    Schiebt Cursor um 1 nach links
. dd1f  b0 eb    bcs $dd0c    War Cursor in HOME-Position ?, dann fertig
. dd21  d0 e8    bne $dd0b    Ist Cursor nicht am rechten Rand ?, dann $dd0b
. dd23  e6 cd    inc $cd      Erhöht Zeilennummer um 1
. dd25  d0 ed    bne $dd14    Cursor up

. dd27  :
. dd27  *** TEXT-MODUS ***
. dd27  :
. dd27  ad 13 ff lda $ff13    Holt Basisadresse des Zeichengenerators (TED)
. dd2a  09 04    ora #$04     Bit 2 setzen
. dd2c  d0 15    bne $dd43    Schreibt TED-Register

. dd2e  :
. dd2e  *** SHIFT - C= - SPERRE ***
. dd2e  :
. dd2e  a9 80    lda #$80     Holt FLAG für SHIFT - C= - Sperre
. dd30  0d 47 05 ora $0547    Setzt Flag
. dd33  30 05    bmi $dd3a    Byte zurückschreiben

. dd35  :
. dd35  *** SHIFT - C= - FREIGABE ***
. dd35  :
. dd35  a9 7f    lda #$7f     Holt FLAG für SHIFT - C= - Freigabe
. dd37  2d 47 05 and $0547    Setzt Flag
. dd3a  8d 47 05 sta $0547    Schreibt Byte zurück
. dd3d  60       rts          ENDE

. dd3e  :
. dd3e  *** GRAFIK-MODUS ***
. dd3e  :
. dd3e  ad 13 ff lda $ff13    Holt Basisadresse des Zeichengenerators (TED)
. dd41  29 fb    and #$fb     Löscht Bit 2
. dd43  8d 13 ff sta $ff13    Schreibt TED-Register zurück
. dd46  60       rts          ENDE

. dd47  :
. dd47  *** SHIFT-CODES BEARBEITEN ***
. dd47  :
. dd47  29 7f    and #$7f     Löscht Bit 7
```

```
. dd49  c9 7f     cmp #$7f      Bit Ø Bit 6 gesetzt ?
. dd4b  dØ Ø2     bne $dd4f     Nein, dann $dd4f
. dd4d  a9 5e     lda #$5e      Durch Code 94 ersetzen
. dd4f  c9 2Ø     cmp #$2Ø      Steuercode ?
. dd51  9Ø Ø3     bcc $dd56     Ja, dann $dd56
. dd53  4c d9 d9  jmp $d9d9     Gibt Zeichen aus & Abschluß

. dd56  a6 cb     ldx $cb       Anführungszeichenmodus aktiv ?
. dd58  fØ Ø5     beq $dd5f     Nein, dann $dd5f
. dd5a  Ø9 4Ø     ora #$4Ø      Setzt Bit 6
. dd5c  4c df d9  jmp $d9df     Gibt Zeichen reverse aus & Abschluß

. dd5f  c9 14     cmp #$14      'INSERT'-Code ?
. dd61  fØ 6b     beq $ddce     Ja, dann $ddce
. dd63  a6 cf     ldx $cf       INSERT-Modus aktiv ?
. dd65  dØ f3     bne $dd5a     Ja, dann $dd5a
. dd67  c9 11     cmp #$11      'Cursor up'-Code ?
. dd69  fØ a2     beq $ddØd     Ja, dann $ddØd
. dd6b  c9 12     cmp #$12      'Reverse off'-Code ?
. dd6d  dØ Ø4     bne $dd73     Nein, dann $dd73
. dd6f  a9 ØØ     lda #$ØØ      Initialisiert
. dd71  85 c2     sta $c2       ... Reverse-Flag
. dd73  c9 1d     cmp #$1d      'Cursor left'-Code ?
. dd75  fØ a5     beq $dd1c     Ja, dann $dd1c
. dd77  c9 13     cmp #$13      'CLEAR'-Code ?
. dd79  dØ Ø3     bne $dd7e     Nein, dann $dd7e
. dd7b  4c 8b d8  jmp $d88b     Löscht Bildschirm (CLEAR SCREEN)

. dd7e  c9 Ø2     cmp #$Ø2      'Flash on'-Code ?
. dd8Ø  dØ Ø5     bne $dd87     Nein, dann $dd87
. dd82  a9 8Ø     lda #$8Ø      Setzt
. dd84  8d 3c Ø5  sta $Ø53c     ... Blink-Flag
. dd87  c9 Ø4     cmp #$Ø4      'Flash off'-Code ?
. dd89  dØ Ø5     bne $dd9Ø     Nein, dann $dd9Ø
. dd8b  a9 ØØ     lda #$ØØ      Initialisiert
. dd8d  8d 3c Ø5  sta $Ø53c     ... Blink-Flag
. dd9Ø  c9 Øe     cmp #$Øe      Grafik-Modus ?
. dd92  fØ aa     beq $dd3e     Ja, dann $dd3e
. dd94  Ø9 8Ø     ora #$8Ø      Setzt Bit 7
. dd96  4c e6 dc  jmp $dce6     Gibt ggf. Farbcode aus

. dd99  :
. dd99  *** DELETE ***
. dd99  :
. dd99  2Ø 1c dd  jsr $dd1c     Cursor left
. dd9c  2Ø f6 df  jsr $dff6     Speichert Cursor-Position
. dd9f  bØ 1Ø     bcs $ddb1     Cursor in HOME-Position ?, dann $ddb1
. dda1  cc e8 Ø7  cpy $Ø7e8     Cursor am rechten Rand ?
. dda4  9Ø 16     bcc $ddbc     Nein, dann $ddbc
. dda6  a6 cd     ldx $cd       Holt Cursorzeile
. dda8  e8        inx           Verringert Cursorzeile um 1
. dda9  2Ø 3b df  jsr $df3b     (GETBIT) Zeilentabelle
```

```
. ddac  bØ Øe    bcs $ddbc    Bit gesetzt ?, dann $ddbc
. ddae  2Ø ff df jsr $dfff    Gibt Leerzeichen aus
. ddb1  a5 cc    lda $cc      Stellt
. ddb3  85 ca    sta $ca      ... Cursor-Position
. ddb5  a5 fe    lda $fe      ... wieder
. ddb7  85 cd    sta $cd      ... her
. ddb9  4c a8 d8 jmp $d8a8    Setzt Cursorzeiger

. ddbc  2Ø bf df jsr $dfbf    Cursor nach rechts
. ddbf  2Ø 2f df jsr $df2f    Holt Zeichen vom Bildschirm
. ddc2  2Ø d4 df jsr $dfd4    Cursor nach links
. ddc5  2Ø 11 eØ jsr $eØ11    Schreibt Zeichen auf Bildschirm
. ddc8  2Ø bf df jsr $dfbf    Cursor nach rechts
. ddcb  4c a1 dd jmp $dda1    Zurück zum Anfang

. ddce  :
. ddce  *** INSERT ***
. ddce  :
. ddce  2Ø f6 df jsr $dff6    Rettet Cursorposition
. ddd1  2Ø 95 df jsr $df95    Setzt Cursor an Zeilenende
. ddd4  e4 fe    cpx $fe      Insert
. ddd6  dØ Ø2    bne $ddda    ... vor
. ddd8  c4 cc    cpy $cc      ... Zeilenende ?
. ddda  9Ø 21    bcc $ddfd    Nein, dann $ddfd
. dddc  2Ø f8 d9 jsr $d9f8    Scroll ab, bei Zeilenende
. dddf  bØ 22    bcs $deØ3    Scrolling gesperrt ?, dann $deØ3
. dde1  2Ø d4 df jsr $dfd4    Cursor nach links
. dde4  2Ø 2f df jsr $df2f    Holt Zeichen vom Bildschirm
. dde7  2Ø bf df jsr $dfbf    Cursor nach rechts
. ddea  2Ø 11 eØ jsr $eØ11    Schreibt Zeichen auf Bildschirm
. dded  2Ø d4 df jsr $dfd4    Cursor nach links
. ddfØ  a6 cd    ldx $cd      Ausgangsposition
. ddf2  e4 fe    cpx $fe      erreicht ?...
. ddf4  dØ eb    bne $dde1
. ddf6  c4 cc    cpy $cc
. ddf8  dØ e7    bne $dde1    Nein, dann $dde1
. ddfa  2Ø ff df jsr $dfff    Schreibt Leerzeichen
. ddfd  e6 cf    inc $cf      Erhöht Insert-Zähler um 1
. ddff  dØ Ø2    bne $deØ3    Zähler größer $ff ?
. deØ1  c6 cf    dec $cf      Zähler maximal $ff
. deØ3  4c b1 dd jmp $ddb1    Stellt Cursorposition wieder her

. deØ6  :
. deØ6  *** ESC-SEQUENZ ***
. deØ6  :
. deØ6  29 7f    and #$7f     Löscht Bit 7
. deØ8  38       sec          Verschiebt Code
. deØ9  e9 41    sbc #$41     ... in den Bereich ($ØØ-$16)
. deØb  c9 17    cmp #$17     Bereich überschritten ?
. deØd  bØ Øa    bcs $de19    Ja, dann fertig
. deØf  Øa       asl          Erzeugt Offset
. de1Ø  aa       tax          ... in ESC-Tabelle
```

```
. de11  bd 1b de lda $de1b,x  Holt
. de14  48       pha          ... Routinenadresse
. de15  bd 1a de lda $de1a,x  ... und legt diese
. de18  48       pha          ... auf Stack
. de19  60       rts          Ruft Routine auf

. de1a  :
. de1a  *** ESC-TABELLE ***
. de1a  :
. de1a  28 df  $df29          ESC-A  Auto-Insert einschalten
. de1c  5f de  $de60          ESC-B  WINDOW-Untergrenze
. de1e  25 df  $df26          ESC-C  Auto-Insert ausschalten
. de20  9f de  $de9f          ESC-D  Zeile löschen
. de22  18 de  $de19          ESC-E  RTS
. de24  18 de  $de19          ESC-F  RTS
. de26  18 de  $de19          ESC-G  RTS
. de28  18 de  $de19          ESC-H  RTS
. de2a  8a de  $de8b          ESC-I  Zeile einfügen
. de2c  81 df  $df82          ESC-J  Setzt Cursor an Zeilenanfang
. de2e  94 df  $df95          ESC-K  Setzt Cursor ans Zeilenende
. de30  1c df  $df1d          ESC-L  Scrolling freigeben
. de32  1f df  $df20          ESC-M  Scrolling sperren
. de34  87 d8  $d888          ESC-N  Window löschen
. de36  9a dc  $dc9b          ESC-O  Initialisiert Flags
. de38  e0 de  $dee1          ESC-P  Löscht Zeile links vom Cursor
. de3a  ca de  $decb          ESC-Q  Löscht Zeile rechts vom Cursor
. de3c  47 de  $de48          ESC-R  Generiert Rand um Bildschirmfenster
. de3e  18 de  $de19          ESC-S  RTS
. de40  5d de  $de5e          ESC-T  WINDOW-Obergrenze
. de42  18 de  $de19          ESC-U  RTS
. de44  f5 de  $def6          ESC-V  Scroll auf
. de46  03 df  $df03          ESC-W  Scroll ab

. de48  :
. de48  *** ESC-R  BILDSCHIRMFENSTER MIT RAND ***
. de48  :
. de48  20 70 de jsr $de70    Stellt gesamten Bildschirm zur Verfügung
. de4b  20 8b d8 jsr $d88b    Löscht Bildschirm
. de4e  a9 01    lda #$01     Setzt
. de50  aa       tax          ... Window-Oberbegrenzung
. de51  20 7a de jsr $de7a    ... auf Zeile 2, Spalte 2
. de54  a9 17    lda #$17     Setzt
. de56  a2 26    ldx #$26     ... Window-Untergrenze
. de58  20 67 de jsr $de67    ... auf Zeile 24, Spalte 39
. de5b  4c 9a d8 jmp $d89a    Cursor in Home-Position

. de5e  :
. de5e  *** ESC-T  WINDOW-OBERGRENZE ***
. de5e  :
. de5e  18       clc          Flag für Obergrenze

. de5f  24       .byte $24
```

```
. de6Ø  :
. de6Ø  *** ESC-B  WINDOW-UNTERGRENZE ***
. de6Ø  :
. de6Ø  38       sec          Flag für Untergrenze
. de61  a6 ca    ldx $ca      Holt Cursorspalte
. de63  a5 cd    lda $cd      Holt Cursorzeile
. de65  9Ø 13    bcc $de7a    Obergrenze schreiben
. de67  8d e5 Ø7 sta $Ø7e5    Schreibt unteren Rand
. de6a  8e e8 Ø7 stx $Ø7e8    Schreibt rechten Rand
. de6d  4c 8Ø de jmp $de8Ø    Löscht Zeilentabelle

. de7Ø  :
. de7Ø  *** VOLLEN BILDSCHIRM HERSTELLEN ***
. de7Ø  :
. de7Ø  a9 18    lda #$18     Holt maximale
. de72  a2 27    ldx #$27     ... Bildschirmgröße (25*4Ø)
. de74  2Ø 67 de jsr $de67    Schreibt untere Bildschirmgrenze
. de77  a9 ØØ    lda #$ØØ     Holt kleinste Zeilen-
. de79  aa       tax          ... und Spaltennummer
. de7a  8d e6 Ø7 sta $Ø7e6    Schreibt oberen Rand
. de7d  8e e7 Ø7 stx $Ø7e7    Schreibt linken Rand
. de8Ø  a9 ØØ    lda #$ØØ     Löscht
. de82  a2 Ø4    ldx #$Ø4     ... Zeilentabelle...
. de84  9d ed Ø7 sta $Ø7ed,x
. de87  ca       dex
. de88  dØ fa    bne $de84
. de8a  6Ø       rts          ENDE

. de8b  :
. de8b  *** ESC-I  ZEILE EINFÜGEN ***
. de8b  :
. de8b  2Ø 5e da jsr $da5e    Scroll ab
. de8e  2Ø a1 d8 jsr $d8a1    Setzt Cursorzeiger
. de91  e8       inx          Erhöht Zeilenzeiger
. de92  2Ø 3b df jsr $df3b    (GETBIT) Zeilentabelle
. de95  Ø8       php
. de96  2Ø 46 df jsr $df46    (PUTBIT) Zeilentabelle
. de99  28       plp
. de9a  bØ Ø3    bcs $de9f    War Zeilenbit gesetzt ?, dann $de9f
. de9c  38       sec          Setzt Flag
. de9d  66 c4    ror $c4      ... für 'Ganze Zeile'
. de9f  6Ø       rts          ENDE

. deaØ  :
. deaØ  *** ESC-D  ZEILE LÖSCHEN ***
. deaØ  :
. deaØ  2Ø 87 df jsr $df87    Setzt Cursor an Zeilenanfang
. dea3  ad e6 Ø7 lda $Ø7e6    Holt 1. Zeile
. dea6  48       pha          Zeile auf Stack
. dea7  a5 cd    lda $cd      Holt Cursorzeile
. dea9  8d e6 Ø7 sta $Ø7e6    Schreibt 1. Zeile
```

```
. deac  ad ec Ø7 lda $Ø7ec    Holt Scroll-Flag
. deaf  48       pha          Flag auf Stack
. debØ  a9 8Ø    lda #$8Ø     Scroll
. deb2  8d ec Ø7 sta $Ø7ec    ... einschalten
. deb5  2Ø 9e da jsr $da9e    Scroll auf
. deb8  68       pla          Scroll-Flag
. deb9  8d ec Ø7 sta $Ø7ec    ... wiederherstellen
. debc  ad e6 Ø7 lda $Ø7e6    Gelöschte Zeile
. debf  85 cd    sta $cd      ... gleich Cursorzeile
. dec1  68       pla          Stellt 1. Zeile
. dec2  8d e6 Ø7 sta $Ø7e6    ... wieder her
. dec5  38       sec          Setzt Flag für
. dec6  66 c4    ror $c4      ... 'Ganze Zeile'
. dec8  4c a1 d8 jmp $d8a1    Setzt Cursor-Zeiger

. decb  :
. decb  *** ESC-Q  ZEILE RECHTS VOM CURSOR LÖSCHEN ***
. decb  :
. decb  2Ø f6 df jsr $dff6    Rettet Cursorposition
. dece  2Ø fd da jsr $dafd    Löscht Zeile ab Cursor
. ded1  e6 cd    inc $cd      Erhöht Cursorzeile um 1
. ded3  2Ø a8 d8 jsr $d8a8    Setzt Cursorzeiger
. ded6  ac e7 Ø7 ldy $Ø7e7    Holt linken Rand
. ded9  2Ø 39 df jsr $df39    (GETBIT) Zeilentabelle
. dedc  bØ fØ    bcs $dece    Bit gesetzt ?, dann $dece
. dede  4c b1 dd jmp $ddb1    Stellt Cursorposition wieder her

. dee1  :
. dee1  *** ESC-P  ZEILE LINKS VOM CURSOR LÖSCHEN ***
. dee1  :
. dee1  2Ø f6 df jsr $dff6    Rettet Cursorposition
. dee4  2Ø ff df jsr $dfff    Gibt Leerzeichen aus
. dee7  cc e7 Ø7 cpy $Ø7e7    Linker Rand erreicht ?
. deea  dØ Ø5    bne $def1    Nein, dann $def1
. deec  2Ø 39 df jsr $df39    (GETBIT) Zeilentabelle
. deef  9Ø ed    bcc $dede    Bit nicht gesetzt ?, dann $dede
. def1  2Ø d4 df jsr $dfd4    Cursor links
. def4  9Ø ee    bcc $dee4    Springt immer nach $dee4

. def6  :
. def6  *** ESC-V  SCROLL AUF ***
. def6  :
. def6  2Ø f6 df jsr $dff6    Rettet Cursorposition
. def9  8a       txa          Cursorzeile auf
. defa  48       pha          ... Stack retten
. defb  2Ø 89 da jsr $da89    Scroll auf
. defe  68       pla          Cursorzeile
. deff  85 fe    sta $fe      ... wiederherstellen
. dfØ1  4c de de jmp $dede    Stellt Cursorposition wieder her

. dfØ4  :
. dfØ4  *** ESC-W  SCROLL AB ***
```

```
. df04  :
. df04  20 f6 df jsr $dff6     Rettet Cursorposition
. df07  20 39 df jsr $df39     (GETBIT) Zeilentabelle
. df0a  b0 03    bcs $df0f     Bit gesetzt ?, dann $df0f
. df0c  38       sec           Setzt Flag
. df0d  66 c4    ror $c4       ... für 'Ganze Zeile'
. df0f  ad e6 07 lda $07e6     Setzt Cursorzeile gleich
. df12  85 cd    sta $cd       ... oberen Rand
. df14  20 5e da jsr $da5e     Scroll ab
. df17  20 4a df jsr $df4a     (CLRBIT) Zeilentabelle
. df1a  4c de de jmp $dede     Stellt Cursorposition wieder her

. df1d  :
. df1d  *** ESC-L  SCROLLING FREIGEBEN ***
. df1d  :
. df1d  a9 00    lda #$00      Flag für Freigabe

. df1f  2c       .byte $2c

. df20  :
. df20  *** ESC-M  SCROLLING SPERREN ***
. df20  :
. df20  a9 80    lda #$80      Flag für Sperre
. df22  8d e9 07 sta $07e9     Schreibt Scroll-Flag
. df25  60       rts           ENDE

. df26  :
. df26  *** ESC-C  AUTO-INSERT AUS ***
. df26  :
. df26  a9 00    lda #$00      Flag für AUS

. df28  2c       .byte $2c

. df29  :
. df29  *** ESC-A  AUTO-INSERT EIN ***
. df29  :
. df29  a9 ff    lda #$ff      Flag für EIN
. df2b  8d ea 07 sta $07ea     Schreibt Insert-Flag
. df2e  60       rts           ENDE

. df2f  :
. df2f  *** ZEICHEN VOM BILDSCHIRM HOLEN ***
. df2f  :
. df2f  a4 ca    ldy $ca       Holt Cursorzeiger
. df31  b1 ea    lda ($ea),y   Holt Farbe an Stelle des Cursors
. df33  8d ed 07 sta $07ed     Speichert Farbcode
. df36  b1 c8    lda ($c8),y   Holt Zeichen an Stelle des Cursors
. df38  60       rts           ENDE

. df39  :
. df39  *** ZEILENTABELLE ***
. df39  :
```

```
.       (GETBIT)

. df39  a6 cd    ldx $cd      Holt Cursorzeile
. df3b  2Ø 66 df jsr $df66    Erzeugt Bitposition
. df3e  3d ee Ø7 and $Ø7ee,x  Maskiert mit Bit-Tabelle
. df41  c9 Ø1    cmp #$Ø1     Gesetztes Bit setzt Carry
. df43  4c 55 df jmp $df55    Holt Byte-Offset in Tabelle und ENDE

        (PUTBIT)

. df46  a6 cd    ldx $cd      Holt Cursorzeile
. df48  bØ Øf    bcs $df59    Carry=1 ?, dann SETBIT

        (CLRBIT)

. df4a  2Ø 66 df jsr $df66    Erzeugt Bitposition
. df4d  49 ff    eor #$ff     Invertiert alle Bits
. df4f  3d ee Ø7 and $Ø7ee,x  Maskiert mit Bit-Tabelle
. df52  9d ee Ø7 sta $Ø7ee,x  In Bit-Tabelle zurückschreiben
. df55  ae e9 Ø2 ldx $Ø2e9    Holt Offset in Tabelle
. df58  6Ø       rts          ENDE

        (SETBIT)

. df59  2c e9 Ø7 bit $Ø7e9    Gibt Scroll frei
. df5c  7Ø dd    bvs $df3b    Bit 6 gesetzt ?, dann $df3b
. df5e  2Ø 66 df jsr $df66    Erzeugt Bitposition
. df61  1d ee Ø7 ora $Ø7ee,x  ODER-Verknüpfung mit Bit-Tabelle
. df64  dØ ec    bne $df52    Springt immer nach $df52

        ERZEUGT BITPOSITION

. df66  8e e9 Ø2 stx $Ø2e9    Schreibt Cursorzeile
. df69  8a       txa          Isoliert
. df6a  29 Ø7    and #$Ø7     ... Bit Ø
. df6c  aa       tax          ... bis Bit 2
. df6d  bd 7a df lda $df7a,x  Holt Bitwert
. df7Ø  48       pha          Bitwert auf Stack legen
. df71  ad e9 Ø2 lda $Ø2e9    Holt Cursorzeile
. df74  4a       lsr          Dividiert
. df75  4a       lsr          ... Cursorzeile
. df76  4a       lsr          ... durch 8
. df77  aa       tax          Ergebnis gleich Offset in Tabelle
. df78  68       pla          Holt Bitwert vom Stack
. df79  6Ø       rts          ENDE

        BITWERT (BIT 7 - BIT Ø)

. df7a  8Ø 4Ø 2Ø 1Ø Ø4 Ø2 Ø1

. df82  :
```

```
. df82  *** ESC-J  SETZT CURSOR AN ZEILENANFANG ***
. df82  :
. df82  ac e7 Ø7 ldy $Ø7e7    Setzt Cursorspalte
. df85  84 ca    sty $ca      ... gleich linken Rand
. df87  2Ø 39 df jsr $df39    (GETBIT) Zeilentabelle
. df8a  9Ø Ø6    bcc $df92    Bit nicht gesetzt ?, dann $df92
. df8c  c6 cd    dec $cd      Erniedrigt Cursorzeile um 1
. df8e  1Ø f7    bpl $df87    Zeile war noch nicht 1. Zeile ?, dann $df87
. df9Ø  e6 cd    inc $cd      Cursorzeile gleich 1. Zeile
. df92  4c a8 d8 jmp $d8a8    Setzt Cursorzeiger

. df95  :
. df95  *** ESC-K  SETZT CURSOR ANS ZEILENENDE ***
. df95  :
. df95  e6 cd    inc $cd      Erhöht aktuelle Zeile um 1
. df97  2Ø 39 df jsr $df39    (GETBIT) Zeilentabelle
. df9a  bØ f9    bcs $df95    Bit gesetzt ?, dann $df95
. df9c  c6 cd    dec $cd      Erniedrigt Cursorzeile um 1
. df9e  2Ø a8 d8 jsr $d8a8    Setzt Cursorzeiger
. dfa1  ac e8 Ø7 ldy $Ø7e8    Setzt Cursorzeiger
. dfa4  84 ca    sty $ca      ... gleich rechten Rand
. dfa6  2Ø 2f df jsr $df2f    Holt Zeichen vom Bildschirm
. dfa9  c9 2Ø    cmp #$2Ø     Zeichen gleich Leerzeichen ?
. dfab  dØ Øf    bne $dfbc    Nein, dann $dfbc
. dfad  cc e7 Ø7 cpy $Ø7e7    Linker Rand erreicht ?
. dfbØ  dØ Ø5    bne $dfb7    Nein, dann $dfb7
. dfb2  2Ø 39 df jsr $df39    (GETBIT) Zeilentabelle
. dfb5  9Ø Ø5    bcc $dfbc    Bit nicht gesetzt ?, dann $dfbc
. dfb7  2Ø d4 df jsr $dfd4    Cursor links
. dfba  9Ø ea    bcc $dfa6    Cursor nicht in HOME-Position ?, dann $dfa6
. dfbc  84 c3    sty $c3      Speicher Position des letzten Zeichens
. dfbe  6Ø       rts          ENDE

. dfbf  :
. dfbf  *** CURSOR RECHTS ***
. dfbf  :
. dfbf  48       pha          Akku auf Stack retten
. dfcØ  a4 ca    ldy $ca      Holt Cursorspalte
. dfc2  cc e8 Ø7 cpy $Ø7e8    Cursor am rechten Rand ?
. dfc5  9Ø Ø8    bcc $dfcf    Nein, dann $dfcf
. dfc7  2Ø 21 da jsr $da21    Neue Zeile
. dfca  ac e7 Ø7 ldy $Ø7e7    Holt linken Rand
. dfcd  88       dey          Cursorspalte um 1 verringern
. dfce  38       sec
. dfcf  c8       iny          Cursorspalte um 1 erhöhen
. dfdØ  84 ca    sty $ca      Schreibt Cursorspalte
. dfd2  68       pla          Akku vom Stack holen
. dfd3  6Ø       rts          ENDE

. dfd4  :
. dfd4  *** CURSOR LINKS ***
. dfd4  :
```

```
. dfd4  a4 ca     ldy $ca       Holt Cursorspalte
. dfd6  88        dey           Cursorspalte um 1 verringern
. dfd7  30 05     bmi $dfde     Linker Rand überschritten ?, dann $dfde
. dfd9  cc e7 07  cpy $07e7     Linker Rand erreicht ?
. dfdc  b0 11     bcs $dfef     Nein, dann $dfef
. dfde  ac e6 07  ldy $07e6     Holt oberen Rand
. dfe1  c4 cd     cpy $cd       Rand gleich Cursorzeile ?
. dfe3  b0 10     bcs $dff5     Ja, dann fertig
. dfe5  c6 cd     dec $cd       Erniedrigt Cursorzeile
. dfe7  48        pha
. dfe8  20 a8 d8  jsr $d8a8     Setzt Cursorzeile
. dfeb  68        pla
. dfec  ac e8 07  ldy $07e8     Holt rechten Rand
. dfef  84 ca     sty $ca       Schreibt Cursorspalte
. dff1  cc e8 07  cpy $07e8     Vergleicht mit rechtem Rand
. dff4  18        clc
. dff5  60        rts           ENDE

. dff6  :
. dff6  *** CURSORPOSITION RETTEN ***
. dff6  :
. dff6  a4 ca     ldy $ca       Speichert
. dff8  84 cc     sty $cc       ... Cursorspalte
. dffa  a6 cd     ldx $cd       Speichert
. dffc  86 fe     stx $fe       ... Cursorzeile
. dffe  60        rts           ENDE

. dfff  :
. dfff  *** SCHREIBT ZEICHEN AUF BILDSCHIRM ***
. dfff  :
. dfff  a9 20     lda #$20      Zeichen gleich Leerzeichen

. e001  a4 ca     ldy $ca       Holt Cursorspalte
. e003  91 c8     sta ($c8),y   Schreibt Zeichen in Bildschirmzeile
. e005  20 b4 d8  jsr $d8b4     Setzt Zeiger im Farb-RAM
. e008  ad 3b 05  lda $053b     Holt aktuelle Farbe
. e00b  0d 3c 05  ora $053c     Schaltet ggf. Blinken ein
. e00e  91 ea     sta ($ea),y   Schreibt Farbcode in Farb-RAM
. e010  60        rts           ENDE

. e011  a4 ca     ldy $ca       Holt Cursorspalte
. e013  91 c8     sta ($c8),y   Schreibt Zeichen in Bildschirmzeile
. e015  20 b4 d8  jsr $d8b4     Setzt Zeiger im Farb-RAM
. e018  ad ed 07  lda $07ed     Holt Farbe
. e01b  91 ea     sta ($ea),y   Schreibt Farbcode in Farb-RAM
. e01d  60        rts           ENDE

. e01e  :
. e01e  *** TASTENCODE-TABELLE ***
. e01e  :
. e01e  26 e0     $e026         Tabelle 1: Taste allein
. e020  67 e0     $e067         Tabelle 2: SHIFT + Taste
```

```
. e022  ab e0     $e0ab         Tabelle 3: C= + Taste
. e024  e9 e0     $e0e9         Tabelle 4: CTRL + Taste

      TABELLE 1: Tastatur

. e026 14 0d 5c 8c 85 89 86 40
. e02e 33 57 41 34 5a 53 45 01
. e036 35 52 44 36 43 46 54 58
. e03e 37 59 47 38 42 48 55 56
. e046 39 49 4a 30 4d 4b 4f 4e
. e04e 11 50 4c 91 2e 3a 2d 2c
. e056 9d 2a 3b 1d 1b 3d 2b 2f
. e05e 31 13 04 32 20 02 51 03
. e066 ff

      TABELLE 2: SHIFT + Tastatur

. e067 94 8d a9 88 8a 87 8b ba
. e06f 23 d7 c1 24 da d3 c5 01
. e077 25 d2 c4 26 c3 c6 d4 d8
. e07f 27 d9 c7 28 c2 c8 d5 d6
. e087 29 c9 ca 5e cd cb cf ce
. e08f 11 d0 cc 91 3e 5b dd 3c
. e097 9d c0 5d 1d 1b 5f db 3f
. e09f 21 93 04 22 a0 02 d1 83
. e0a7 ff 94 8d a8

      TABELLE 3: C= + Tastatur

. e0ab 88 8a 87 8b a4 96 b3 b0
. e0b3 97 ad ae b1 01 98 b2 ac
. e0bb 99 bc bb a3 bd 9a b7 a5
. e0c3 9b bf b4 b8 be 29 a2 b5
. e0cb 30 a7 a1 b9 aa 11 af b6
. e0d3 91 3e 5b dc 3c 9d df 5d
. e0db 1d 1b de a6 3f 81 93 04
. e0e3 95 a0 02 ab 83 ff

      TABELLE 4: CTRL + Tastatur

. e0e9 ff ff 1c ff ff ff ff ff
. e0f1 1c 17 01 9f 1a 13 05 ff
. e0f9 9c 12 04 1e 03 06 14 18
. e101 1f 19 07 9e 02 08 15 16
. e109 12 09 0a 92 0d 0b 0f 0e
. e111 ff 10 0c ff 84 1b ff 82
. e119 ff ff 1d ff 1b 06 ff ff
. e121 90 ff ff 05 ff ff 11 ff
. e129 ff

. e13a  :
. e13a  *** RUN-TEXT ***
```

```
. e13a  :
. e13a 44 cc 22 2a            'dL"*'

. e13e Ød                     CR-Code

. e13f 52 55 4e               'run'

. e132 Ød                     CR-Code

. e133  :
. e133  *** FARB-TABELLEN ***
. e133  :

      CODES DER FARBTASTEN

. e133 9Ø Ø5 1c 9f 9c 1e 1f 9e
. e13b 81 95 96 97 98 99 9a 9b

      FARBCODES

. e143 ØØ 71 32 63 44 35 46 77
. e14b 48 29 5a 6b 5c 6d 2e 5f

. e153  :
. e153  *** TALK ***
. e153  :
. e153  Ø9 4Ø     ora #$4Ø     TALK-Bit

. e155  2c        .byte $2c

. e156  :
. e156  *** LISTEN ***
. e156  :
. e156  Ø9 2Ø     ora #$2Ø     LISTEN-Bit
. e158  48        pha          Geräteadresse auf Stack legen
. e159  24 94     bit $94      Byte im Puffer ?
. e15b  1Ø Øa     bpl $e167    Nein, dann $e167
. e15d  38        sec
. e15e  66 a6     ror $a6      Register für seriellen Bus
. e16Ø  2Ø 81 e1  jsr $e181    Byte ausgeben
. e163  46 94     lsr $94      Setzt Pufferflag zurück
. e165  46 a6     lsr $a6      Register für seriellen Bus
. e167  68        pla          Holt Geräteadresse mit TALK/LISTEN-Bit
. e168  85 95     sta $95      Schreibt Zeichen in Puffer für seriellen Bus
. e16a  78        sei
. e16b  2Ø c6 e2  jsr $e2c6    Gibt DATA Ø aus
. e16e  2Ø bf e2  jsr $e2bf    Gibt CLOCK 1 aus
. e171  a5 Ø1     lda $Ø1      Holt Ein-/Ausgabe-Register
. e173  Ø9 Ø4     ora #$Ø4     Löscht Bit 2
. e175  85 Ø1     sta $Ø1      Schreibt Ein-/Ausgabe-Register
. e177  78        sei
. e178  2Ø bf e2  jsr $e2bf    Gibt CLOCK 1 aus
```

```
. e17b  20 c6 e2 jsr $e2c6    Gibt DATA Ø aus
. e17e  20 dc e2 jsr $e2dc    Wartet 1 Millisekunde
. e181  :
. e181  *** BYTE AUSGEBEN ***
. e181  :
. e181  78       sei
. e182  20 c6 e2 jsr $e2c6    Gibt DATA Ø aus
. e185  20 d4 e2 jsr $e2d4    Data Eingabe
. e188  b0 5f    bcs $e1e9    Data=1 ?, 'DEVICE NOT PRESENT' in Status
. e18a  20 b8 e2 jsr $e2b8    Gibt CLOCK Ø aus
. e18d  24 a6    bit $a6
. e18f  10 0e    bpl $e19f
. e191  20 d4 e2 jsr $e2d4    Data Eingabe
. e194  90 fb    bcc $e191    Data = Ø ?, dann warten
. e196  a5 01    lda $01      Holt Ein-/Ausgabe-Register
. e198  c5 01    cmp $01      Register verändert ?
. e19a  d0 fa    bne $e196    Ja, dann warten
. e19c  0a       asl          Data 1 empfangen ?
. e19d  b0 f7    bcs $e196    Ja, dann warten
. e19f  20 d4 e2 jsr $e2d4    Data Eingabe
. e1a2  90 fb    bcc $e19f    Data Ø empfangen ?, dann warten
. e1a4  20 bf e2 jsr $e2bf    Gibt CLOCK 1 aus
. e1a7  a9 08    lda #$08     Setzt Bitzähler
. e1a9  85 aa    sta $aa      ... auf 8 Bit
. e1ab  20 d4 e2 jsr $e2d4    Data Eingabe
. e1ae  90 3e    bcc $e1ee    Data Ø empfangen ?, dann TIMEOUT
. e1b0  66 95    ror $95      Rotiert Datenbyte nach Carry
. e1b2  b0 05    bcs $e1b9    Bit gesetzt ?, dann $e1b9
. e1b4  20 cd e2 jsr $e2cd    Gibt Data 1 aus
. e1b7  d0 03    bne $e1bc    Springt immer nach $e1bc

. e1b9  20 c6 e2 jsr $e2c6    Gibt Data Ø aus
. e1bc  20 11 e3 jsr $e311    Wartet 20 Mikrosekunden
. e1bf  20 b8 e2 jsr $e2b8    Gibt Clock Ø aus
. e1c2  20 11 e3 jsr $e311    Wartet 20 Mikrosekunden
. e1c5  a5 01    lda $01      Gibt
. e1c7  29 fe    and #$fe     ... Data Ø
. e1c9  09 02    ora #$02     ... und Clock 1
. e1cb  85 01    sta $01      ... aus
. e1cd  c6 aa    dec $aa      Bitzähler um 1 Bit verringern
. e1cf  d0 da    bne $e1ab    Noch nicht Null ?, dann weitere Bits ausgeben
. e1d1  8a       txa          X-Reg. auf
. e1d2  48       pha          ... Stack retten
. e1d3  a2 78    ldx #$78     Zeitschleife (zirka 1 Millisekunde)
. e1d5  a5 01    lda $01      Wartet bis
. e1d7  c5 01    cmp $01      ... Ein-/Ausgabe-Port
. e1d9  d0 fa    bne $e1d5    ... stabil ist
. e1db  0a       asl          Data Ø empfangen ?
. e1dc  90 07    bcc $e1e5    Ja, dann fertig
. e1de  ca       dex          Wartezeit verringern
. e1df  d0 f4    bne $e1d5    Noch nicht Null ?, dann warten
. e1e1  68       pla          Holt X-Reg.
```

```
. e1e2  aa       tax          ... vom Stack
. e1e3  b0 09    bcs $e1ee    TIMEOUT

. e1e5  68       pla          Holt X-Reg.
. e1e6  aa       tax          ... vom Stack
. e1e7  58       cli          Interrupt zulassen
. e1e8  60       rts          ENDE

. e1e9  a9 80    lda #$80     Code für 'DEVICE NOT PRESENT'
. e1eb  4c f0 e1 jmp $e1f0    Abschluß

. e1ee  a9 03    lda #$03     Code für 'TIMEOUT'
. e1f0  20 1e f4 jsr $f41e    Code in Status bringen
. e1f3  58       cli          Interrupt zulassen
. e1f4  18       clc          Springt immer
. e1f5  90 4b    bcc $e242    ... nach $e242

. e1f7  :
. e1f7  *** SECOND ***
. e1f7  :
. e1f7  85 95    sta $95      Schreibt Sekundäradresse in Puffer für Ausgabe
. e1f9  20 77 e1 jsr $e177    Gibt Byte aus
. e1fc  a5 01    lda $01      Gibt
. e1fe  29 fb    and #$fb     ... 'ATN'
. e200  85 01    sta $01      ... aus
. e202  60       rts          ENDE

. e203  :
. e203  *** TKSA ***
. e203  :
. e203  85 95    sta $95      Schreibt Sekundäradresse in Puffer für Ausgabe
. e205  20 77 e1 jsr $e177    Gibt Byte aus
. e208  24 90    bit $90      Holt Status
. e20a  30 36    bmi $e242    'DEVICE NOT PRESENT' ?, dann $e242
. e20c  78       sei          Kein Interrupt zugelassen
. e20d  20 cd e2 jsr $e2cd    Gibt Data 1 aus
. e210  20 fc e1 jsr $e1fc    Gibt 'ATN' aus
. e213  20 b8 e2 jsr $e2b8    Gibt Clock 0 aus
. e216  24 01    bit $01      Data 1 ?
. e218  70 fc    bvs $e216    Ja, dann warten
. e21a  58       cli          Interrupt zulassen
. e21b  60       rts          ENDE

. e21c  ff       ???

. e21d  :
. e21d  *** CIOUT ***
. e21d  :
. e21d  24 94    bit $94      Byte im Puffer ?
. e21f  30 05    bmi $e226    Ja, dann $e226
. e221  38       sec          Setzt
. e222  66 94    ror $94      ... Puffer-Flag
```

```
. e224  dØ Ø5     bne $e22b    Springt immer nach $e22b

. e226  48        pha          Datenbyte auf Stack retten
. e227  2Ø 81 e1  jsr $e181    Gibt Datenbyte aus Puffer aus
. e22a  68        pla          Holt Datenbyte vom Stack
. e22b  85 95     sta $95      Schreibt Datenbyte in Puffer
. e22d  18        clc
. e22e  6Ø        rts          ENDE

. e22f  :
. e22f  *** UNTALK ***
. e22f  :
. e22f  78        sei          Kein Interrupt zugelassen
. e23Ø  2Ø bf e2  jsr $e2bf    Gibt Clock 1 aus
. e233  a5 Ø1     lda $Ø1      Gibt
. e235  Ø9 Ø4     ora #$Ø4     ... 'ATN'
. e237  85 Ø1     sta $Ø1      ... aus
. e239  a9 5f     lda #$5f     Holt Code für 'UNTALK'
. e23b  dØ Ø2     bne $e23f    Springt nach $e23f

. e23d  :
. e23d  *** UNLISTEN ***
. e23d  :
. e23d  a9 3f     lda #$3f     Holt Code für 'UNLISTEN'
. e23f  2Ø 58 e1  jsr $e158    LISTEN/TALK
. e242  2Ø fc e1  jsr $e1fc    Gibt 'ATN' aus
. e245  8a        txa          X-Reg. nach Akku
. e246  a2 14     ldx #$14     Zeitschleife (zirka 5Ø Mikrosekunden)
. e248  ca        dex          Zähler um 1 erniedrigen
. e249  dØ fd     bne $e248    Noch nicht Null ?, dann warten
. e24b  aa        tax          Akku nach X-Reg.
. e24c  2Ø b8 e2  jsr $e2b8    Gibt Clock Ø aus
. e24f  4c c6 e2  jmp $e2c6    Gibt Data Ø aus

. e252  :
. e252  *** ACPTR: BYTE EMPFANGEN ***
. e252  :
. e252  78        sei          Kein Interrupt zugelassen
. e253  a9 ØØ     lda #$ØØ     Initialisiert
. e255  85 aa     sta $aa      ... Ende-Flag
. e257  2Ø b8 e2  jsr $e2b8    Gibt Clock Ø aus
. e25a  8a        txa          X-Reg. auf
. e25b  48        pha          ... Stack retten
. e25c  2Ø d4 e2  jsr $e2d4    Data Eingabe
. e25f  1Ø fb     bpl $e25c    Data Ø empfangen ?, dann warten
. e261  a2 2Ø     ldx #$2Ø     Zeitschleife (zirka Ø.25 Millisekunden)
. e263  2Ø c6 e2  jsr $e2c6    Gibt Data Ø aus
. e266  a5 Ø1     lda $Ø1      Wartet Stabilisierung
. e268  c5 Ø1     cmp $Ø1      ... am Ein-/Ausgabe-Port
. e26a  dØ fa     bne $e266    ... ab
. e26c  Øa        asl          Data Ø empfangen ?
. e26d  1Ø 1f     bpl $e28e    Ja, dann $e28e
```

```
. e26f  ca       dex          Vermindert Wartezeit
. e270  d0 f4    bne $e266    Zähler noch nicht Null ?, dann warten
. e272  a5 aa    lda $aa      Ende-Flag gesetzt ?
. e274  f0 07    beq $e27d    Nein, dann $e27d
. e276  68       pla          X-Reg. vom
. e277  aa       tax          ... Stack holen
. e278  a9 02    lda #$02     Code für 'READ TIMEOUT'
. e27a  4c f0 e1 jmp $e1f0    Code in Status bringen

. e27d  20 cd e2 jsr $e2cd    Gibt Data 1 aus
. e280  a2 40    ldx #$40     Zeitschleife (zirka 0.16 Millisekunden)
. e282  ca       dex          Vermindert Wartezeit
. e283  d0 fd    bne $e282    Zähler noch nicht Null ?, dann warten
. e285  a9 40    lda #$40     Holt EOI-Bit
. e287  20 1e f4 jsr $f41e    Schreibt Bit in Status
. e28a  e6 aa    inc $aa      Setzt Ende-Flag
. e28c  d0 d3    bne $e261    Springt immer nach $e261

. e28e  a2 08    ldx #$08     Holt Bitzähler (8 Bit)
. e290  a5 01    lda $01      Holt Byte vom Ein-/Ausgabe-Port
. e292  0a       asl          Clock 1 empfangen ?
. e293  10 fb    bpl $e290    Nein, dann warten
. e295  66 a8    ror $a8      Rotiert Byte im Eingaberegister
. e297  a5 01    lda $01      Ein-/Ausgabe-Port
. e299  c5 01    cmp $01      ... stabil ?
. e29b  d0 fa    bne $e297    Nein, dann warten
. e29d  0a       asl          Clock 1 empfangen ?
. e29e  30 f7    bmi $e297    Ja, dann ebenfalls warten
. e2a0  ca       dex          Erniedrigt Bitzähler
. e2a1  d0 ed    bne $e290    Noch nicht Null ?, dann $e290
. e2a3  86 aa    stx $aa      Löscht Ende-Flag
. e2a5  68       pla          Holt X-Reg.
. e2a6  aa       tax          ... vom Stack
. e2a7  20 cd e2 jsr $e2cd    Gibt Data 1 aus
. e2aa  a9 40    lda #$40     ???
. e2ac  24 90    bit $90      Status
. e2ae  50 03    bvc $e2b3    Kein EOI ?, dann $e2b3
. e2b0  20 45 e2 jsr $e245    Gibt nach 50 Zyklen Clock 0 & Data 0 aus
. e2b3  a5 a8    lda $a8      Holt empfangenes Datenbyte
. e2b5  58       cli          Interrupt zulassen
. e2b6  18       clc
. e2b7  60       rts          ENDE

. e2b8  :
. e2b8  *** BUS-AUSGABE ***
. e2b8  :
. e2b8  a5 01    lda $01      Gibt
. e2ba  29 fd    and #$fd     ... Clock 0
. e2bc  85 01    sta $01      ... aus
. e2be  60       rts          ENDE

. e2bf  a5 01    lda $01      Gibt
```

```
. e2c1  Ø9 Ø2     ora #$Ø2      ... Clock 1
. e2c3  85 Ø1     sta $Ø1       ... aus
. e2c5  6Ø        rts           ENDE

. e2c6  a5 Ø1     lda $Ø1       Gibt
. e2c8  29 fe     and #$fe      ... Data Ø
. e2ca  85 Ø1     sta $Ø1       ... aus
. e2cc  6Ø        rts           ENDE

. e2cd  a5 Ø1     lda $Ø1       Gibt
. e2cf  Ø9 Ø1     ora #$Ø1      ... Data 1
. e2d1  85 Ø1     sta $Ø1       ... aus
. e2d3  6Ø        rts           ENDE
. e2d4  :
. e2d4  *** BUS-EMPFANG ***
. e2d4  :
. e2d4  a5 Ø1     lda $Ø1       Ist Ein-/Ausgabe-Port
. e2d6  c5 Ø1     cmp $Ø1       ... stabil ?
. e2d8  dØ fa     bne $e2d4     Nein, dann warten
. e2da  Øa        asl           Carry := 1 (für Data) / Bit 7 (für Clock)
. e2db  6Ø        rts           ENDE

. e2dc  :
. e2dc  *** ZEITSCHLEIFEN ***
. e2dc  :
. e2dc  2Ø f8 e2  jsr $e2f8     Setzt Timer 2 (1 Millisekunde)
. e2df  a9 1Ø     lda #$1Ø      Unterlauf von
. e2e1  2c Ø9 ff  bit $ffØ9     ... Timer 2 ?
. e2e4  fØ fb     beq $e2e1     Nein, dann warten
. e2e6  8d Ø9 ff  sta $ffØ9     Schreibt Interrupt-Request-Register
. e2e9  6Ø        rts           ENDE

. e2ea  2Ø fc e2  jsr $e2fc     Setzt Timer 2 (16 Millisekunde)
. e2ed  a9 1Ø     lda #$1Ø      Unterlauf von
. e2ef  2c Ø9 ff  bit $ffØ9     ... Timer 2 ?
. e2f2  fØ fb     beq $e2ef     Nein, dann warten
. e2f4  8d Ø9 ff  sta $ffØ9     Schreibt Interrupt-Request-Register
. e2f7  6Ø        rts           ENDE

. e2f8  a9 Ø4     lda #$Ø4      Holt Parameter für 1 ms (4 * 256 Zyklen)
. e2fa  dØ Ø2     bne $e2fe     Springt immer nach $e2fe

. e2fc  a9 4Ø     lda #$4Ø      Holt Parameter für 16 ms (64 * 256 Zyklen)
. e2fe  Ø8        php
. e2ff  48        pha           Wartezeit auf Stack legen
. e3ØØ  78        sei           Kein Interrupt zugelassen
. e3Ø1  a9 ØØ     lda #$ØØ      Setzt Timer (Low)
. e3Ø3  8d Ø2 ff  sta $ffØ2     ... auf Null
. e3Ø6  68        pla           Holt Wartezeit vom Stack
. e3Ø7  8d Ø3 ff  sta $ffØ3     Schreibt gewünschte
. e3Øa  a9 1Ø     lda #$1Ø      ... Wartezeit in Timer (High)
. e3Øc  8d Ø9 ff  sta $ffØ9     Startet Timer
```

```
. e30f  28       plp
. e310  60       rts          ENDE

. e311  8a       txa          Warteschleife für
. e312  a2 05    ldx #$05     ... zirka
. e314  ca       dex          ... 18 Mikrosekunden (5*5+12 Zyklen)
. e315  d0 fd    bne $e314    ...
. e317  aa       tax
. e318  60       rts          ENDE

. e319  :
. e319  *** PRÜFT PLAY-TASTE ***
. e319  :
. e319  38       sec          Einsprungadresse bei SAVE (Carry=1)

. e31a  24       .byte $24

. e31b  18       clc          Einsprungadresse bei LOAD/VERIFY (Carry=0)
. e31c  ad 10 fd lda $fd10    PLAY-Taste
. e31f  29 04    and #$04     ... gedrückt ?
. e321  f0 3f    beq $e362    Ja, dann $e362
. e323  08       php
. e324  20 d8 fb jsr $fbd8    Gibt Meldung aus

. e327  0d                    CR-Code
. e328  50 52 45 53 53        'press'
. e32d  20 50 4c 41 59 20     ' play '

. e323  00       .byte $00    Endmarke

. e334  28       plp
. e335  90 0d    bcc $e344    LOAD-Flag ?, dann $e344
. e337  20 d8 fb jsr $fbd8    Gibt Meldung aus

. e33a  26 20                 '& '
. e33c  52 45 43 4f 52 44 20  'record '
. e343  00       .byte $00    Endmarke

. e344  20 d8 fb jsr $fbd8    Gibt Meldung aus

. e347  4f 4e 20 54 41 50 41  'on tape'

. e34e  00       .byte $00    Endmarke

. e34f  20 cb fb jsr $fbcb    STOP-Taste gedrückt ?
. e352  b0 0f    bcs $e363    Ja, dann $e363
. e354  ad 10 fd lda $fd10    PLAY-Taste
. e357  29 04    and #$04     ... gedrückt ?
. e359  d0 f4    bne $e34f    Nein, dann warten
. e35b  20 d8 fb jsr $fbd8    Gibt Meldung aus

. e35e  0d                    CR-Code
```

```
. e35f  4f 4b                 'ok'
. e361  ØØ        .byte $ØØ   Endmarke

. e362  18        clc
. e363  6Ø        rts         ENDE

. e364  :
. e364  *** KASSETTENPORT EINSCHALTEN ***
. e364  :
. e364  78        sei         Kein Interrupt zugelassen
. e365  ad Ø6 ff lda $ffØ6    Schaltet
. e368  29 ef     and #$ef    ... Bildschirm
. e36a  8d Ø6 ff sta $ffØ6    ... aus
. e36d  ad Øa ff lda $ffØa    Sperrt Rasterinterrupt
. e37Ø  29 fd     and #$fd    ... und ermöglicht
. e372  Ø9 Ø8     ora #$Ø8    ... Interrupt bei
. e374  8d Øa ff sta $ffØa    ... Unterlauf von Timer 1
. e377  6Ø        rts         ENDE

. e378  :
. e378  *** KASSETTENPORT AUSSCHALTEN ***
. e378  :
. e378  78        sei         Kein Interrupt zugelassen
. e379  ad Ø6 ff lda $ffØ6    Schaltet
. e37c  Ø9 1Ø     ora #$1Ø    ... Bildschirm
. e37e  8d Ø6 ff sta $ffØ6    ... ein
. e381  ad Øa ff lda $ffØa    Gestattet Rasterinterrupt
. e384  29 f7     and #$f7    ... und unterbindet
. e386  Ø9 Ø2     ora #$Ø2    ... Interrupt bei
. e388  8d Øa ff sta $ffØa    ... Unterlauf von Timer 1
. e38b  58        cli         Interrupt zugelassen
. e38c  6Ø        rts         ENDE

. e38d  :
. e38d  *** MOTOR EINSCHALTEN ***
. e38d  :
. e38d  Ø8        php
. e38e  38        sec         Setzt
. e38f  6e fc Ø7 ror $Ø7fc    Motor-Flag
. e392  a5 Ø1     lda $Ø1     Schaltet
. e394  29 f5     and #$f5    ... Motor
. e396  85 Ø1     sta $Ø1     ... ein
. e398  a2 1e     ldx #$1e    Ruft 3Ømal
. e39a  2Ø ea e2 jsr $e2ea    ... Zeitschleife (16 Millisekunden)
. e39d  ca        dex         ... auf
. e39e  dØ fa     bne $e39a   ... Wartezeit zirka Ø.5 Sekunden
. e3aØ  28        plp
. e3a1  6Ø        rts         ENDE

. e3a2  :
. e3a2  *** COPYRIGHT ***
. e3a2  :
```

```
. e3a2  'c1984commodore'

. e3b0  :
. e3b0  *** MOTOR AUSSCHALTEN ***
. e3b0  :
. e3b0  a5 01    lda $01      Schaltet
. e3b1  09 08    ora $08      ... Motor
. e3b4  85 01    sta $01      ... aus
. e3b6  60       rts          ENDE

. e3b7  :
. e3b7  *** KASSETTENPUFFER LÖSCHEN ***
. e3b7  :
. e3b7  a0 00    ldy #$00     Initialisiert Offset im Kassettenpuffer
. e3b9  a9 20    lda #$20     Holt Leerzeichen
. e3bb  91 b6    sta ($b6),y  Schreibt Leerzeichen in Puffer
. e3bd  c8       iny          Erhöht Offset um 1
. e3be  c0 c0    cpy #$c0     Schon 192 Byte gelöscht ?
. e3c0  d0 f9    bne $e3bb    Nein, dann weiterlöschen
. e3c2  60       rts          ENDE

. e3c3  :
. e3c3  *** INITIALISIERT PUFFERZEIGER ***
. e3c3  :
. e3c3  48       pha
. e3c4  a9 33    lda #$33     Setzt
. e3c6  85 b6    sta $b6      ... Zeiger
. e3c8  a9 03    lda #$03     ... auf
. e3ca  85 b7    sta $b7      ... Adresse $0333
. e3cc  68       pla
. e3cd  60       rts          ENDE

. e3ce  :
. e3ce  *** ABBRUCH DURCH STOP-TASTE ***
. e3ce  :
. e3ce  20 cb fb jsr $fbcb    STOP-Taste gedrückt ?
. e3d1  90 10    bcc $e3e3    Nein, dann $e3e3
. e3d3  20 b0 e3 jsr $e3b0    Motor ausschalten
. e3d6  20 78 e3 jsr $e378    Kassettenport ausschalten
. e3d9  ae be 07 ldx $07be    Stellt
. e3dc  9a       txs          ... Stackmarkierung
. e3dd  a9 00    lda #$00     ... wieder
. e3df  8d be 07 sta $07be    ... her
. e3e2  38       sec
. e3e3  60       rts          ENDE

. e3e4  :
. e3e4  *** TIMER SETZEN ***
. e3e4  :
. e3e4  ad 09 ff lda $ff09    Ist Timer 1 abgelaufen
. e3e7  2d 0a ff and $ff0a    ... und Kassettenport
```

```
. e3ea  29 Ø8     and #$Ø8      ... eingeschaltet ?
. e3ec  dØ Ø1     bne $e3ef     Ja, dann $e3ef
. e3ee  6Ø        rts           ENDE

. e3ef  8d Ø9 ff  sta $ffØ9     Startet Timer
. e3f2  78        sei           Kein Interrupt zugelassen
. e3f3  a9 9Ø     lda #$9Ø      Setzt Timer 1
. e3f5  8d ØØ ff  sta $ffØØ     ... auf 132ØØ
. e3f8  a9 33     lda #$33      ... Zyklen
. e3fa  8d Ø1 ff  sta $ffØ1     ... (zirka 13.2 Millisekunden)
. e3fd  ae bf Ø7  ldx $Ø7bf     Holt Stackmarkierung
. e4ØØ  9a        txs           ... für DROP-Taste
. e4Ø1  38        sec
. e4Ø2  6Ø        rts           ENDE

. e4Ø3  a9 a8     lda #$a8      Setzt Timer 1
. e4Ø5  8d ØØ ff  sta $ffØØ     ... auf 186ØØ
. e4Ø8  a9 48     lda #$48      ... Zyklen
. e4Øa  8d Ø1 ff  sta $ffØ1     ... (zirka 18.6 Millisekunden)
. e4Ød  a9 Ø8     lda #$Ø8      Startet
. e4Øf  8d Ø9 ff  sta $ffØ9     ... Timer 1
. e412  6Ø        rts           ENDE

. e413  :
. e413  *** PULS SCHREIBEN ***
. e413  :
. e413  38        sec           Carry=1 (für einmalige Wiederholung)
. e414  bØ Ø1     bcs $e417     Springt immer nach $e417

. e416  18        clc           EINSPRUNG
. e417  8c ca Ø7  sty $Ø7ca     Y-Reg.
. e41a  8e cb Ø7  stx $Ø7cb     ... und X-Reg. zwischenspeichern
. e41d  ac c8 Ø7  ldy $Ø7c8     Holt
. e42Ø  ae c9 Ø7  ldx $Ø7c9     ... Wartezeit
. e423  a9 1Ø     lda #$1Ø      Unterlauf von
. e425  2c Ø9 ff  bit $ffØ9     ... Timer 2 ?
. e428  fØ fb     beq $e425     Nein, dann warten
. e42a  8c Ø2 ff  sty $ffØ2     Schreibt Wartezeit
. e42d  8e Ø3 ff  stx $ffØ3     ... nach Timer 2
. e43Ø  8d Ø9 ff  sta $ffØ9     Startet Timer 2
. e433  a5 Ø1     lda $Ø1       Invertiert
. e435  49 Ø2     eor #$Ø2      ... Bit 1
. e437  85 Ø1     sta $Ø1       ... (Datenpegel)
. e439  Ø8        php
. e43a  2Ø ce e3  jsr $e3ce     Abbruch, wenn STOP-Taste gedrückt
. e43d  28        plp
. e43e  ac ca Ø7  ldy $Ø7ca     Y-Reg.
. e441  ae cb Ø7  ldx $Ø7cb     ... und X-Reg. wiederherstellen
. e444  bØ dØ     bcs $e416     Zweifach-Aufruf ?, dann nochmal
. e446  6Ø        rts           ENDE

. e447  :
```

```
. e447  *** ZEITEN SETZEN ***
. e447  :
. e447  a9 4e    lda #$4e      Setzt
. e449  8d c8 Ø7 sta $Ø7c8     ... Zeit
. e44c  a9 Ø3    lda #$Ø3      ... gleich
. e44e  8d c9 Ø7 sta $Ø7c9     ... 846 Mikrosekunden
. e451  6Ø       rts           ENDE

. e452  a9 dØ    lda #$dØ      Setzt
. e454  8d c8 Ø7 sta $Ø7c8     ... Zeit
. e457  a9 ØØ    lda #$ØØ      ... gleich
. e459  8d c9 Ø7 sta $Ø7c9     ... 2Ø8 Mikrosekunden
. e45c  6Ø       rts           ENDE

. e45d  a9 a4    lda #$a4      Setzt
. e45f  8d c8 Ø7 sta $Ø7c8     ... Zeit
. e462  a9 Ø1    lda #$Ø1      ... gleich
. e464  8d c9 Ø7 sta $Ø7c9     ... 42Ø Mikrosekunden
. e467  6Ø       rts           ENDE

. e468  :
. e468  *** BIT Ø SENDEN ***
. e468  :
. e468  2Ø 52 e4 jsr $e452     Zeit gleich 2Ø8 Mikrosekunden
. e46b  2Ø 13 e4 jsr $e413     Schreibt Puls
. e46e  2Ø 5d e4 jsr $e45d     Zeit gleich 42Ø Mikrosekunden
. e471  4c 13 e4 jmp $e413     Schreibt Puls

. e474  :
. e474  *** BIT 1 SENDEN ***
. e474  :
. e474  2Ø 5d e4 jsr $e45d     Zeit gleich 42Ø Mikrosekunden
. e477  2Ø 13 e4 jsr $e413     Schreibt Puls
. e47a  2Ø 52 e4 jsr $e452     Zeit gleich 2Ø8 Mikrosekunden
. e47d  4c 13 e4 jmp $e413     Schreibt Puls

. e48Ø  :
. e48Ø  *** STARTBIT SENDEN ***
. e48Ø  :
. e48Ø  2Ø 47 e4 jsr $e447     Zeit gleich 846 Mikrosekunden
. e483  2Ø 13 e4 jsr $e413     Schreibt Puls
. e486  2Ø 5d e4 jsr $e45d     Zeit gleich 42Ø Mikrosekunden
. e489  4c 13 e4 jmp $e413     Schreibt Puls

. e48c  :
. e48c  *** BYTE SCHREIBEN ***
. e48c  :
. e48c  85 a7    sta $a7       Datenbyte zwischenspeichern
. e48e  a9 Ø1    lda #$Ø1      Holt Ein-/Ausgabe-Port
. e49Ø  8d b1 Ø7 sta $Ø7b1     Schreibt Prüfsumme
. e493  2Ø 8Ø e4 jsr $e48Ø     Sendet Startbit
. e496  a2 Ø8    ldx #$Ø8      Bitzähler
```

```
. e498  66 a7     ror $a7       Rotiert Bit für Bit ins Carry-Flag
. e49a  b0 09     bcs $e4a5     Bit 1 ?,dann $e4a5
. e49c  ee b1 07  inc $07b1     Erhöht Prüfsumme um 1
. e49f  20 68 e4  jsr $e468     Sendet Bit 0
. e4a2  4c a8 e4  jmp $e4a8     Springt nach $e4a8

. e4a5  20 74 e4  jsr $e474     Sendet Bit 1
. e4a8  ca        dex           Schon gesamtes Byte bearbeitet ?
. e4a9  d0 ed     bne $e498     Nein, dann weiter
. e4ab  6e b1 07  ror $07b1     Prüfsumme gleich 1 ?
. e4ae  b0 06     bcs $e4b6     Ja, dann $e4b6
. e4b0  20 68 e4  jsr $e468     Sendet Bit 0
. e4b3  4c b9 e4  jmp $e4b9     ENDE

. e4b6  20 74 e4  jsr $e474     Sendet Bit 1
. e4b9  60        rts           ENDE

. e4ba  :
. e4ba  *** STRING SCHREIBEN ***
. e4ba  :
. e4ba  ba        tsx           Stack-Zeiger
. e4bb  8e be 07  stx $07be     ... zwischenspeichern
. e4be  a5 01     lda $01       Legt
. e4c0  09 02     ora #$02      ... Datenausgang
. e4c2  85 01     sta $01       ... hoch
. e4c4  20 52 e4  jsr $e452     Zeit gleich 208 Mikrosekunden
. e4c7  a0 01     ldy #$01      Timer 2
. e4c9  8c 03 ff  sty $ff03     ... auf 256 Mikrosekunden
. e4cc  a9 10     lda #$10      ... stellen
. e4ce  8d 09 ff  sta $ff09     Timer 2 starten
. e4d1  24 f7     bit $f7       Pass 2 ?
. e4d3  10 04     bpl $e4d9     Ja, dann $e4d9
. e4d5  a0 40     ldy #$40
. e4d7  a2 fe     ldx #$fe
. e4d9  20 13 e4  jsr $e413     Schreibt Puls von 208 Mikrosekunden
. e4dc  ca        dex           64 * 254 Pulse
. e4dd  d0 fa     bne $e4d9     ... schreiben
. e4df  88        dey           ... Bei ebensolangen Pausen ergibt sich
. e4e0  d0 f7     bne $e4d9     ... so ein Vorspann von 6.8 Sekunden
. e4e2  a0 09     ldy #$09      Countdown (9,8,7....) senden
. e4e4  98        tya           Byte nach Akku
. e4e5  05 f7     ora $f7       Bit 7 ist in Pass 1 gesetzt
. e4e7  20 8c e4  jsr $e48c     Schreibt Byte
. e4ea  88        dey           Runterzählen
. e4eb  d0 f7     bne $e4e4     Noch nicht Null ?, dann weiter

-------------------------------------------------------------------------
. e4ed  a0 00     ldy #$00      Initialisiert   (Alte Betriebssystem-Version)
. e4ef  84 f5     sty $f5       ... Prüfsumme
. e4f1  a5 f8     lda $f8       Holt Block-Typ
. e4f3  f0 09     beq $e4fe     Block-Typ $00 ?, dann nicht schreiben
. e4f5  45 f5     eor $f5       Berechnet
```

```
. e4f7  85 f5     sta $f5       ... Block-Prüfsumme
. e4f9  a5 f8     lda $f8       Holt Block-Typ
. e4fb  20 8c e4  jsr $e48c     Schreibt Byte
. e4fe  b1 ba     lda ($ba),y   Holt Datenbyte
-------------------------------------------------------------------------
. e4ed  a5 f8     lda $f8       Schreibt Block-Typ
. e4ef  85 f5     sta $f5       ... in Register für Prüfsumme
. e4f1  f0 03     beq $e4f6     Block-Typ $00 ?, dann nicht schreiben
. e4f3  20 8c e4  jsr $e48c     Schreibt Byte
. e4f6  a0 00     ldy #$00      Holt Datenbyte
. e4f8  a9 ba     lda #$ba      ... aus
. e4fa  8d df 07  sta $07df     ... Kassettenpuffer
. e4fd  20 d9 07  jsr $07d9     ... Indirekter RAM-Zugriff (LDA($ba),Y)
. e500  48        pha           Byte aus Stack retten
. e501  45 f5     eor $f5       Berechnet
. e503  85 f5     sta $f5       ... Block-Prüfsumme
. e505  68        pla           Holt Datenbyte vom Stack
. e506  20 8c e4  jsr $e48c     Schreibt Byte
. e509  e6 ba     inc $ba       Erhöht
. e50b  d0 02     bne $e50f     ... Quellzeiger
. e50d  e6 bb     inc $bb       ... um 1
-------------------------------------------------------------------------
. e50f  ee f3 03  inc $03f3     Erhöht Byte-Zähler (Low) um 1      (Alte
. e512  d0 ea     bne $e4fe     Überlauf ?, dann Datenbyte holen    Betriebs-
. e514  ee f4 03  inc $03f4     Erhöht Byte-Zähler (High) um 1      System-
. e517  d0 d5     bne $e4fe     Überlauf ?, dann Datenbyte holen    Version)
-------------------------------------------------------------------------
. e50f  ee f3 03  inc $03f3     Erhöht Byte-Zähler (Low) um 1
. e512  d0 e2     bne $e4f6     Überlauf ?, dann Datenbyte holen
. e514  ee f4 03  inc $03f4     Erhöht Byte-Zähler (High) um 1
. e517  d0 dd     bne $e4f6     Überlauf ?, dann Datenbyte holen
. e519  a5 f5     lda $f5       Holt Block-Prüfsumme
. e51b  20 8c e4  jsr $e48c     Schreibt Byte
. e51e  20 5d e4  jsr $e45d     Zeit gleich 420 Mikrosekunden
. e521  20 13 e4  jsr $e413     Puls schreiben
. e524  20 52 e4  jsr $e452     Zeit gleich 208 Mikrosekunden
. e527  a0 01     ldy #$01
. e529  a2 c2     ldx #$c2
. e52b  20 13 e4  jsr $e413     Puls (208 Mikrosekunden) schreiben
. e52e  ca        dex           Nachspann besteht aus 194 * 2 Pulse zu
. e52f  d0 fa     bne $e52b     ... je 208 Mikrosekunden.
. e531  88        dey           ... Bei ebensolangen Pausen ergibt sich so
. e532  d0 f7     bne $e52b     ... ein Nachspann von 160 Millisekunden
. e534  60        rts           ENDE
. e535  :
. e535  *** PUFFER SCHREIBEN ***
. e535  :
. e535  20 19 e3  jsr $e319     Prüft PLAY-Taste
. e538  20 64 e3  jsr $e364     Kassettenport einschalten
. e53b  20 8d e3  jsr $e38d     Motor einschalten
. e53e  b0 26     bcs $e566     STOP-Taste gedrückt ?, dann $e566
. e540  a9 80     lda #$80      Setzt Flag
```

```
. e542  85 f7     sta $f7       ... für 1. Pass
. e544  a5 b6     lda $b6       Schreibt
. e546  85 ba     sta $ba       ... Pufferanfang
. e548  a5 b7     lda $b7       ... in
. e54a  85 bb     sta $bb       ... Quellzeiger
. e54c  a9 41     lda #$41      Schreibt $ff41
. e54e  8d f3 03  sta $03f3     ... ($FFFF-Pufferlänge)
. e551  a9 ff     lda #$ff      ... in
. e553  8d f4 03  sta $03f4     ... Byte-Zähler
. e556  20 ba e4  jsr $e4ba     String schreiben
. e559  b0 0b     bcs $e566     Fehler ?, dann $e566
. e55b  a5 f7     lda $f7       Pass 1 ?
. e55d  10 06     bpl $e565     Nein, dann $e565
. e55f  a9 00     lda #$00      Setzt Flag
. e561  85 f7     sta $f7       ... für 2. Pass
. e563  10 df     bpl $e544     Springt immer nach $e544

. e565  18        clc
. e566  20 b0 e3  jsr $e3b0     Motor ausschalten
. e569  4c 78 e3  jmp $e378     Kassettenport ausschalten

. e56c  :
. e56c  *** HEADERBLOCK SCHREIBEN ***
. e56c  :
. e56c  20 c3 e3  jsr $e3c3     Initialisiert Pufferzeiger
. e56f  20 b7 e3  jsr $e3b7     Löscht Kassettenpuffer
. e572  a0 00     ldy #$00
. e574  a5 b2     lda $b2       Schreibt
. e576  91 b6     sta ($b6),y   ... Anfangsadresse
. e578  c8        iny           ... in
. e579  a5 b3     lda $b3       ... Puffer...
. e57b  91 b6     sta ($b6),y
. e57d  c8        iny
. e57e  a5 9d     lda $9d       Schreibt
. e580  91 b6     sta ($b6),y   ... Endadresse
. e582  c8        iny           ... in
. e583  a5 9e     lda $9e       ... Puffer...
. e585  91 b6     sta ($b6),y
. e587  c8        iny
. e588  8c b3 07  sty $07b3     Schreibt Offset in Puffer
. e58b  a0 00     ldy #$00      Initialisiert
. e58d  8c b2 07  sty $07b2     ... Offset in Filenamen
. e590  ac b2 07  ldy $07b2     Holt Offset in Filenamen
. e593  c4 ab     cpy $ab       Länge des Dateinamens erreicht
. e595  f0 16     beq $e5ad     Ja, dann $e5ad
. e597  a9 af     lda #$af      Holt Adresse des Filenamens
. e599  8d df 07  sta $07df     Holt Zeichen
. e59c  20 d9 07  jsr $07d9     ... aus Filnamen (LDA(Akku),Y)
. e59f  ac b3 07  ldy $07b3     Holt Offset in Puffer
. e5a2  91 b6     sta ($b6),y   Schreibt Filenamen in Puffer
. e5a4  ee b2 07  inc $07b2     Erhöht Offset in Puffer
. e5a7  ee b3 07  inc $07b3     ... und Offset in Filenamen um 1
```

```
. e5aa  4c 90 e5 jmp $e590    Kopiert weiter

. e5ad  4c 35 e5 jmp $e535    Schreibt Puffer auf Band

. e5b0  :
. e5b0  *** SPEICHERBEREICH SCHREIBEN ***
. e5b0  :
. e5b0  20 19 e3 jsr $e319    Prüft PLAY-Taste
. e5b3  20 64 e3 jsr $e364    Kassettenport einschalten
. e5b6  20 8d e3 jsr $e38d    Motor einschalten
. e5b9  b0 2f    bcs $e5ea     STOP-Taste gedrückt ?, dann $e5ea
. e5bb  a9 80    lda #$80     Setzt Flag
. e5bd  85 f7    sta $f7      ... für 1. Pass
. e5bf  a5 b2    lda $b2      Kopiert
. e5c1  85 ba    sta $ba      ... Anfangsadresse
. e5c3  a5 b3    lda $b3      ... in
. e5c5  85 bb    sta $bb      ... Quellzeiger
. e5c7  18       clc          Subtrahiert
. e5c8  a5 9d    lda $9d      ... Anfangsadresse
. e5ca  e5 b2    sbc $b2      ... vom Ende
. e5cc  49 ff    eor #$ff     ... des Speicherbereichs,
. e5ce  8d f3 03 sta $03f3    ... invertiert
. e5d1  a5 9e    lda $9e      ... das Ergebnis
. e5d3  e5 b3    sbc $b3      ... und schreibt
. e5d5  49 ff    eor #$ff     ... invertiertes Ergebnis
. e5d7  8d f4 03 sta $03f4    ... in Byte-Zähler
. e5da  20 ba e4 jsr $e4ba    Schreibt String
. e5dd  b0 0b    bcs $e5ea    Fehler ?, dann $e5ea
. e5df  a5 f7    lda $f7      Pass 1 ?
. e5e1  10 06    bpl $e5e9    Nein, dann $e5e9
. e5e3  a9 00    lda #$00     Setzt Flag
. e5e5  85 f7    sta $f7      ... für Pass 2
. e5e7  10 d6    bpl $e5bf    Speicherbereich nochmal auf Band schreiben

. e5e9  18       clc
. e5ea  20 b0 e3 jsr $e3b0    Motor ausschalten
. e5ed  4c 78 e3 jmp $e378    Kassettenport ausschalten

. e5f0  :
. e5f0  *** EOT-BLOCK SCHREIBEN ***
. e5f0  :
. e5f0  20 b7 e3 jsr $e3b7    Löscht Kassettenpuffer
. e5f3  a9 05    lda #$05     Holt Filetyp 5
. e5f5  85 f8    sta $f8      ... (End Of Tape)
. e5f7  4c 35 e5 jmp $e535    Schreibt Puffer auf Band

. e5fa  40 00 80 .byte $40 $00 $80

. e5fd  :
. e5fd  *** PULS LESEN ***
. e5fd  :
. e5fd  ae b8 07 ldx $07b8    Holt Zeitkonstante
```

```
. e600  ac b9 07 ldy $07b9    ... für kurzen Puls
. e603  ad bb 07 lda $07bb    Holt Zeitkonstante
. e606  48       pha          ... für langen Puls
. e607  ad ba 07 lda $07ba    ... und legt diesen
. e60a  48       pha          ... auf Stack
. e60b  a9 10    lda #$10     Wartet
. e60d  24 01    bit $01      ... bis Data 1 am Port
. e60f  f0 fc    beq $e60d    ... anliegt
. e611  24 01    bit $01      Wartet
. e613  d0 fc    bne $e611    ... bis Data 0 am Port anliegt
. e615  8e 02 ff stx $ff02    Setzt Timer 2
. e618  8c 03 ff sty $ff03    ... auf kurzen Puls
. e61b  68       pla          Setzt
. e61c  8d 04 ff sta $ff04    ... Timer 3
. e61f  68       pla          ... auf
. e620  8d 05 ff sta $ff05    ... langen Puls
. e623  a9 50    lda #$50     Startet
. e625  8d 09 ff sta $ff09    ... Timer 2 und 3
. e628  a5 01    lda $01      Ist Signal am Port
. e62a  c5 01    cmp $01      ... stabil ?
. e62c  d0 fa    bne $e628    Nein, dann warten
. e62e  29 10    and #$10     Data 1 ?
. e630  d0 d1    bne $e603    Ja, dann nochmal von vorne
. e632  20 ce e3 jsr $e3ce    Abbruch, falls STOP-Taste gedrückt
. e635  a9 10    lda #$10
. e637  24 01    bit $01      Data 1 ?
. e639  d0 47    bne $e682    Ja, dann kurzer Puls gelesen
. e63b  2c 09 ff bit $ff09    Unterlauf Timer 2 ?
. e63e  f0 f7    beq $e637    Nein, dann warten
. e640  a5 01    lda $01      Ist Signal am Port
. e642  c5 01    cmp $01      ... stabil ?
. e644  d0 fa    bne $e640    Nein, dann warten
. e646  29 10    and #$10     Data 1 ?
. e648  d0 38    bne $e682    Ja, dann kurzer Puls gelesen
. e64a  a9 40    lda #$40     Unterlauf
. e64c  2c 09 ff bit $ff09    ... Timer 3 ?
. e64f  f0 fb    beq $e64c    Nein, dann warten
. e651  a5 01    lda $01      Ist Signal am Port
. e653  c5 01    cmp $01      ... stabil ?
. e655  d0 fa    bne $e651    Nein, dann warten
. e657  29 10    and #$10     Data 1 ?
. e659  d0 2c    bne $e687    Ja, dann langer Puls gelesen
. e65b  ad bc 07 lda $07bc    Belegt Timer 2 erneut
. e65e  8d 02 ff sta $ff02    ... mit Zeitkonstante
. e661  ad bd 07 lda $07bd    ... für
. e664  8d 03 ff sta $ff03    ... kurzen Puls
. e667  a9 10    lda #$10     Startet
. e669  8d 09 ff sta $ff09    ... Timer 2
. e66c  a9 10    lda #$10     Timer 2
. e66e  2c 09 ff bit $ff09    ... abgelaufen ?
. e671  f0 fb    beq $e66e    Nein, dann warten
. e673  a5 01    lda $01      Ist Signal am Port
```

```
. e675  c5 01    cmp $01       ... stabil ?
. e677  d0 fa    bne $e673     Nein, dann warten
. e679  29 10    and #$10      Data 0 ?
. e67b  f0 0f    beq $e68c     Ja, dann Fehler
. e67d  2c fc e5 bit $e5fc     Doppelt langer Puls setzt N-Flag
. e680  30 08    bmi $e68a     Springt nach $e68a

. e682  2c fa e5 bit $e5fa     Kurzer Puls setzt V-Flag
. e685  70 03    bvs $e68a     Springt nach $e68a

. e687  2c fb e5 bit $e5fb     Langer Puls setzt Z-Flag
. e68a  18       clc           Flag für 'OK'
. e68b  60       rts           ENDE

. e68c  38       sec           Setzt Fehler-Flag
. e68d  60       rts           ENDE

. e68e  40 00 80 .byte $40 $00 $80

. e691  :
. e691  *** BIT LESEN ***
. e691  :
. e691  20 fd e5 jsr $e5fd     Puls lesen
. e694  b0 3d    bcs $e6d3     Kein Puls ?, dann Fehler
. e696  70 12    bvs $e6aa     Kurzer Puls ?, dann $e6aa
. e698  10 02    bpl $e69c     Langer Puls ?, dann $e69c
. e69a  30 27    bmi $e6c3     Doppelt langer Puls ?, dann $e6c3

     LANGER PULS

. e69c  20 fd e5 jsr $e5fd     Puls lesen
. e69f  b0 32    bcs $e6d3     Kein Puls ?, dann Fehler
. e6a1  70 02    bvs $e6a5     Kurzer Puls ?, dann Bit 1
. e6a3  50 2e    bvc $e6d3     Sonst Fehler

     BIT 1

. e6a5  2c 8f e6 bit $e68f     Bit 1 setzt Z-Flag
. e6a8  18       clc           Flag für 'OK'
. e6a9  60       rts           ENDE

     KURZER PULS

. e6aa  20 fd e5 jsr $e5fd     Puls lesen
. e6ad  70 04    bvs $e6b3     Kurzer Puls ?, dann Vorspann
. e6af  10 0d    bpl $e6be     Langer Puls ?, dann Bit 0
. e6b1  30 20    bmi $e6d3     Sonst Fehler

     VORSPANN

. e6b3  20 fd e5 jsr $e5fd     Puls lesen
. e6b6  b0 1b    bcs $e6d3     Kein Puls ?, dann Fehler
```

```
. e6b8  70 f9     bvs $e6b3    Kurzer Puls ?, dann Vorspann
. e6ba  10 17     bpl $e6d3    Langer Puls ?, dann Fehler
. e6bc  30 05     bmi $e6c3    Doppelt langer Puls ?, dann Startbit

      BIT 0

. e6be  2c 8e e6 bit $e68e     Bit 0 setzt V-Flag
. e6c1  18       clc           Flag für 'OK'
. e6c2  60       rts           ENDE

      DOPPELT LANGER PULS

. e6c3  20 fd e5 jsr $e5fd     Puls lesen
. e6c6  b0 0b    bcs $e6d3     Kein Puls ?, dann Fehler
. e6c8  70 09    bvs $e6d3     Kurzer Puls ?, dann Fehler
. e6ca  10 02    bpl $e6ce     Langer Puls ?, dann Startbit
. e6cc  30 05    bmi $e6d3     Sonst Fehler

      STARTBIT

. e6ce  2c 90 e6 bit $e690     Startbit setzt N-Flag
. e6d1  18       clc           Flag für 'OK'
. e6d2  60       rts           ENDE

. e6d3  38       sec           Setzt Fehler-Flag
. e6d4  60       rts           ENDE

. e6d5  :
. e6d5  *** STARTBIT SUCHEN ***
. e6d5  :
. e6d5  ba       tsx           Stack-Zeiger
. e6d6  8e bf 07 stx $07bf     ... zwischenspeichern
. e6d9  18       clc           Setzt
. e6da  6e cc 07 ror $07cc     ... Startbit-Flag
. e6dd  58       cli           Kein Interrupt zugelassen
. e6de  20 91 e6 jsr $e691     Bit lesen
. e6e1  b0 fb    bcs $e6de     Kein Bit ?, dann weiterlesen
. e6e3  70 f9    bvs $e6de     Bit 0 ?, dann weiterlesen
. e6e5  10 f7    bpl $e6de     Bit 1 ?, dann weiterlesen
. e6e7  20 03 e4 jsr $e403     Timer 1 auf 16.8 ms setzen und starten
. e6ea  18       clc
. e6eb  60       rts           ENDE

. e6ec  :
. e6ec  *** BYTE LESEN ***
. e6ec  :
. e6ec  2c cc 07 bit $07cc     Flag für Startbit gesetzt ?
. e6ef  30 51    bmi $e742     Ja, dann Fehler
. e6f1  20 d5 e6 jsr $e6d5     Sucht Startbit
. e6f4  b0 4c    bcs $e742     STOP-Taste gedrückt ?, dann $e742
. e6f6  a9 01    lda #$01      Setzt
. e6f8  8d b1 07 sta $07b1     ... Prüfsumme auf 1
```

```
. e6fb  a2 Ø8     ldx #$Ø8      Setzt Bitzähler
. e6fd  8e b5 Ø7  stx $Ø7b5     ... auf 8 Bits
. e7ØØ  38        sec           Setzt Flag
. e7Ø1  6e cc Ø7  ror $Ø7cc     ... für Startbit
. e7Ø4  2Ø 91 e6  jsr $e691     Bit lesen
. e7Ø7  bØ 39     bcs $e742     Kein Bit ?, dann Fehler
. e7Ø9  7Ø Ø4     bvs $e7Øf     Bit Ø ?, dann $e7Øf
. e7Øb  1Ø Øf     bpl $e71c     Bit 1 ?, dann $e71c
. e7Ød  3Ø 33     bmi $e742     Sonst Fehler

. e7Øf  18        clc           Bit Ø
. e71Ø  66 a7     ror $a7       ... einrotieren
. e712  ee b1 Ø7  inc $Ø7b1     Erhöht Prüfsumme um 1
. e715  ce b5 Ø7  dec $Ø7b5     Schon gesamtes Byte gelesen ?
. e718  dØ ea     bne $e7Ø4     Nein, dann weiterlesen
. e71a  fØ Ø8     beq $e724     Ja, dann $e724

. e71c  38        sec           Bit 1
. e71d  66 a7     ror $a7       ... einrotieren
. e71f  ce b5 Ø7  dec $Ø7b5     Schon gesamtes Byte gelesen ?
. e722  dØ eØ     bne $e7Ø4     Nein, dann weiterlesen
. e724  2Ø 91 e6  jsr $e691     Holt Prüfsummen-Bit
. e727  bØ 19     bcs $e742     Kein Bit ?, dann Fehler
. e729  7Ø Ø4     bvs $e72f     Bit Ø ? ,dann $e72f
. e72b  1Ø Øb     bpl $e738     Bit 1 ?, dann $e738
. e72d  3Ø 13     bmi $e742     Sonst Fehler

. e72f  ad b1 Ø7  lda $Ø7b1     Prüfsumme
. e732  29 Ø1     and #$Ø1      ... gleich Ø ?
. e734  dØ Øc     bne $e742     Nein, dann Fehler
. e736  fØ Ø7     beq $e73f     Ja, dann $e73f

. e738  ad b1 Ø7  lda $Ø7b1     Prüfsumme
. e73b  29 Ø1     and #$Ø1      ... gleich 1 ?
. e73d  fØ Ø3     beq $e742     Nein, dann Fehler
. e73f  18        clc           Flag für 'OK'
. e74Ø  9Ø Ø1     bcc $e743     Springt immer nach $e743

. e742  38        sec           Setzt Flag für Fehler
. e743  78        sei           Kein Interrupt zugelassen
. e744  Ø8        php           Fehler/OK-Flag retten
. e745  18        clc           Setzt
. e746  6e cc Ø7  ror $Ø7cc     ... Startbit-Flag zurück
. e749  28        plp           Fehler/OK-Flag holen
. e74a  6Ø        rts           ENDE

. e74b  :
. e74b  *** STRING LESEN ***
. e74b  :
. e74b  ba        tsx           Stack-Zeiger
. e74c  8e be Ø7  stx $Ø7be     ... zwischenspeichern
. e74f  a5 93     lda $93       Holt LOAD/VERIFY-Flag
```

```
. e751  f0 03    beq $e756    LOAD ?, dann $e756
. e753  38       sec          Setzt
. e754  66 93    ror $93      ... Verify-Flag
. e756  20 8d e3 jsr $e38d    Motor einschalten
. e759  20 64 e3 jsr $e364    Kassettenport einschalten
. e75c  ad c0 07 lda $07c0    Kopiert
. e75f  85 b6    sta $b6      ... Anfangsadresse
. e761  ad c1 07 lda $07c1    ... in
. e764  85 b7    sta $b7      ... Zielzeiger
. e766  ad c2 07 lda $07c2    Schreibt
. e769  8d f5 03 sta $03f5    ... invertierte
. e76c  ad c3 07 lda $07c3    ... Stringlänge
. e76f  8d f6 03 sta $03f6    ... in Byte-Zähler
. e772  20 1d e9 jsr $e91d    Lesen vorbereiten
. e775  a0 00    ldy #$00     Initialisiert
. e777  8c b6 07 sty $07b6    ... Fehlerzähler 1,
. e77a  8c b7 07 sty $07b7    ... Fehlerzähler 2,
. e77d  84 f5    sty $f5      ... Prüfsumme
. e77f  84 b1    sty $b1      ... Fehlerbilanz
. e781  84 f8    sty $f8      ... und File-Typ
-------------------------------------------------------------------------
. e783  88       dey          Setzt        (Alte Betriebssystem-Version)
. e784  84 a9    sty $a9      ... Fehler-Flag für Pass 1
. e786  84 92    sty $92      ... und Fehler-Flag für Pass 2
-------------------------------------------------------------------------
. e783  a9 b6    lda #$b6     Schreibt Adresse $b6
. e785  8d df 07 sta $07df    ... für indirekten RAM-Zugriff
. e788  2c b0 07 bit $07b0    Wird in Puffer gelesen ?
. e78b  10 13    bpl $e7a0    Nein, dann $e7a0
. e78d  20 ec e6 jsr $e6ec    Byte lesen
. e790  b0 0b    bcs $e79d    Fehler ?, dann $e79d
. e792  a5 a7    lda $a7      Holt Datenbyte
. e794  85 f8    sta $f8      Schreibt Filetyp
. e796  45 f5    eor $f5      Berechnet
. e798  85 f5    sta $f5      ... Prüfsumme
. e79a  4c a0 e7 jmp $e7a0    Springt nach $e7a0

. e79d  38       sec          Setzt Bit
. e79e  66 f8    ror $f8      ... im File-Typ
. e7a0  20 ec e6 jsr $e6ec    Byte lesen
. e7a3  b0 19    bcs $e7be    Fehler ?, dann $e7be
```

```
. e7a5  a5 a7     lda $a7       Holt Datenbyte (Alte Betriebssystem-Version)
. e7a7  a0 00     ldy #$00      Überprüft
. e7a9  24 93     bit $93       ... LOAD/VERIFY-Flag
. e7ab  10 08     bpl $e7b5     LOAD ?, dann $e7b5
. e7ad  d1 b6     cmp ($b6),y   Vergleicht Datenbyte mit Byte im Speicher
. e7af  f0 04     beq $e7b5     Bytes identisch ?, dann $e7b5
. e7b1  84 a9     sty $a9       Setzt Fehlerflag für Pass 1
. e7b3  d0 02     bne $e7b7     Springt immer nach $e7b7
```

```
. e7a5  a0 00     ldy #$00      Holt Byte
. e7a7  20 d9 07  jsr $07d9     ... aus Speicher (LDA($b6),Y)
. e7aa  ea        nop
. e7ab  24 93     bit $93       Überprüft LOAD/VERIFY-Flag
. e7ad  30 02     bmi $e7b1     VERIFY ?, dann $e7b1
. e7af  a5 a7     lda $a7       Holt Datenbyte
. e7b1  c5 a7     cmp $a7       Vergleicht Datenbyte mit Byte im Speicher
. e7b3  d0 09     bne $e7be     Bytes nicht identisch ?, dann Fehler
. e7b5  91 b6     sta ($b6),y   Speichert Datenbyte
. e7b7  45 f5     eor $f5       Berechnet
. e7b9  85 f5     sta $f5       ... Prüfsumme
. e7bb  4c dc e7  jmp $e7dc     Springt immer nach $e7dc

. e7be  ac b6 07  ldy $07b6     Holt Fehlerzähler 1
. e7c1  c0 1e     cpy #$1e      Schon zu viele Fehler ? (max. 30)
. e7c3  b0 12     bcs $e7d7     Ja, dann $e7d7
. e7c5  a5 b6     lda $b6       Speichert Fehleradresse
. e7c7  99 37 04  sta $0437,y   ... für Vergleich
. e7ca  a5 b7     lda $b7       ... in
. e7cc  99 55 04  sta $0455,y   ... Pass 2
. e7cf  ee b6 07  inc $07b6     Erhöht Fehlerzähler 1 um 1
. e7d2  e6 b1     inc $b1       Erhöht Fehlerbilanz
. e7d4  4c dc e7  jmp $e7dc     Springt immer nach $e7dc

. e7d7  a9 ff     lda #$ff      Setzt Fehlerzähler 1
. e7d9  8d b6 07  sta $07b6     ... auf $ff
. e7dc  e6 b6     inc $b6       Erhöht
. e7de  d0 02     bne $e7e2     ... Zielzeiger
. e7e0  e6 b7     inc $b7       ... um 1
. e7e2  ee f5 03  inc $03f5     Erhöht
. e7e5  d0 b9     bne $e7a0     ... Bytezähler
. e7e7  ee f6 03  inc $03f6     ... um 1
. e7ea  d0 b4     bne $e7a0     Noch nicht alle Bytes gelesen ?, dann weiter
. e7ec  ad b6 07  lda $07b6     Holt Fehlerzähler 1
. e7ef  8d b7 07  sta $07b7     ... und schreibt Fehlerzähler 2
. e7f2  20 ec e6  jsr $e6ec     Byte lesen
. e7f5  ad b7 07  lda $07b7     Holt Fehlerzähler 2
. e7f8  d0 06     bne $e800     Kein Fehler ?, dann $e800
. e7fa  a5 a7     lda $a7       Holt Datenbyte
. e7fc  c5 f5     cmp $f5       Vergleicht Datenbyte mit Prüfsumme
. e7fe  d0 03     bne $e803     Beide Bytes nicht identisch ?, dann $e803
. e800  4c 0a e8  jmp $e80a     Springt nach $e80a
```

```
. e8Ø3  a5 f7     lda $f7       Pass 1 ?
. e8Ø5  3Ø Ø3     bmi $e8Øa     Ja, dann $e8Øa
. e8Ø7  4c b7 e8  jmp $e8b7     Sonst Fehler

. e8Øa  a5 f7     lda $f7       Pass 1 ?
. e8Øc  3Ø Øb     bmi $e819     Ja, dann $e819
. e8Øe  ad b7 Ø7  lda $Ø7b7     Keine Fehler in Pass 1 ?
. e811  fØ Ø3     beq $e816     Ja, dann $e816
. e813  4c b7 e8  jmp $e8b7     Sonst Fehler

. e816  4c c7 e8  jmp $e8c7     Kassettenoperation abschließen

. e819  a9 ØØ     lda #$ØØ      Initialisiert
. e81b  8d b6 Ø7  sta $Ø7b6     ... Fehlerzähler 1
. e81e  85 f5     sta $f5       ... und Prüfsumme
. e82Ø  ad cØ Ø7  lda $Ø7cØ     Kopiert
. e823  85 b6     sta $b6       ... Anfangsadresse
. e825  ad c1 Ø7  lda $Ø7c1     ... in
. e828  85 b7     sta $b7       ... Zielzeiger
. e82a  ad c2 Ø7  lda $Ø7c2     Schreibt
. e82d  8d f5 Ø3  sta $Ø3f5     ... invertierte
. e83Ø  ad c3 Ø7  lda $Ø7c3     ... Stringlänge
. e833  8d f6 Ø3  sta $Ø3f6     ... in Byte-Zähler
. e836  2Ø 1d e9  jsr $e91d     Lesen vorbereiten
. e839  2c bØ Ø7  bit $Ø7bØ     Wird im Puffer gelesen ?
. e83c  1Ø 15     bpl $e853     Nein, dann $e853
. e83e  2Ø ec e6  jsr $e6ec     Byte lesen
. e841  24 f8     bit $f8       Bit 7 im File-Typ gesetzt ?
. e843  1Ø Ø8     bpl $e84d     Nein, dann $e84d
. e845  a5 a7     lda $a7       Holt Datenbyte
. e847  85 f8     sta $f8       Schreibt File-Typ
. e849  9Ø Ø2     bcc $e84d     Kein Fehler ?, dann $e84d
. e84b  66 f8     ror $f8       Rotiert Ø ein
. e84d  a5 f8     lda $f8       Holt File-Typ
. e84f  45 f5     eor $f5       Berechnet
. e851  85 f5     sta $f5       ... Prüfsumme
. e853  2Ø ec e6  jsr $e6ec     Byte lesen
. e856  6e c4 Ø7  ror $Ø7c4     Lesefehler ?, dann Bit 7 setzen
. e859  a5 a7     lda $a7       Holt Datenbyte
. e85b  45 f5     eor $f5       Berechnet
. e85d  85 f5     sta $f5       ... Prüfsumme
. e85f  2c b7 Ø7  bit $Ø7b7     Bit 7 Fehlerzähler 2 gesetzt
. e862  3Ø 32     bmi $e896     Ja, dann $e896
. e864  ac b6 Ø7  ldy $Ø7b6     Vergleicht Fehlerzähler 1
. e867  cc b7 Ø7  cpy $Ø7b7     ... mit Fehlerzähler 2
. e86a  fØ 2a     beq $e896     Gleich viele Fehler ?, dann $e896
. e86c  b9 37 Ø4  lda $Ø437,y   Vergleicht Fehler
. e86f  c5 b6     cmp $b6       ... aus Pass 1
. e871  dØ 23     bne $e896     ... mit
. e873  b9 55 Ø4  lda $Ø455,y   ... Zielzeiger
. e876  c5 b7     cmp $b7
```

```
. e878  dØ 1c    bne $e896    Nicht gleich ?, dann $e896
. e87a  ee b6 Ø7 inc $Ø7b6    Erhöht Fehlerzähler 1 um 1
. e87d  ad c4 Ø7 lda $Ø7c4    Fehler beim Lesen ?
. e88Ø  3Ø 14    bmi $e896    Ja, dann $e896
. e882  aØ ØØ    ldy #$ØØ
```

```
. e884  a5 a7    lda $a7      Holt Datenbyte  (Alte Betriebssystem-Version)
. e886  24 93    bit $93      Überprüft LOAD/VERIFY-Flag
. e888  1Ø Ø8    bpl $e892    LOAD ?, dann $e892
. e88a  d1 b6    cmp ($b6),y  Vergleicht Datenbyte mit Byte im Speicher
. e88c  fØ Ø4    beq $e892    Bytes identisch ?, dann $e892
. e88e  84 92    sty $92      Setzt Fehler-Flag für Pass 2
```

```
. e884  2Ø d9 Ø7 jsr $Ø7d9    Holt Byte aus Speicher (LDA($b6),Y)
. e887  ea       nop
. e888  24 93    bit $93      Überprüft LOAD/VERIFY-Flag
. e88a  3Ø Ø2    bmi $e88e    VERIFY ?, dann $e88e
. e88c  a5 a7    lda $a7      Holt Datenbyte
. e88e  c5 a7    cmp $a7      Vergleicht Datenbyte mit Byte im Speicher
. e89Ø  dØ Ø4    bne $e896    Bytes nicht identisch ?, dann $e896
. e892  c6 b1    dec $b1      Erniedrigt Offset in Fehlertabelle um 1
. e894  91 b6    sta ($b6),y  Speichert Datenbyte
. e896  e6 b6    inc $b6      Erhöht
. e898  dØ Ø2    bne $e89c    ... Zielzeiger
. e89a  e6 b7    inc $b7      ... um 1
. e89c  ee f5 Ø3 inc $Ø3f5    Erhöht
. e89f  dØ b2    bne $e853    ... Byte-Zähler
. e8a1  ee f6 Ø3 inc $Ø3f6    ... um 1
. e8a4  dØ ad    bne $e853    Noch nicht alle Bytes gelesen ?, dann weiter
. e8a6  2Ø ec e6 jsr $e6ec    Byte lesen
```

```
. e8a9  a5 b1    lda $b1      Holt Offset in Fehlertabelle     (Alte
. e8ab  dØ Øa    bne $e8b7    Noch nicht Null ?, dann Fehler    Betriebs-
. e8ad  a5 92    lda $92      Fehler-Flag von Pass 1            System-
. e8af  25 a9    and $a9      ... und Pass 2 gleich Null ?      Version)
. e8b1  fØ Øc    beq $e8bf    Ja, dann $e8bf
. e8b3  a5 f8    lda $f8      Bit 7 im File-Typ gesetzt ?
. e8b5  1Ø 1Ø    bpl $e8c7    Nein, dann $e8c7
. e8b7  a9 6Ø    lda #$6Ø     Odert $6Ø
. e8b9  2Ø 1e f4 jsr $f41e    ... in Status
. e8bc  38       sec
. e8bd  bØ Ø9    bcs $e8c8    Springt immer nach $e8c8

. e8bf  a9 1Ø    lda #$1Ø     Odert $1Ø
. e8c1  2Ø 1e f4 jsr $f41e    ... in Status
. e8c4  38       sec
. e8c5  bØ Ø1    bcs $e8c8    Springt immer nach $e8c8
```

```
. e8a9  a9 00     lda #$00      Initialisiert  (Neue Betriebssystem-Version)
. e8ab  85 90     sta $90       ... Status
. e8ad  a5 f8     lda $f8       Holt File-Typ
. e8af  a6 b1     ldx $b1       Holt Offset in Fehlertabelle
. e8b1  f0 14     beq $e8c7     File-Typ gleich Null ?, dann $e8c7
. e8b3  24 93     bit $93       Überprüft LOAD/VERIFY-Flag
. e8b5  30 08     bmi $e8bf     VERIFY ?, dann $e8bf
. e8b7  a9 60     lda #$60      Schreibt $60
. e8b9  85 90     sta $90       ... in Status
. e8bb  38        sec           Flag für Fehler
. e8bc  4c c8 e8  jmp $e8c8     Kassettenoperation abschließen

. e8bf  a9 10     lda #$10      Schreibt $10
. e8c1  85 90     sta $90       ... in Status
. e8c3  38        sec           Flag für Fehler
. e8c4  4c c8 e8  jmp $e8c8     Kassettenoperation abschließen

. e8c7  18        clc           Flag für kein Fehler
. e8c8  20 b0 e3  jsr $e3b0     Motor ausschalten
. e8cb  20 78 e3  jsr $e378     Kassettenport ausschalten
. e8ce  60        rts           ENDE

. e8cf  33 03     .adr $0333    Anfangsadresse Kassettenpuffer
. e8d1  41 ff     .adr $ff41    $ffff - Länge des Kassettenpuffers ($be)

. e8d3  :
. e8d3  *** BLOCK VOM BANDPUFFER LESEN ***
. e8d3  :
. e8d3  a0 03     ldy #$03      Kopiert
. e8d5  b9 cf e8  lda $e8cf,y   ... Adressen
. e8d8  99 c0 07  sta $07c0,y   ... nach
. e8db  88        dey           ... $07c0-$07c3
. e8dc  10 f7     bpl $e8d5
. e8de  8c b0 07  sty $07b0     Flag für 'Lesen im Puffer' initialisieren
. e8e1  a5 93     lda $93       Holt LOAD/VERIFY-Flag
. e8e3  48        pha           Flag auf Stack retten
. e8e4  c8        iny           Erhöht Y-Reg. um 1
. e8e5  84 93     sty $93       Setzt VERIFY-Flag
. e8e7  8c 39 05  sty $0539     Setzt Pointer auf 1. Zeichen im Puffer
. e8ea  20 4b e7  jsr $e74b     String lesen
. e8ed  68        pla           LOAD/VERIFY-Flag
. e8ee  85 93     sta $93       ... wiederherstellen
. e8f0  4c c3 e3  jmp $e3c3     Initialisiert Pufferzeiger

. e8f3  :
. e8f3  *** PROGRAMMFILE LESEN ***
. e8f3  :
. e8f3  a5 b2     lda $b2       Schreibt
. e8f5  8d c0 07  sta $07c0     ... Ladeanfang
. e8f8  a5 b3     lda $b3       ... in
. e8fa  8d c1 07  sta $07c1     ... Zielzeiger
. e8fd  18        clc           Berechnet Länge
```

```
. e8fe  a5 9d     lda $9d       ... Länge des
. e900  e5 b2     sbc $b2       ... zu ladenden Files,
. e902  49 ff     eor #$ff      ... invertiert
. e904  8d c2 07  sta $07c2     ... das Ergebnis
. e907  a5 9e     lda $9e       ... und schreibt
. e909  e5 b3     sbc $b3       ... das invertierte
. e90b  49 ff     eor #$ff      ... Ergebnis
. e90d  8d c3 07  sta $07c3     ... in den Byte-Zähler
. e910  18        clc           Flag für
. e911  6e b0 07  ror $07b0     ... 'Lesen im Puffer' initialisieren
. e914  4c 4b e7  jmp $e74b     String lesen

. e917  :
. e917  *** PULSTABELLE ***
. e917  :
. e917  02 01     .time $0102   Zeitvorgabe für 'Kurzen Puls'
. e919  02 02     .time $0202   Zeitvorgabe für 'Langen Puls'
. e91b  0d 02     .time $020d   Zeitvorgabe für 'Doppelt langen Puls'

. e91d  :
. e91d  *** LESEN VORBEREITEN ***
. e91d  :
. e91d  a2 05     ldx #$05      Übernimmt
. e91f  bd 17 e9                ... Zeitkonstanten
. e922  9d b8 07                ... aus
. e925  ca        dex           ... Pulstabelle
. e926  10 f7     bpl $e91f
. e928  a9 0a     lda #$0a      Setzt Zähler
. e92a  8d c5 07  sta $07c5     ... auf 10
. e92d  20 fd e5  jsr $e5fd     Puls lesen
. e930  b0 f6     bcs $e928     Kein Puls ?, dann nochmal
. e932  50 f4     bvc $e928     kein kurzer Puls ?, dann nochmal
. e934  ce c5 07  dec $07c5     Erniedrigt Zähler um 1
. e937  d0 f4     bne $e92d     Noch keine 10 Pulse gelesen ?, dann $e92d
. e939  a9 00     lda #$00      Initialisiert
. e93b  85 ba     sta $ba       ... Gesamtzeit-
. e93d  85 bb     sta $bb       ... zähler
. e93f  a0 10     ldy #$10      Holt Zähler für Pulse
. e941  a2 00     ldx #$00      Holt Zähler für Zeitschleifen
. e943  a9 10     lda #$10      Setzt Bit 4
. e945  24 01     bit $01       Data 0 ?
. e947  f0 fc     beq $e945     Ja, dann warten
. e949  24 01     bit $01       Data 1 ?
. e94b  d0 fc     bne $e949     Ja, dann warten
. e94d  e8        inx           Erhöht Zähler für Zeitschleife um 1
. e94e  f0 e9     beq $e939     Überlauf ?, dann alles wiederholen
. e950  24 01     bit $01       Data 0 ?
. e952  f0 f9     beq $e94d     Ja, dann Zähler f. Zeilschl. erhöhen und warten
. e954  e8        inx           Erhöht Zähler für Zeitschleife
. e955  f0 e2     beq $e939     Überlauf ?, dann alles wiederholen
. e957  24 01     bit $01       Data 1 ?
. e959  d0 f9     bne $e954     Ja, dann Zähler f. Zeilschl. erhöhen und warten
```

```
. e95b  8a       txa          Addiert
. e95c  18       clc          ... Zählerstand
. e95d  65 ba    adc $ba      ... zum
. e95f  85 ba    sta $ba      ... Gesamtzeit-
. e961  a9 ØØ    lda #$ØØ     ... zähler...
. e963  65 bb    adc $bb
. e965  85 bb    sta $bb
. e967  88       dey          Schon 16 Takte gemessen ?
. e968  dØ d7    bne $e941    Nein, dann $e941
. e96a  46 bb    lsr $bb      Dividiert
. e96c  66 ba    ror $ba      ... Gesamtzeit
. e96e  46 bb    lsr $bb      ... durch
. e97Ø  66 ba    ror $ba      ... vier
. e972  a5 ba    lda $ba      Schreibt Ergebnis
. e974  8d b8 Ø7 sta $Ø7b8    ... als Zeitkonstante
. e977  Øa       asl          ... für 'Kurzen Puls',
. e978  8d ba Ø7 sta $Ø7ba    ... und
. e97b  8d bc Ø7 sta $Ø7bc    ... verdoppeltes
. e97e  a5 bb    lda $bb      ... Ergebnis
. e98Ø  8d b9 Ø7 sta $Ø7b9    ... als Zeitkonstante
. e983  2a       rol          ... für 'Langen Puls'
. e984  8d bb Ø7 sta $Ø7bb    ... und
. e987  8d bd Ø7 sta $Ø7bd    ... 'Doppelt Langen Puls'
. e98a  2Ø fd e5 jsr $e5fd    Puls lesen
. e98d  bØ fb    bcs $e98a    Kein Puls ?, dann nochmal
. e98f  7Ø f9    bvs $e98a    Kurzer Puls ?, dann nochmal
. e991  1Ø f7    bpl $e98a    Langer Puls ?, dann nochmal
. e993  2Ø fd e5 jsr $e5fd    Puls lesen
. e996  bØ f2    bcs $e98a    Kein Puls ?, dann nochmal
. e998  7Ø fØ    bvs $e98a    Kurzer Puls ?, dann nochmal
. e99a  3Ø ee    bmi $e98a    Doppelt langer Puls ?, dann nochmal
. e99c  18       clc          Startbit erkannt
. e99d  6e cc Ø7 ror $Ø7cc    Startbit-Flag initialisieren
. e9aØ  2Ø Ø3 e4 jsr $e4Ø3    Timer 1 auf 18.6 ms setzen und starten
. e9a3  a9 Ø3    lda #$Ø3     Setzt Fehlerzähler
. e9a5  8d c6 Ø7 sta $Ø7c6    ... auf 3
. e9a8  2Ø f6 e6 jsr $e6f6    Byte lesen
. e9ab  9Ø Ø3    bcc $e9bØ    Kein Fehler ?, dann $e9bØ
. e9ad  ce c6 Ø7 dec $Ø7c6    Fehlerzähler um 1 erniedrigen
. e9bØ  2Ø ec e6 jsr $e6ec    Byte lesen
. e9b3  9Ø Ø8    bcc $e9bd    Kein Fehler ?, dann $e9bd
. e9b5  ce c6 Ø7 dec $Ø7c6    Fehlerzähler um 1 erniedrigen
. e9b8  dØ Ø3    bne $e9bd    Noch keine 3 Fehler ?, dann $e9bd
. e9ba  4c 1d e9 jmp $e91d    Erneut Lesen vorbereiten

. e9bd  a5 a7    lda $a7      Holt gelesenes Byte
. e9bf  29 Øf    and #$Øf     Isoliert unteres Halbbyte
. e9c1  c9 Ø1    cmp #$Ø1     Ergebnis gleich 1 ? (Countdown zu Ende)
. e9c3  dØ eb    bne $e9bØ    Nein, dann weiterlesen
. e9c5  a5 a7    lda $a7      Holt gelesenes Byte
. e9c7  29 8Ø    and #$8Ø     Isoliert Bit 7 (Pass 1: $8Ø, Pass 2: $ØØ)
. e9c9  85 f7    sta $f7      Setzt Flag für Pass
```

```
. e9cb  60       rts          ENDE

. e9cc  :
. e9cc  *** FILEHEADER LESEN ***
. e9cc  :
. e9cc  20 d3 e8 jsr $e8d3    Block vom Bandpuffer lesen
. e9cf  b0 4d    bcs $ea1e    Fehler ?, dann $ea1e
. e9d1  a5 f8    lda $f8      Holt File-Typ
. e9d3  c9 05    cmp #$05     'End Of Tape' ?
. e9d5  f0 43    beq $ea1a    Ja, dann $ea1a
. e9d7  c9 01    cmp #$01     'Programm' ?
. e9d9  f0 08    beq $e9e3    Ja, dann $e9e3
. e9db  c9 03    cmp #$03     File-Typ 3 ?
. e9dd  f0 04    beq $e9e3    Ja, dann $e9e3
. e9df  c9 04    cmp #$04     'Daten' ?
. e9e1  d0 e9    bne $e9cc    Nein, dann $e9cc
. e9e3  aa       tax          File-Typ nach X-Reg.
. e9e4  24 9a    bit $9a      Meldung zugelassen ?
. e9e6  10 2f    bpl $ea17    Nein, dann $ea17
. e9e8  20 d8 fb jsr $fbd8    Meldung ausgeben

. e9eb  0d       .byte $0d    CR-Code
. e9ec  46 4f 55 4e 44 20     'found '
. e9f2  00       .byte $00    Endmarke

. e9f3  a0 04    ldy #$04     Erste 4 Zeichen überlesen
. e9f5  b1 b6    lda ($b6),y  Holt Zeichen aus Dateinamen
. e9f7  20 d2 ff jsr $ffd2    (BSOUT) Zeichen ausgeben
. e9fa  c8       iny          Erhöht Zeichenzähler
. e9fb  c0 15    cpy #$15     Schon 16 Zeichen ausgegeben ?
. e9fd  d0 f6    bne $e9f5    Nein, dann $e9f5
. e9ff  a2 ff    ldx #$ff     Warteschleife:
. ea01  20 ea e2 jsr $e2ea    Zeitschleife 16 Millisekunden
. ea04  20 ea e2 jsr $e2ea    Zeitschleife 16 Millisekunden
. ea07  ca       dex          Zähler erniedrigen
. ea08  f0 0d    beq $ea17    8 Sekunden verstrichen ?, dann $ea17
. ea0a  a9 7f    lda #$7f
. ea0c  20 70 db jsr $db70    Decoder Abfrage
. ea0f  c9 7f    cmp #$7f     STOP-Taste gedrückt ?
. ea11  f0 0b    beq $ea1e    Ja, dann Abbruch
. ea13  c9 df    cmp #$df     Commodore-Taste gedrückt ?
. ea15  d0 ea    bne $ea01    Nein, dann warten
. ea17  18       clc          Flag für 'OK'
. ea18  a5 f8    lda $f8      Holt File-Typ
. ea1a  60       rts          ENDE

. ea1b  ea       nop
. ea1c  ea       nop
. ea1d  ea       nop

. ea1e  a9 00    lda #$00
. ea20  60       rts          ENDE
```

```
. ea21  :
. ea21  *** HEADER IDENTIFIZIEREN ***
. ea21  :
. ea21  20 cc e9 jsr $e9cc    Fileheader lesen
. ea24  b0 2d    bcs $ea53    Abbruch ?, dann $ea53
. ea26  c9 05    cmp #$05     'End Of Tape' ?
. ea28  f0 2b    beq $ea55    Ja, dann $ea55
. ea2a  a0 ff    ldy #$ff
. ea2c  c8       iny          Erhöht Offset im Filenamen
. ea2d  c4 ab    cpy $ab      Ende des Filenamens erreicht ?
. ea2f  f0 26    beq $ea57    Ja, dann $ea57
. ea31  a9 af    lda #$af     Schreibt Adresse des Filenamens nach
. ea33  8d df 07 sta $07df    ... $07df für indirekten RAM-Zugriff
. ea36  20 d9 07 jsr $07d9    Holt Zeichen aus Filenamen (LDA($af),Y)
. ea39  d9 37 03 cmp $0337,y  Vergleich mit Namen im Puffer
. ea3c  f0 ee    beq $ea2c    Beide Zeichen identisch ?, dann $ea2c
. ea3e  46 f8    lsr $f8      Verschiebt File-Typ um 1 Bit nach rechts
. ea40  90 df    bcc $ea21    War Bit 1 gesetzt ?, dann $ea21
. ea42  a0 ff    ldy #$ff     Setzt
. ea44  8c c3 07 sty $07c3    ... Byte-Zähler
. ea47  88       dey          ... auf
. ea48  8c c2 07 sty $07c2    ... $fffe
. ea4b  a0 01    ldy #$01     Nächsten Block
. ea4d  20 d5 e8 jsr $e8d5    ... lesen
. ea50  4c 21 ea jmp $ea21    Header identifizieren

. ea53  a9 00    lda #$00
. ea55  38       sec          Setzt Fehler-Flag
. ea56  60       rts          ENDE

. ea57  18       clc          Setzt 'OK'-Flag
. ea58  a5 f8    lda $f8      Holt File-Typ
. ea5a  60       rts          ENDE

. ea5b  :
. ea5b  *** RS-232 ROUTINEN ***
. ea5b  :
. ea5b  ad d4 07 lda $07d4    Holt ACIA-Status
. ea5e  29 10    and #$10     Bit 4 gesetzt ?
. ea60  f0 32    beq $ea94    Nein, dann $ea94
. ea62  ad 10 fd lda $fd10    Holt Ein-/Ausgabe-Leitung
. ea65  29 02    and #$02     Bit 2 gesetzt ?
. ea67  f0 2b    beq $ea94    Nein, dann fertig
. ea69  a2 00    ldx #$00
. ea6b  2c d0 07 bit $07d0    Systemzeichen im Ausgaberegister
. ea6e  10 09    bpl $ea79    Nein, dann fertig
. ea70  ad cf 07 lda $07cf    Holt zu sendendes Systemzeichen
. ea73  8e d0 07 stx $07d0    Initialisiert Ausgaberegister
. ea76  4c 89 ea jmp $ea89    Springt nach $ea89

. ea79  2c ce 07 bit $07ce    Datenbyte im Ausgaberegister ?
```

```
. ea7c  10 16    bpl $ea94    Nein, dann fertig
. ea7e  2c d6 07 bit $07d6    Pausen-Flag gesetzt ?
. ea81  30 11    bmi $ea94    Ja, dann fertig
. ea83  ad cd 07 lda $07cd    Holt Datenbyte
. ea86  8e ce 07 stx $07ce    Initialisiert Ausgaberegister
. ea89  8d 00 fd sta $fd00    Schickt Datenbyte an RS-232
. ea8c  ad d4 07 lda $07d4    Holt ACIA-Status
. ea8f  29 ef    and #$ef     Löscht Bit 4
. ea91  8d d4 07 sta $07d4    Schreibt ACIA-Status
. ea94  60       rts          ENDE

. ea95  ad d4 07 lda $07d4    Holt ACIA-Status
. ea98  29 08    and #$08     Bit 3 gesetzt ?
. ea9a  f0 54    beq $eaf0    Nein, dann $eaf0
. ea9c  ad d4 07 lda $07d4    Löscht
. ea9f  29 f7    and #$f7     ... Bit 3
. eaa1  8d d4 07 sta $07d4    ... im ACIA-Status
. eaa4  ad 00 fd lda $fd00    Holt Datenbyte von RS-232
. eaa7  f0 19    beq $eac2    Datenbyte gleich 0 ?, dann $eac2
. eaa9  8d d5 07 sta $07d5    Speichert Datenbyte
. eaac  c5 fc    cmp $fc      Datenbyte gleich $ff (von Open)
. eaae  d0 07    bne $eab7    Nein, dann $eab7
. eab0  a9 00    lda #$00     Initialisiert
. eab2  8d d6 07 sta $07d6    ... Pausen-Flag
. eab5  f0 39    beq $eaf0    Springt immer nach $eaf0

. eab7  c5 fd    cmp $fd      Datenbyte gleich $ff (von Open)
. eab9  d0 07    bne $eac2    Nein, dann $eac2
. eabb  a9 ff    lda #$ff     Setzt
. eabd  8d d6 07 sta $07d6    ... Pausen-Flag
. eac0  d0 2e    bne $eaf0    Springt immer nach $eaf0

. eac2  ad d3 07 lda $07d3    Holt Anzahl der Zeichen im Eingabepuffer
. eac5  c9 3f    cmp #$3f     Schon 64 Zeichen gelesen ?
. eac7  f0 27    beq $eaf0    Ja, dann $eaf0
. eac9  c9 38    cmp #$38     Schon 57 Zeichen gelesen ?
. eacb  d0 0f    bne $eadc    Nein, dann $eadc
. eacd  a5 fd    lda $fd      Holt zu sendendes Zeichen für X-Off
. eacf  f0 0b    beq $eadc    Zeichen gleich 0 ?, dann $eadc
. ead1  8d cf 07 sta $07cf    Schreibt zu sendendes System-Zeichen
. ead4  a9 ff    lda #$ff     Setzt Flag für
. ead6  8d d0 07 sta $07d0    ... Systemzeichen im Ausgaberegister
. ead9  8d d7 07 sta $07d7    ... und ferngesteuerte Pause
. eadc  ae d1 07 ldx $07d1    Holt Pufferzeiger
. eadf  e8       inx          Erhöht Pufferzeiger
. eae0  8a       txa          Und-Verknüpfung
. eae1  29 3f    and #$3f     ... mit $3f
. eae3  8d d1 07 sta $07d1    Pufferzeiger abspeichern
. eae6  aa       tax
. eae7  ad d5 07 lda $07d5    Holt Datenbyte
. eaea  9d f7 03 sta $03f7,x  Schreibt Datenbyte in Puffer
. eaed  ee d3 07 inc $07d3    Erhöht Zeichen-Zähler im Puffer um 1
```

```
. eaf0  60       rts          ENDE

. eaf1  :
. eaf1  *** RS-232 EINGABE ***
. eaf1  :
. eaf1  ad d3 07 lda $07d3    Eingabepuffer leer ?
. eaf4  f0 34    beq $eb2a    Ja, dann $eb2a
. eaf6  08       php
. eaf7  78       sei          Kein Interrupt zugelassen
. eaf8  ae d2 07 ldx $07d2    Holt Zeiger auf Ende des Eingabepuffers
. eafb  e8       inx          Erhöht Zeiger um 1
. eafc  8a       txa          UND-Verknüpfung
. eafd  29 3f    and #$3f     ... mit $3f
. eaff  8d d2 07 sta $07d2    Zeiger zurückschreiben
. eb02  28       plp
. eb03  aa       tax
. eb04  bd f7 03 lda $03f7,x  Holt Byte aus Puffer
. eb07  48       pha          Byte auf Stack retten
. eb08  ce d3 07 dec $07d3    Erniedrigt Anzahl der Zeichen im Puffer
. eb0b  ad d3 07 lda $07d3    Holt Anzahl der Bytes im Eingabepuffer
. eb0e  c9 08    cmp #$08     Noch 8 Byte im Puffer ?
. eb10  d0 19    bne $eb2b    Nein, dann $eb2b
. eb12  2c d7 07 bit $07d7    Ferngesteuerte Pause ?
. eb15  10 14    bpl $eb2b    Nein, dann $eb2b
. eb17  a5 fc    lda $fc      Holt zu sendendes X-ON-Zeichen
. eb19  f0 10    beq $eb2b    Zeichen gleich Null ?, dann $eb2b
. eb1b  8d cf 07 sta $07cf    Schreibt Zeichen in Ausgabepuffer
. eb1e  38       sec          Setzt Flag für
. eb1f  6e d0 07 ror $07d0    ... Systemzeichen im Ausgaberegister
. eb22  4e d7 07 lsr $07d7    Initialisierter Flag für ferngesteuerte Pause
. eb25  2c d8 07 bit $07d8    ACIA offen ?
. eb28  10 0b    bpl $eb35    Nein, dann $eb35
. eb2a  48       pha
. eb2b  ad d4 07 lda $07d4    Holt ACIA-Status
. eb2e  29 4f    and #$4f     Berechnet
. eb30  49 40    eor #$40     ... Status
. eb32  85 90    sta $90      Schreibt Status (ST)
. eb34  68       pla
. eb35  18       clc
. eb36  60       rts          ENDE

. eb37  :
. eb37  *** RS-232 AUSGABE ***
. eb37  :
. eb37  2c ce 07 bit $07ce    Ausgaberegister frei ?
. eb3a  30 fb    bmi $eb37    Nein, dann warten
. eb3c  8d cd 07 sta $07cd    Schreibt Datenbyte in Ausgaberegister
. eb3f  38       sec          Setzt Flag für
. eb40  6e ce 07 ror $07ce    ... Zeichen im Ausgaberegister
. eb43  4c 2a eb jmp $eb2a    Abschluß

. eb46  :
```

```
. eb46  *** RS-232 ARBEITSBEREICH INITIALISIEREN ***
. eb46  :
. eb46  a9 00    lda #$00      Löscht
. eb48  a2 0b    ldx #$0b      ... Bereich
. eb4a  9d cd 07 sta $07cd,x   ... $07c3-$07cd
. eb4d  ca       dex
. eb4e  10 fa    bpl $eb4a
. eb50  8d 01 fd sta $fd01     Löscht RS-232 Statusregister
. eb53  85 fc    sta $fc       Löscht X-On-Code
. eb55  85 fd    sta $fd       Löscht X-Off-Code
. eb57  60       rts           ENDE

. eb58  :
. eb58  *** EIN-/AUSGABE-MELDUNGEN ***
. eb58  :
. eb58  0d       .byte $0d     CR-Code
. eb59           .byte 'i/o error #'
. eb64  0d       .byte $0d     CR-Code
. eb65           .byte 'searching for
. eb73  0d       .byte $0d     CR-Code
. eb74           .byte 'press play on tapE'
. eb86           .byte 'press record & play on tapE'
. eba1  0d       .byte $0d     CR-Code
. eba2           .byte 'loadinG'
. eba9  0d       .byte $0d     CR-Code
. ebaa           .byte 'saving '
. ebb1  0d       .byte $0d     CR-Code
. ebb2           .byte 'verifyinG'
. ebbb  0d       .byte $0d     CR-Code
. ebbc           .byte 'found '
. ebc2  0d       .byte $0d     CR-Code
. ebc3           .byte 'ok'
. ebc5  8d       .byte $0d     CR-Code

. ebc6  :
. ebc6  *** EIN-/AUSGABE-MELDUNGEN AUSGEBEN ***
. ebc6  :
. ebc6  24 9a    bit $9a       Meldungen zugelassen ?
. ebc8  10 0d    bpl $ebd7     Nein, dann $ebd7
. ebca  b9 58 eb lda $eb58,y   Holt Zeichen aus Meldung
. ebcd  08       php
. ebce  29 7f    and #$7f      Isoliert Bit 0-6
. ebd0  20 d2 ff jsr $ffd2     (BSOUT) Gibt Zeichen aus
. ebd3  c8       iny           Erhöht Offset in Meldung
. ebd4  28       plp           Meldung zu Ende ?
. ebd5  10 f3    bpl $ebca     Nein, dann $ebca
. ebd7  18       clc
. ebd8  60       rts           ENDE

. ebd9  :
. ebd9  *** GETIN ***
. ebd9  :
```

```
. ebd9  a5 98    lda $98      Eingabegerät gleich Tastatur ?
. ebdb  dØ 1a    bne $ebf7    Nein, dann $ebf7
. ebdd  a5 ef    lda $ef      Tastaturpuffer-Index
. ebdf  Ød 5d Ø5 ora $Ø55d    ... und Funktionstasten-Index gleich Null ?
. ebe2  fØ 3e    beq $ec22    Ja, dann $ec22
. ebe4  78       sei          Kein Interrupt zugelassen
. ebe5  4c c1 d8 jmp $d8c1    Holt Code aus Tastaturpuffer

. ebe8  :
. ebe8  *** BASIN ***
. ebe8  :
. ebe8  a5 98    lda $98      Eingabekanal gleich Tastatur ?
. ebea  dØ Øb    bne $ebf7    Nein, dann $ebf7
. ebec  a5 ca    lda $ca      Holt aktuelle Cursorspalte
. ebee  85 c5    sta $c5      Schreibt letzten Spaltenwert
. ebfØ  a5 cd    lda $cd      Holt aktuelle Cursorzeile
. ebf2  85 c4    sta $c4      Schreibt letzten Zeilenwert
. ebf4  4c 65 d9 jmp $d965    Holt Zeichen vom Bildschirm

. ebf7  c9 Ø3    cmp #$Ø3     Eingabegerät gleich Bildschirm ?
. ebf9  dØ 1f    bne $ec1a    Nein, dann $ec1a
. ebfb  Ø5 c7    ora $c7      Setzt
. ebfd  85 c7    sta $c7      ... Übernahme-Flag
. ebff  ad e8 Ø7 lda $Ø7e8    Holt rechten Bildschirmrand
. ecØ2  85 c3    sta $c3      Schreibt Ende der log. Zeile für Eingabe
. ecØ4  4c 65 d9 jmp $d965    Holt Zeichen vom Bildschirm

. ecØ7  2Ø ba fb jsr $fbba    Rettet X- und Y-Register
. ecØa  c9 Ø1    cmp #$Ø1     Eingabegerät gleich Kassette ?
. ecØc  dØ Ø6    bne $ec14    Nein, dann $ec14
. ecØe  2Ø 24 ec jsr $ec24    Holt Byte von Kassette
. ec11  4c c4 fb jmp $fbc4    Stellt X- und Y-Register wieder her
. ec14  2Ø f1 ea jsr $eaf1    Holt Byte von RS-232
. ec17  4c c4 fb jmp $fbc4    Stellt X- und Y-Register wieder her

. ec1a  9Ø eb    bcc $ecØ7    Eingabe nicht vom Bus ?, dann $ecØ7
. ec1c  a5 9Ø    lda $9Ø      Status OK ?
. ec1e  fØ 6b    beq $ec8b    Ja, dann $ec8b
. ec2Ø  a9 Ød    lda #$Ød     Holt CR-Code
. ec22  18       clc
. ec23  6Ø       rts          ENDE

. ec24  :
. ec24  *** BYTE VON KASSETTE HOLEN ***
. ec24  :
. ec24  ac 39 Ø5 ldy $Ø539    Holt Pufferzeiger
. ec27  cØ bf    cpy #$bf     Schon Pufferende erreicht ?
. ec29  9Ø Ø6    bcc $ec31    Nein, dann $ec31
. ec2b  2Ø d3 e8 jsr $e8d3    Block in Puffer lesen
. ec2e  9Ø f4    bcc $ec24    Kein Fehler ?, dann $ec24
. ec3Ø  6Ø       rts          ENDE
```

```
. ec31  ac 39 Ø5 ldy $Ø539     Holt Pufferzeiger
. ec34  b1 b6    lda ($b6),y   Holt Byte aus Puffer
. ec36  48       pha           Datenbyt auf Stack retten
. ec37  c8       iny           Erhöht Offset im Puffer um 1
. ec38  cØ bf    cpy #$bf      Pufferende erreicht ?
. ec3a  bØ Ø9    bcs $ec45     Ja, dann $ec45
. ec3c  b1 b6    lda ($b6),y   Holt nächstes Byte aus Puffer
. ec3e  dØ Ø5    bne $ec45     Nicht Ø-Code ?, dann $ec45
. ec4Ø  a9 4Ø    lda #$4Ø      Odert EOI-Bit
. ec42  2Ø 1e f4 jsr $f41e     ... in Status
. ec45  ee 39 Ø5 inc $Ø539     Erhöht Pufferzeiger um 1
. ec48  68       pla           Holt Datenbyte vom Stack
. ec49  18       clc
. ec4a  6Ø       rts           ENDE

. ec4b  :
. ec4b  *** BSOUT ***
. ec4b  :
. ec4b  48       pha           Datenbyte auf Stack retten
. ec4c  a5 99    lda $99       Holt Ausgabekanal
. ec4e  c9 Ø3    cmp #$Ø3      Ausgabegerät gleich Bildschirm ?
. ec5Ø  dØ Ø4    bne $ec56     Nein, dann $ec56
. ec52  68       pla           Holt Datenbyte vom Stack
. ec53  4c 49 dc jmp $dc49     Zeichen auf Bildschirm ausgeben

. ec56  9Ø Ø4    bcc $ec5c     A-Gerät = Tast., Kass., RS-232 ?, dann $ec5c
. ec58  68       pla           Holt Datenbyte vom Stack
. ec59  4c df ec jmp $ecdf     (CIOUT) Ausgabe auf Bus

. ec5c  2Ø b7 fb jsr $fbb7     Rettet Akku, X- und Y-Register
. ec5f  c9 Ø1    cmp #$Ø1      Ausgabegerät gleich Kassette ?
. ec61  dØ 21    bne $ec84     Nein, dann $ec84
. ec63  ac 39 Ø5 ldy $Ø539     Holt Pufferzeiger
. ec66  cØ bf    cpy #$bf      Puffer voll ?
. ec68  9Ø Øb    bcc $ec75     Nein, dann $ec75
. ec6a  2Ø 35 e5 jsr $e535     Schreibt Puffer auf Band
. ec6d  bØ Øf    bcs $ec7e     Fehler ?, dann $ec7e
. ec6f  a9 Ø2    lda #$Ø2      Schreibt
. ec71  85 f8    sta $f8       ... File-Typ 2 (für Daten)
. ec73  aØ ØØ    ldy #$ØØ      Initialisiert Offset im Puffer
. ec75  68       pla           Holt Datenbyte vom Stack
. ec76  91 b6    sta ($b6),y   Schreibt Datenbyte in Puffer
. ec78  c8       iny           Erhöht Offset um 1
. ec79  8c 39 Ø5 sty $Ø539     Schreibt neuen Pufferzeiger
. ec7c  9Ø Øa    bcc $ec88     Springt immer nach $ec88

. ec7e  68       pla           Korrigiert Stack
. ec7f  a9 ØØ    lda #$ØØ
. ec81  4c c4 fb jmp $fbc4     Stellt X- und Y-Register wieder her

. ec84  68       pla           Holt Datenbyte vom Stack
. ec85  2Ø 37 eb jsr $eb37     Gibt Zeichen auf RS-232 aus
```

```
. ec88  4c c1 fb jmp $fbc1    Stellt Akku, X- und Y-Register wieder her

. ec8b  :
. ec8b  *** ACPTR ***
. ec8b  :
. ec8b  86 ba    stx $ba      X-Reg. zwischenspeichern
. ec8d  24 f9    bit $f9      IEC-Bus ?
. ec8f  70 05    bvs $ec96    Ja, dann $ec96
. ec91  a6 ba    ldx $ba      Holt X-Reg.
. ec93  4c 52 e2 jmp $e252    Holt Byte vom seriellen Bus

     IEC-Bus

. ec96  a5 f9    lda $f9      Holt IEC-BUS-Flag
. ec98  29 30    and #$30     Offset für Geräteadresse
. ec9a  aa       tax          Offset 48 für Gerät #8, Offset 0 für Gerät #9
. ec9b  a9 84    lda #$84
. ec9d  9d c0 fe sta $fec0,x
. eca0  bd c2 fe lda $fec2,x
. eca3  30 fb    bmi $eca0
. eca5  a9 00    lda #$00
. eca7  9d c3 fe sta $fec3,x
. ecaa  9d c2 fe sta $fec2,x
. ecad  bd c2 fe lda $fec2,x
. ecb0  10 fb    bpl $ecad
. ecb2  bd c1 fe lda $fec1,x  Holt IEC-Status
. ecb5  29 03    and #$03     Isoliert Bit 0 und Bit 1
. ecb7  c9 03    cmp #$03     Bit 0 und Bit 1 gesetzt ?
. ecb9  d0 02    bne $ecbd    Nein, dann $ecbd
. ecbb  a9 40    lda #$40     Holt EOI-Bit
. ecbd  20 1e f4 jsr $f41e    In Status (ST) odern
. ecc0  bd c0 fe lda $fec0,x
. ecc3  48       pha
. ecc4  a9 40    lda #$40
. ecc6  9d c2 fe sta $fec2,x
. ecc9  bd c2 fe lda $fec2,x
. eccc  30 fb    bmi $ecc9
. ecce  a9 ff    lda #$ff
. ecd0  9d c3 fe sta $fec3,x
. ecd3  a9 00    lda #$00
. ecd5  9d c0 fe sta $fec0,x
. ecd8  9d c2 fe sta $fec2,x
. ecdb  4c d4 ed jmp $edd4    IEC-Abschluß
. ecde  ea       nop

. ecdf  :
. ecdf  *** CIOUT ***
. ecdf  :
. ecdf  24 f9    bit $f9      IEC-Bus ?
. ece1  30 03    bmi $ece6    Ja, dann $ece6
. ece3  4c 1d e2 jmp $e21d    Byte auf seriellen Bus ausgeben
```

```
. ece6  48       pha          Datenbyte auf Stack retten
. ece7  8d e8 Ø5 sta $Ø5e8    Schreibt Byte nach Flag für Kennedy-R/W
. ecea  a9 83    lda #$83
. ecec  86 ba    stx $ba      Speichert X-Reg.
. ecee  48       pha
. ecef  a5 f9    lda $f9      Holt IEC-Bus-Flag
. ecf1  29 3Ø    and #$3Ø     Offset für Geräteadresse
. ecf3  aa       tax          Offset 48 für Gerät #8, Offset Ø für Gerät #9
. ecf4  68       pla
. ecf5  9d cØ fe sta $fecØ,x
. ecf8  bd c2 fe lda $fec2,x
. ecfb  3Ø fb    bmi $ecf8
. ecfd  ad e8 Ø5 lda $Ø5e8    Holt Datenbyte
. edØØ  9d cØ fe sta $fecØ,x
. edØ3  a9 ØØ    lda #$ØØ
. edØ5  9d c2 fe sta $fec2,x
. edØ8  bd c2 fe lda $fec2,x
. edØb  1Ø fb    bpl $edØ8
. edØd  bd c1 fe lda $fec1,x  Holt IEC-Status
. ed1Ø  29 Ø3    and #$Ø3     Isoliert Bit Ø und Bit 1
. ed12  2Ø 1e f4 jsr $f41e    In Status (ST) odern
. ed15  4c db ed jmp $eddb    IEC-Abschluß

. ed18  :
. ed18  *** CHKIN ***
. ed18  :
. ed18  2Ø e8 ee jsr $eee8    Sucht File in Tabelle
. ed1b  fØ Ø3    beq $ed2Ø    File gefunden ?, dann $ed2Ø
. ed1d  4c 79 f2 jmp $f279    'I/O ERROR #3' ausgeben (file not open)

. ed2Ø  2Ø f8 ee jsr $eef8    Übernimmt File-Daten aus Tabelle
. ed23  fØ 11    beq $ed36    Eingabegerät gleich Tastatur ?, dann $ed36
. ed25  c9 Ø3    cmp #$Ø3     Eingabegerät gleich Bildschirm ?
. ed27  fØ Ød    beq $ed36    Ja, dann $ed36
. ed29  bØ Øf    bcs $ed3a    Eingabegerät gleich Bus ?, dann $ed3a
. ed2b  c9 Ø2    cmp #$Ø2     Eingabegerät gleich Kassette ?
. ed2d  dØ 28    bne $ed57    Ja, dann $ed57
. ed2f  2Ø 25 eb jsr $eb25    CHKIN von RS-232
. ed32  bØ Ø5    bcs $ed39    Fehler ?, dann $ed39
. ed34  a5 ae    lda $ae      Holt Geräteadresse
. ed36  85 98    sta $98      Schreibt Eingabekanal
. ed38  18       clc
. ed39  6Ø       rts          ENDE

     CHKIN VOM BUS

. ed3a  aa       tax
. ed3b  2Ø fa ed jsr $edfa    (TALK)
. ed3e  24 9Ø    bit $9Ø      Überprüft Status
```

```
. ed4Ø  3Ø 12     bmi $ed54    Bit 7 gesetzt ?, dann Fehler
. ed42  a5 ad     lda $ad      Holt Sekundäradresse
. ed44  1Ø Ø6     bpl $ed4c    Sekundäradresse kleiner 128 ?, dann $ed4c
. ed46  2Ø 13 ee  jsr $ee13    Gibt 'ATN' aus
. ed49  4c 4f ed  jmp $ed4f    Springt nach $ed4f

. ed4c  2Ø 1a ee  jsr $ee1a    (TKSA) Übergibt Sekundäradresse für TALK
. ed4f  8a        txa
. ed5Ø  24 9Ø     bit $9Ø      Status OK ?
. ed52  1Ø e2     bpl $ed36    Ja, dann $ed36
. ed54  4c 7f f2  jmp $f27f    'I/O ERROR #5' ausgeben (device not present)

      KASSETTE

. ed57  a6 ad     ldx $ad      Sekundäradresse
. ed59  eØ 6Ø     cpx #$6Ø     ... gleich 96 ($6Ø) ?
. ed5b  fØ d9     beq $ed36    Ja, dann $ed36
. ed5d  4c 82 f2  jmp $f282    'I/O ERROR #6' ausgeben (not input file)

. ed6Ø  :
. ed6Ø  *** CKOUT ***
. ed6Ø  :
. ed6Ø  2Ø e8 ee  jsr $eee8    Sucht File in Tabelle
. ed63  fØ Ø3     beq $ed68    File gefunden ?, dann $ed68
. ed65  4c 79 f2  jmp $f279    'I/O ERROR #3' ausgeben (file not open)

. ed68  2Ø f8 ee  jsr $eef8    Übernimmt File-Daten aus Tabelle
. ed6b  dØ Ø3     bne $ed7Ø    Ausgabegerät nicht Tastatur ?, dann $ed7Ø
. ed6d  4c 85 f2  jmp $f285    'I/O ERROR #7' ausgeben (not output file)

. ed7Ø  c9 Ø3     cmp #$Ø3     Ausgabegerät gleich Bildschirm ?
. ed72  fØ Ød     beq $ed81    Ja, dann $ed81
. ed74  bØ Øf     bcs $ed85    Ausgabegerät gleich Bus ?, dann $ed85
. ed76  c9 Ø2     cmp #$Ø2     Ausgabegerät gleich Datasette ?
. ed78  dØ 27     bne $eda1    Ja, dann $eda1
. ed7a  2Ø 25 eb  jsr $eb25    CHKOUT RS-232
. ed7d  bØ Ø5     bcs $ed84    Fehler ?, dann $ed84
. ed7f  a5 ae     lda $ae      Holt Geräteadresse
. ed81  85 99     sta $99      Schreibt Ausgabekanal
. ed83  18        clc
. ed84  6Ø        rts          ENDE

      BUS

. ed85  aa        tax
. ed86  2Ø 2c ee  jsr $ee2c    (LISTEN)
. ed89  24 9Ø     bit $9Ø      Überprüft Status
. ed8b  3Ø 11     bmi $ed9e    Bit 7 gesetzt ?, dann Fehler
. ed8d  a5 ad     lda $ad      Holt Sekundäradresse
. ed8f  1Ø Ø5     bpl $ed96    Sekundäradresse kleiner 128 ?, dann $ed96
. ed91  2Ø 45 ee  jsr $ee45    Gibt 'ATN' aus
. ed94  dØ Ø3     bne $ed99    Springt immer nach $ed99
```

```
. ed96  20 4d ee jsr $ee4d    (SECOND)  Übergibt Sekundäradresse für LISTEN
. ed99  8a       txa
. ed9a  24 90    bit $90      Status OK ?
. ed9c  10 e3    bpl $ed81    Ja, dann $ed81
. ed9e  4c 7f f2 jmp $f27f    'I/O ERROR #5' ausgeben (device not present)

      KASSETTE

. eda1  a6 ad    ldx $ad      Sekundäradresse
. eda3  e0 60    cpx #$60     ... gleich 96 ($60) ?
. eda5  f0 c6    beq $ed6d    Ja, dann 'I/O ERROR #7' ausgeben
. eda7  d0 d8    bne $ed81    Ausgabekanal schreiben und ENDE

. eda9  :
. eda9  *** IEC-BUS PRÜFEN ***
. eda9  :
. eda9  48       pha          Akku und
. edaa  86 ba    stx $ba      ... X-Reg. zwischenspeichern
. edac  a2 30    ldx #$30     Holt Offset für Geräteadresse 8
. edae  a5 ae    lda $ae      Holt Geräteadresse
. edb0  c9 08    cmp #$08     Geräteadresse
. edb2  f0 06    beq $edba    ... 8 oder
. edb4  c9 09    cmp #$09     ... 9 ?
. edb6  d0 17    bne $edcf    Nein, dann edcf
. edb8  a2 00    ldx #$00     Holt Offset für Geräteadresse 9
. edba  a9 55    lda #$55     Ist Adresse $fec0 (für Geräteadresse 9) bzw.
. edbc  9d c0 fe sta $fec0,x  ... Adresse $fef0 (für Geräteadresse 8)
. edbf  5d c0 fe eor $fec0,x  ... eine RAM-Adresse ?
. edc2  d0 0b    bne $edcf    Nein, dann kein IEC-Bus
. edc4  bd c1 fe lda $fec1,x  Ist in der darauffolgenden
. edc7  29 02    and #$02     ... Adresse Bit 1 gesetzt ?
. edc9  d0 04    bne $edcf    Ja, dann kein IEC-Bus
. edcb  86 f9    stx $f9      Speichert Offset für Geräteadresse
. edcd  18       clc          Setzt Flag für 'IEC-Bus'

. edce  24       .byte $24

. edcf  38       sec          Setzt Flag für 'kein IEC-Bus'
. edd0  a6 ba    ldx $ba      Stellt X-Reg.
. edd2  68       pla          ... und Akku wieder her
. edd3  60       rts          ENDE

. edd4  :
. edd4  *** IEC-ABSCHLUSS ***
. edd4  :

      EINGABE

. edd4  bd c2 fe lda $fec2,x
. edd7  10 fb    bpl $edd4
. edd9  30 05    bmi $ede0
```

```
      AUSGABE

. eddb  a9 ØØ    lda #$ØØ
. eddd  9d cØ fe sta $fecØ,x
. edeØ  a9 4Ø    lda #$4Ø
. ede2  9d c2 fe sta $fec2,x
. ede5  a6 ba    ldx $ba       Stellt X-Reg.
. ede7  68       pla           ... und Akku wieder her
. ede8  18       clc
. ede9  6Ø       rts           ENDE

. edea  :
. edea  *** TEILE VON IOINIT ***
. edea  :
. edea  8d f2 fe sta $fef2     (Akku = $4Ø)
. eded  8d c5 fe sta $fec5
. edfØ  8d c2 fe sta $fec2
. edf3  ca       dex           (X-Reg. = $FF)
. edf4  8e c3 fe stx $fec3
. edf7  4c ea cf jmp $cfea     Schaltet Tongenerator ab und Ende

. edfa  :
. edfa  *** TALK ***
. edfa  :
. edfa  2Ø a9 ed jsr $eda9     IEC-Bus ?
. edfd  9Ø Ø3    bcc $eeØ2     Ja, dann $eeØ2
. edff  4c 53 e1 jmp $e153     TALK am seriellen Bus

. eeØ2  48       pha
. eeØ3  a9 4Ø    lda #$4Ø      Schreibt $4Ø
. eeØ5  8d e8 Ø5 sta $Ø5e8     ... ins Ausgaberegister
. eeØ8  a5 f9    lda $f9       Setzt
. eeØa  Ø9 4Ø    ora #$4Ø      ... TALK-Bit
. eeØc  85 f9    sta $f9       ... im IEC-Bus-Flag
. eeØe  a9 81    lda #$81
. ee1Ø  4c ec ec jmp $ecec     Gibt TALK aus

. ee13  :
. ee13  *** TKATN ***
. ee13  :
. ee13  24 f9    bit $f9       IEC-Bus ?
. ee15  7Ø 35    bvs $ee4c     Ja, dann $ee4c
. ee17  4c Øc e2 jmp $e2Øc     Gibt 'ATN' aus

. ee1a  :
. ee1a  *** TKSA ***
. ee1a  :
. ee1a  24 f9    bit $f9       IEC-Bus ?
. ee1c  7Ø Ø3    bvs $ee21     Ja, dann $ee21
. ee1e  4c Ø3 e2 jmp $e2Ø3     TALK am seriellen Bus

. ee21  48       pha
```

```
. ee22  a5 ad     lda $ad       Holt Sekundäradresse    (Alte Version)
```

```
. ee22  ea        nop
. ee23  ea        nop
. ee24  8d e8 Ø5  sta $Ø5e8     Schreibt Akku in Ausgaberegister
. ee27  a9 82     lda #$82
. ee29  4c ec ec  jmp $ecec     CIOUT
. ee2c  :
. ee2c  *** LISTEN ***
. ee2c  :
. ee2c  2Ø a9 ed  jsr $eda9     IEC-Bus ?
. ee2f  9Ø Ø3     bcc $ee34     Ja, dann $ee34
. ee31  4c 56 e1  jmp $e156     LISTEN am Seriellen Bus

. ee34  48        pha
. ee35  a9 2Ø     lda #$2Ø      Setzt Listen-Bit
. ee37  8d e8 Ø5  sta $Ø5e8     ... im Ausgaberegister
. ee3a  a5 f9     lda $f9       Setzt
. ee3c  Ø9 8Ø     ora #$8Ø      ... Listen-Bit
. ee3e  85 f9     sta $f9       ... im IEC-Bus-Flag
. ee4Ø  a9 81     lda #$81
. ee42  4c ec ec  jmp $ecec     Gibt 'LISTEN' aus

. ee45  :
. ee45  *** LSTATN ***
. ee45  :
. ee45  24 f9     bit $f9       IEC-Bus ?
. ee47  3Ø Ø3     bmi $ee4c     Ja, dann fertig
. ee49  4c fc e1  jmp $e1fc     Gibt 'ATN' am seriellen Bus aus

. ee4c  6Ø        rts           ENDE

. ee4d  :
. ee4d  *** SECOND ***
. ee4d  :
. ee4d  24 f9     bit $f9       IEC-Bus ?
. ee4f  3Ø Ø3     bmi $ee54     Ja, dann $ee54
. ee51  4c f7 e1  jmp $e1f7     Gibt 'SECOND' am seriellen Bus aus

. ee54  48        pha
. ee55  8d e8 Ø5  sta $Ø5e8     Schreibt Sekundäradresse ins Ausgabereg.
. ee58  a9 82     lda #$82
. ee5a  4c ec ec  jmp $ecec     Gibt Sekundäradresse aus

. ee5d  :
. ee5d  *** CLOSE ***
. ee5d  :
. ee5d  66 ba     ror $ba       Carry-Bit speichern
. ee5f  2Ø ed ee  jsr $eeed     Sucht File in Tabelle
```

```
. ee62  f0 02     beq $ee66     File gefunden ?, dann ee66
. ee64  18        clc
. ee65  60        rts           ENDE

. ee66  20 f8 ee  jsr $eef8     Übernimmt File-Daten aus Tabelle
. ee69  8a        txa           Rettet Geräteadresse
. ee6a  48        pha           ... auf Stack
. ee6b  a5 ae     lda $ae       Holt logische Adresse
. ee6d  f0 5b     beq $eeca     Log. Adresse gleich Tastatur ?, dann $eeca
. ee6f  c9 03     cmp #$03      Logische Adresse gleich Bildschirm ?
. ee71  f0 57     beq $eeca     Ja, dann $eeca
. ee73  b0 40     bcs $eeb5     Log. Adresse gleich Bus ?, dann $eeb5
. ee75  c9 02     cmp #$02      Logische Adresse gleich Datasette ?
. ee77  d0 08     bne $ee81     Ja, dann $ee81
. ee79  08        php
. ee7a  78        sei           Kein Interrupt zugelassen
. ee7b  20 46 eb  jsr $eb46     Initialisiert RS-232-Arbeitsbereich
. ee7e  28        plp
. ee7f  f0 49     beq $eeca     Springt immer nach $eeca

    DATASETTE

. ee81  a5 ad     lda $ad       Holt Sekundäradresse
. ee83  29 0f     and #$0f      Sekundäradresse gleich 0 (für READ) ?
. ee85  f0 43     beq $eeca     Ja, dann $eeca
. ee87  ac 39 05  ldy $0539     Holt Kassettenpuffer-Zeiger
. ee8a  c0 bf     cpy #$bf      Puffer voll ?
. ee8c  90 0e     bcc $ee9c     Nein, dann $ee9c
. ee8e  20 35 e5  jsr $e535     Schreibt Puffer auf Band
. ee91  b0 12     bcs $eea5     Fehler ?, dann $eea5
. ee93  a9 02     lda #$02      Besetzt File-Typ
. ee95  85 f8     sta $f8       ... mit $02 (für Daten)
. ee97  a0 00     ldy #$00      Initialisiert
. ee99  8c 39 05  sty $0539     ... Pufferzeiger
. ee9c  a9 00     lda #$00      Schließt Puffer
. ee9e  91 b6     sta ($b6),y   ... mit $00-Code ab
. eea0  20 35 e5  jsr $e535     Schreibt Puffer auf Band
. eea3  90 04     bcc $eea9     Kein Fehler ?, dann $eea9
. eea5  68        pla
. eea6  a9 00     lda #$00
. eea8  60        rts           ENDE

. eea9  a5 ad     lda $ad       Holt Sekundäradresse
. eeab  c9 62     cmp #$62      Ist 'End Of Tape'-Block verlangt ?
. eead  d0 1b     bne $eeca     Nein, dann $eeca
. eeaf  20 f0 e5  jsr $e5f0     Schreibt 'EOT'-Block
. eeb2  4c ca ee  jmp $eeca     Springt nach $eeca

    BUS

. eeb5  24 ba     bit $ba       Mit 'SEC' aufgerufen ?
. eeb7  10 0e     bpl $eec7     Nein, dann Floppy-Kanal schließen
```

```
. eeb9  a5 ae     lda $ae       Holt Geräteadresse
. eebb  c9 Ø8     cmp #$Ø8      Geräteadresse kleiner 8 ?
. eebd  9Ø Ø8     bcc $eec7     Ja, dann Floppy-Kanal schließen
. eebf  a5 ad     lda $ad       Sekundäradresse
. eec1  29 Øf     and #$Øf      ... gleich
. eec3  c9 Øf     cmp #$Øf      ... 15 ?
. eec5  fØ Ø3     beq $eeca     Ja, dann $eeca
. eec7  2Ø 11 f2  jsr $f211     Schließt Floppy-Kanal

      ENTFERNT FILE-DATEN AUS TABELLE

. eeca  68        pla           Holt Offset in
. eecb  aa        tax           ... Tabelle von Stack
. eecc  c6 97     dec $97       Erniedrigt Index der File-Tabelle
. eece  e4 97     cpx $97       Offset gleich Index ?
. eedØ  fØ 14     beq $eee6     Ja, dann fertig
. eed2  a4 97     ldy $97       Holt erniedrigten Index
. eed4  b9 Ø9 Ø5  lda $Ø5Ø9,y   Verschiebt
. eed7  9d Ø9 Ø5  sta $Ø5Ø9,x   ... logische Adresse
. eeda  b9 13 Ø5  lda $Ø513,y   Verschiebt
. eedd  9d 13 Ø5  sta $Ø513,x   ... Geräteadresse
. eeeØ  b9 1d Ø5  lda $Ø51d,y   Verschiebt
. eee3  9d 1d Ø5  sta $Ø51d,x   ... Sekundäradresse
. eee6  18        clc
. eee7  6Ø        rts           ENDE

. eee8  :
. eee8  *** FILE IN TABELLE SUCHEN ***
. eee8  :
. eee8  a9 ØØ     lda #$ØØ      Initialisiert
. eeea  85 9Ø     sta $9Ø       ... Status (ST)
. eeec  8a        txa           Holt logische Adresse
. eeed  a6 97     ldx $97       Holt Index der File-Tabelle
. eeef  ca        dex           Erniedrigt Index um 1
. eefØ  3Ø 15     bmi $efØ7     Nicht gefunden ?, dann $efØ7
. eef2  dd Ø9 Ø5  cmp $Ø5Ø9,x   Logische Adresse identisch ?
. eef5  dØ f8     bne $eeef     Nein, dann $eeef
. eef7  6Ø        rts           ENDE

. eef8  bd Ø9 Ø5  lda $Ø5Ø9,x   Übergibt
. eefb  85 ac     sta $ac       ... logische Adresse
. eefd  bd 1d Ø5  lda $Ø51d,x   Übergibt
. efØØ  85 ad     sta $ad       ... Sekundäradresse
. efØ2  bd 13 Ø5  lda $Ø513,x   Übergibt
. efØ5  85 ae     sta $ae       ... Geräteadresse
. efØ7  6Ø        rts           ENDE

. efØ8  :
. efØ8  *** CLALL ***
. efØ8  :
. efØ8  a9 ØØ     lda #$ØØ      Initialisiert
. efØa  85 97     sta $97       Index der File-Tabelle
```

```
. ef0c  :
. ef0c  *** CLRCH ***
. ef0c  :
. ef0c  a2 03    ldx #$03     Ausgabekanal
. ef0e  e4 99    cpx $99      ... nicht Bus ?
. ef10  b0 03    bcs $ef15    Ja, dann $ef15
. ef12  20 23 ef jsr $ef23    UNLISTEN
. ef15  e4 98    cpx $98      Eingabekanal
. ef17  b0 03    bcs $ef1c    ... nicht Bus ?, dann $ef1c
. ef19  20 3b ef jsr $ef3b    UNTALK
. ef1c  86 99    stx $99      Ausgabekanal gleich Bildschirm
. ef1e  a9 00    lda #$00     Eingabekanal
. ef20  85 98    sta $98      ... gleich Tastatur
. ef22  60       rts          ENDE

. ef23  :
. ef23  *** UNLISTEN ***
. ef23  :
. ef23  24 f9    bit $f9      IEC-Bus ?
. ef25  30 03    bmi $ef2a    Ja, dann $ef2a
. ef27  4c 3d e2 jmp $e23d    UNLISTEN am seriellen Bus

. ef2a  48       pha
. ef2b  a9 3f    lda #$3f     Löscht Listen-Bit
. ef2d  8d e8 05 sta $05e8    ... im Ausgaberegister
. ef30  a5 f9    lda $f9      Löscht
. ef32  29 7f    and #$7f     ... Listen-Bit
. ef34  85 f9    sta $f9      ... im IEC-Bus-Flag
. ef36  a9 81    lda #$81
. ef38  4c ec ec jmp $ecec    Gibt 'UNLISTEN' aus

. ef3b  :
. ef3b  *** UNTALK ***
. ef3b  :
. ef3b  24 f9    bit $f9      IEC-Bus ?
. ef3d  70 03    bvs $ef42    Ja, dann $ef42
. ef3f  4c 2f e2 jmp $e22f    UNTALK am seriellen Bus

. ef42  48       pha
. ef43  a9 5f    lda #$5f     Löscht TALK-Bit
. ef45  8d e8 05 sta $05e8    ... im Ausgaberegister
. ef48  a5 f9    lda $f9      Löscht
. ef4a  29 bf    and #$bf     ... Talk-Bit
. ef4c  85 f9    sta $f9      ... im IEC-Bus-Flag
. ef4e  a9 81    lda #$81
. ef50  4c ec ec jmp $ecec    Gibt 'UNTALK' aus

. ef53  :
. ef53  *** OPEN ***
. ef53  :
. ef53  a6 ac    ldx $ac      Sucht logische Adresse
. ef55  20 e8 ee jsr $eee8    ... in File-Tabelle
```

```
. ef58  dØ Ø3     bne $ef5d     Nicht gefunden ?, dann $ef5d
. ef5a  4c 76 f2  jmp $f276     'I/O ERROR #2' ausgeben (file open)

. ef5d  a6 97     ldx $97       Ist Index der File-Tabelle
. ef5f  eØ Øa     cpx #$Øa      ... kleiner 1Ø ?
. ef61  9Ø Ø3     bcc $ef66     Ja, dann $ef66
. ef63  4c 73 f2  jmp $f273     'I/O ERROR #1' ausgeben (too many files)

. ef66  e6 97     inc $97       Erhöht Index der File-Tabelle
. ef68  a5 ac     lda $ac       Kopiert logische Adresse
. ef6a  9d Ø9 Ø5  sta $Ø5Ø9,x   ... in File-Tabelle
. ef6d  a5 ad     lda $ad       Setzt Bit 5 und 6
. ef6f  Ø9 6Ø     ora #$6Ø      ... in Sekundäradresse und
. ef71  85 ad     sta $ad       ... kopiert Sekundäradresse
. ef73  9d 1d Ø5  sta $Ø51d,x   ... in File-Tabelle
. ef76  a5 ae     lda $ae       Kopiert Geräteadresse
. ef78  9d 13 Ø5  sta $Ø513,x   ... in File-Tabelle
. ef7b  fØ Ø9     beq $ef86     Geräteadresse gleich Tastatur ?, dann $ef86
. ef7d  c9 Ø3     cmp #$Ø3      Geräteadresse gleich Bildschirm ?
. ef7f  fØ Ø5     beq $ef86     Ja, dann $ef86
. ef81  9Ø Ø5     bcc $ef88     Adr. gleich Datasette o. RS-232 ?, dann $ef88
. ef83  2Ø Ø5 fØ  jsr $fØØ5    OPEN Bus
. ef86  18        clc           Setzt OK-Flag
. ef87  6Ø        rts           ENDE

. ef88  c9 Ø2     cmp #$Ø2      Adresse gleich 2 für Datasette ?
. ef8a  dØ 2c     bne $efb8     Ja, dann OPEN Datasette

. ef8c  :
. ef8c  *** OPEN RS-232 ***
. ef8c  :
. ef8c  2Ø 46 eb  jsr $eb46     Löscht RS-232-Arbeitsbereich
. ef8f  aa        tax           Akku gleich Ø
. ef9Ø  e8        inx           Schreibt
. ef91  fØ Øb     beq $ef9e     ... $ØØ - $ff
. ef93  8e Ø3 fd  stx $fdØ3     ... 6551 Kontroll-Register
. ef96  ec Ø3 fd  cpx $fdØ3     Register wieder auslesbar ?
. ef99  fØ f5     beq $ef9Ø     Ja, dann RS-232-Gerät angeschlossen
. ef9b  4c 7f f2  jmp $f27f     'I/O ERROR #5' ausgeben (device not present)

. ef9e  38        sec           Setzt Open-Flag
. ef9f  6e d8 Ø7  ror $Ø7d8     ... für RS-232
. efa2  a9 af     lda #$af      Schreibt Adresse des Filenamens
. efa4  8d df Ø7  sta $Ø7df     ... in Routine für indirekten RAM-Zugriff
. efa7  aØ ØØ     ldy #$ØØ      Offset auf 1. Zeichen setzen
. efa9  2Ø d9 Ø7  jsr $Ø7d9     Holt 1. Zeichen aus Namen (LDA($af),Y)
. efac  8d Ø3 fd  sta $fdØ3     Schreibt Zeichen ins 6551-Kontroll-Register
. efaf  c8        iny           Offset auf 2. Zeichen setzen
. efbØ  2Ø d9 Ø7  jsr $Ø7d9     Holt 2. Zeichen aus Namen (LDA($af),Y)
. efb3  8d Ø2 fd  sta $fdØ2     Schreibt 2. Zeichen ins 6551-Kommando-Register
. efb6  18        clc
. efb7  6Ø        rts           ENDE
```

```
. efb8  :
. efb8  *** OPEN DATASETTE ***
. efb8  :
. efb8  a5 ad     lda $ad       Sekundäradresse
. efba  29 Ø f    and #$Øf      ... gleich Null ?
. efbc  dØ 2c     bne $efea     Nein, dann SCHREIBEN

      LESEN

. efbe  2Ø 1b e3  jsr $e31b     PLAY-Taste prüfen
. efc1  bØ 26     bcs $efe9     STOP-Taste gedrückt ?, dann $efe9
. efc3  2Ø 6Ø f1  jsr $f16Ø     Gibt 'SEARCHIN FOR ...' aus
. efc6  a5 ab     lda $ab       Holt Länge des Filenamens
. efc8  fØ Øa     beq $efd4     Länge gleich Null ?, dann $efd4
. efca  2Ø 21 ea  jsr $ea21     Sucht HEADER-Block auf Band
. efcd  9Ø 1Ø     bcc $efdf     Gefunden ?, dann $efdf
. efcf  fØ 18     beq $efe9     Endemarke ?, dann fertig
. efd1  4c 7c f2  jmp $f27c     'I/O ERROR #4' ausgeben (file not found)

. efd4  2Ø cc e9  jsr $e9cc     Fileheader lesen
. efd7  fØ 1Ø     beq $efe9     Endemarke ?, dann fertig
. efd9  bØ f6     bcs $efd1     Kein Block gefunden ?, dann $efd1
. efdb  c9 Ø5     cmp #$Ø5      EOT-Block gefunden ?
. efdd  fØ f2     beq $efd1     Ja, dann $efd1
. efdf  aØ bf     ldy #$bf      Initialisiert
. efe1  8c 39 Ø5  sty $Ø539     ... Pufferzeiger
. efe4  a9 Ø2     lda #$Ø2      Schreibt File-Typ
. efe6  85 f8     sta $f8       ... 2 (für Datenblock)
. efe8  18        clc
. efe9  6Ø        rts           ENDE

      SCHREIBEN

. efea  2Ø 19 e3  jsr $e319     Prüft PLAY-Taste
. efed  bØ fa     bcs $efe9     STOP-Taste gedrückt ?, dann $efe9
. efef  a9 Ø4     lda #$Ø4      Schreibt File-Typ
. eff1  85 f8     sta $f8       ... 4 (für Headerblock)
. eff3  2Ø 6c e5  jsr $e56c     Schreibt Fileheader auf Band
. eff6  bØ Øc     bcs $fØØ4     Fehler ?, dann $fØØ4
. eff8  a9 Ø2     lda #$Ø2      Schreibt File-Typ
. effa  85 f8     sta $f8       ... 2 (für Datenblock)
. effc  aØ ØØ     ldy #$ØØ      Initialisiert
. effe  8c 39 Ø5  sty $Ø539     ... Puffer-
. fØØ1  8c 37 Ø5  sty $Ø537     ... Zeiger
. fØØ4  6Ø        rts           ENDE

. fØØ5  :
. fØØ5  *** OPEN BUS ***
. fØØ5  :
. fØØ5  a5 ad     lda $ad       Holt Sekundäradresse
. fØØ7  3Ø df     bmi $efe8     Bit 7 gesetzt ?, dann Kanal bereits offen
```

```
. f009  a4 ab     ldy $ab      Holt Länge des Filenamens
. f00b  f0 db     beq $efe8    Kein Filename ?, dann fertig
. f00d  a9 00     lda #$00     Initialisiert
. f00f  85 90     sta $90      ... Status (ST)
. f011  a5 ae     lda $ae      Holt Geräteadresse
. f013  20 2c ee  jsr $ee2c    LISTEN
. f016  24 90     bit $90      Bit 7 im Status gesetzt ?
. f018  30 0b     bmi $f025    Ja, dann $f025
. f01a  a5 ad     lda $ad      Holt Sekundäradresse
. f01c  09 f0     ora #$f0     ... und isoliert unteres Halbbyte
. f01e  20 4d ee  jsr $ee4d    SECOND
. f021  a5 90     lda $90      Status OK ?
. f023  10 05     bpl $f02a    Ja, dann $f02a
. f025  68        pla
. f026  68        pla
. f027  4c 7f f2  jmp $f27f    'I/O ERROR #5' ausgeben (device not present)

. f02a  a5 ab     lda $ab      Holt Länge des Filenamens
. f02c  f0 12     beq $f040    Kein Filename ?, dann $f040
. f02e  a0 00     ldy #$00     Initialisiert Offset im Filenamen
. f030  a9 af     lda #$af     Schreibt Adresse des Filenamens
. f032  8d df 07  sta $07df    ... in Routine für inirekten RAM-Zugriff
. f035  20 d9 07  jsr $07d9    Holt Zeichen aus Filenamen (LDA($af),Y)
. f038  20 df ec  jsr $ecdf    (CIOUT) Gibt Zeichen auf Bus aus
. f03b  c8        iny          Erhöht Offset im Filenamen
. f03c  c4 ab     cpy $ab      Ende des Namens erreicht ?
. f03e  d0 f0     bne $f030    Nein, dann $f030
. f040  4c 23 f2  jmp $f223    UNLIST

. f043  :
. f043  *** L O A D / V E R I F Y ***
. f043  :
. f043  86 b4     stx $b4      Holt
. f045  84 b5     sty $b5      ... Ladeadresse
. f047  6c 2e 03  jmp ($032e)  LOAD ($f04a)

. f04a  85 93     sta $93      Schreibt Verify-Flag
. f04c  a9 00     lda #$00     Initialisiert
. f04e  85 90     sta $90      ... Status (ST)
. f050  a5 ae     lda $ae      Geräteadresse ungleich Tastatur ?
. f052  d0 03     bne $f057    Ja, dann $f057
. f054  4c 8b f2  jmp $f28b    Gibt 'ILLEGAL DEVICE NUMBER' aus

. f057  c9 03     cmp #$03     Geräteadresse gleich Bildschirm ?
. f059  f0 f9     beq $f054    Ja, dann $f054
. f05b  b0 07     bcs $f064    Adresse gleich Bus ?, dann $f064
. f05d  c9 02     cmp #$02     Geräteadresse gleich RS-232
. f05f  f0 f3     beq $f054    Ja, dann $f054
. f061  4c f0 f0  jmp $f0f0    LOAD von Kassette
```

```
. f064  a4 ab     ldy $ab      Holt Länge des Filenamens
. f066  d0 03     bne $f06b    Filename vorhanden ?, dann $f06b
. f068  4c 88 f2  jmp $f288    Gibt 'FILE NAME MISSING ERROR' aus

. f06b  a6 ad     ldx $ad      Holt Sekundäradresse
. f06d  20 60 f1  jsr $f160    Gibt 'SEARCHING FOR ...' aus
. f070  a9 60     lda #$60     Schreibt Sekundäradresse
. f072  85 ad     sta $ad      ... für LOAD
. f074  20 05 f0  jsr $f005    OPEN BUS
. f077  a5 ae     lda $ae      Holt Geräteadresse
. f079  20 fa ed  jsr $edfa    TALK
. f07c  a5 ad     lda $ad      Holt Sekundäradresse
. f07e  20 1a ee  jsr $ee1a    (TKSA) Gibt Sekundäradresse aus
. f081  20 8b ec  jsr $ec8b    (ACPTR) Holt Byte vom seriellen Bus
. f084  85 9d     sta $9d      Schreibt Ladeadresse (Low)
. f086  a5 90     lda $90      Holt Status
. f088  4a        lsr          War Bit 1
. f089  4a        lsr          ... gesetzt
. f08a  b0 5c     bcs $f0e8    Ja, dann Timeout
. f08c  20 8b ec  jsr $ec8b    (ACPTR) Holt Byte vom seriellen Bus
. f08f  85 9e     sta $9e      Schreibt Ladeadresse (High)
. f091  8a        txa          Sekundäradresse ungleich Null ?
. f092  d0 08     bne $f09c    Ja, dann Linkbyte als Ladeadresse verwenden
. f094  a5 b4     lda $b4      Holt
. f096  85 9d     sta $9d      ... Ladeadresse
. f098  a5 b5     lda $b5      ... aus
. f09a  85 9e     sta $9e      ... $b4-$b5
. f09c  20 89 f1  jsr $f189    Gibt 'LOADING' bzw. 'VERIFYING' aus
. f09f  a9 fd     lda #$fd     Setzt
. f0a1  25 90     and $90      ... Timeout-Bit
. f0a3  85 90     sta $90      ... zurück
. f0a5  20 e1 ff  jsr $ffe1    STOP-Taste gedrückt ?
. f0a8  d0 03     bne $f0ad    Nein, dann $f0ad
. f0aa  4c ff f1  jmp $f1ff    CLOSE Bus

. f0ad  20 8b ec  jsr $ec8b    (ACPTR) Holt Byte vom seriellen Bus
. f0b0  aa        tax          Byte nach X-Reg.
. f0b1  a5 90     lda $90      Holt Status
. f0b3  4a        lsr          War Timeout-Bit
. f0b4  4a        lsr          ... gesetzt ?
. f0b5  b0 e8     bcs $f09f    Ja, dann Ladeversuch wiederholen
. f0b7  8a        txa          Byte nach Akku
. f0b8  a4 93     ldy $93      Überprüft LOAD/VERIFY-Flag
. f0ba  f0 18     beq $f0d4    LOAD ?, dann $f0d4
. f0bc  a0 00     ldy #$00
. f0be  8d c7 07  sta $07c7    Speichert Zeichen
. f0c1  a9 9d     lda #$9d     Schreibt Ladeadresse in Routine
. f0c3  8d df 07  sta $07df    ... für indirekten RAM-Zugriff
. f0c6  20 d9 07  jsr $07d9    Holt Byte aus Speicher (LDA($9d),Y)
. f0c9  cd c7 07  cmp $07c7    Beide Bytes gleich ?
. f0cc  f0 08     beq $f0d6    Ja, dann $f0d6
. f0ce  a9 10     lda #$10     Setzt 'VERIFY ERROR'-Bit
```

```
. f0d0  20 1e f4 jsr $f41e    ... in Status

. f0d3  2c       .byte $2c

. f0d4  91 9d    sta ($9d),y  Schreibt geladenes Byte
. f0d6  e6 9d    inc $9d      Erhöht
. f0d8  d0 02    bne $f0dc    ... Ladezeiger
. f0da  e6 9e    inc $9e      ... um 1
. f0dc  24 90    bit $90      EOI-Bit gesetzt ?
. f0de  50 bf    bvc $f09f    Nein, dann $f09f
. f0e0  20 3b ef jsr $ef3b    UNTALK
. f0e3  20 11 f2 jsr $f211    CLOSE Bus
. f0e6  90 03    bcc $f0eb    Kein Fehler ?, dann $f0eb
. f0e8  4c 7c f2 jmp $f27c    Gibt 'FILE NOT FOUND ERROR' aus

. f0eb  a6 9d    ldx $9d      Holt
. f0ed  a4 9e    ldy $9e      ... Lade-Endadresse
. f0ef  60       rts          ENDE

     KASSETTE

. f0f0  20 1b e3 jsr $e31b    Prüft PLAY-Taste
. f0f3  b0 fa    bcs $f0ef    STOP-Taste gedrückt ?, dann $f0ef
. f0f5  20 60 f1 jsr $f160    Gibt 'SEARCHING FOR ...' aus
. f0f8  a5 ab    lda $ab      Holt Länge des Filenamens
. f0fa  f0 09    beq $f105    Kein Filename ?, dann $f105
. f0fc  20 21 ea jsr $ea21    Sucht Header auf Band
. f0ff  90 0b    bcc $f10c    Header gefunden ?, dann $f10c
. f101  f0 ec    beq $f0ef    'EOT'-Block gefunden ?, dann $f0ef
. f103  b0 e3    bcs $f0e8    'FILE NOT FOUND ERROR' ausgeben

. f105  20 cc e9 jsr $e9cc    Liest Block vom Band
. f108  f0 e5    beq $f0ef    'EOT'-Block gefunden ?, dann $f0ef
. f10a  b0 dc    bcs $f0e8    Kein Block gefunden ?, dann Fehler
. f10c  a5 f8    lda $f8      Holt File-Typ
. f10e  c9 01    cmp #$01     File-Typ 1 (Programm) ?
. f110  f0 12    beq $f124    Ja, dann $f124
. f112  c9 03    cmp #$03     File-Typ 3 (PRG mit Sekundäradresse 1) ?
. f114  d0 e2    bne $f0f8    Nein, dann weitersuchen
. f116  a0 00    ldy #$00
. f118  b1 b6    lda ($b6),y  Kopiert Ladeadresse (Low)
. f11a  85 b4    sta $b4      ... aus File-Header
. f11c  c8       iny
. f11d  b1 b6    lda ($b6),y  Kopiert Ladeadresse (High)
. f11f  85 b5    sta $b5      ... aus File-Header
. f121  4c 28 f1 jmp $f128    Springt nach $f128

. f124  a5 ad    lda $ad      Holt Sekundäradresse
. f126  d0 ee    bne $f116    Sekundäradresse ungleich Null ?, dann $f116
. f128  38       sec          File-Länge (X-Reg., Y-Reg.)
. f129  a0 02    ldy #$02     ... berechnet
. f12b  b1 b6    lda ($b6),y  ... sich
```

```
. f12d  a0 00     ldy #$00      ... aus
. f12f  f1 b6     sbc ($b6),y   ... Endadresse
. f131  aa        tax           ... aus
. f132  a0 03     ldy #$03      ... Header
. f134  b1 b6     lda ($b6),y   ... abzüglich
. f136  a0 01     ldy #$01      ... Anfangsadresse
. f138  f1 b6     sbc ($b6),y   ... aus
. f13a  a8        tay           ... Header
. f13b  18        clc           Endadresse im Speicher
. f13c  8a        txa           ... berechnet
. f13d  65 b4     adc $b4       ... sich
. f13f  85 9d     sta $9d       ... aus
. f141  98        tya           ... Filelänge (X-Reg., Y-Reg.)
. f142  65 b5     adc $b5       ... plus
. f144  85 9e     sta $9e       ... Ladeanfangs-Adresse
. f146  a5 b4     lda $b4       Ladezeiger
. f148  85 b2     sta $b2       ... gleich
. f14a  a5 b5     lda $b5       ... Ladeanfangs-
. f14c  85 b3     sta $b3es     ... Adresse
. f14e  20 89 f1  jsr $f189     Gibt 'LOADING ...' aus
. f151  20 f3 e8  jsr $e8f3     Lädt File vom Band
. f154  90 95     bcc $f0eb     Kein Fehler ?, dann $f0eb
. f156  a9 1d     lda #$1d      Holt 'LOAD'-Code
. f158  24 93     bit $93       Überprüft LOAD/VERIFY-Flag
. f15a  10 93     bpl $f0ef     LOAD ?, dann $f0ef
. f15c  a9 1c     lda #$1c      Holt 'VERIFY'-Code
. f15e  d0 8f     bne $f0ef     Springt immer nach $f0ef

. f160  a5 9a     lda $9a       Holt Flag für Meldung
. f162  10 24     bpl $f188     Flag nicht gesetzt ?
. f164  a0 0c     ldy #$0c      Holt SEARCHING-Code
. f166  20 ca eb  jsr $ebca     Gibt 'SEARCHING' aus
. f169  a5 ab     lda $ab       Holt Länge des Filenamens
. f16b  f0 1b     beq $f188     Kein Filename ?, dann $f188
. f16d  a0 17     ldy #$17      Holt 'FOR'-Code
. f16f  20 ca eb  jsr $ebca     Gibt 'FOR' aus
. f172  a4 ab     ldy $ab       Holt Länge des Filenamens
. f174  f0 12     beq $f188     Kein Filename ?, dann $f188
. f176  a0 00     ldy #$00
. f178  a9 af     lda #$af      Schreibt Adresse des Filenamens
. f17a  8d df 07  sta $07df     ... in Routine für indirekten RAM-Zugriff
. f17d  20 d9 07  jsr $07d9     Holt Zeichen aus Filenamen (LDA($af),Y)
. f180  20 d2 ff  jsr $ffd2     (BSOUT) Gibt Zeichen aus
. f183  c8        iny           Erhöht Offset im Filenamen
. f184  c4 ab     cpy $ab       Ende des Filenamens erreicht ?
. f186  d0 f0     bne $f178     Nein, dann $f178
. f188  60        rts           ENDE

. f189  a0 49     ldy #$49      Holt 'LOADING'-Code
. f18b  a5 93     lda $93       Überprüft LOAD/VERIFY-Flag
. f18d  f0 02     beq $f191     LOAD ?, dann $f191
. f18f  a0 59     ldy #$59      Holt 'VERIFYING'-Code
```

```
. f191  4c c6 eb jmp $ebc6    Gibt 'LOADING' bzw. 'VERIFYING' aus

. f194  :
. f194  *** S A V E ***
. f194  :
. f194  86 9d    stx $9d      Holt
. f196  84 9e    sty $9e      ... Endadresse
. f198  aa       tax          Holt Zeiger auf Anfangsadresse
. f199  b5 00    lda $00,x    Holt
. f19b  85 b2    sta $b2      ... Anfangsadresse
. f19d  b5 01    lda $01,x
. f19f  85 b3    sta $b3
. f1a1  6c 30 03 jmp ($0330)  SAVE ($f1a4)

. f1a4  a5 ae    lda $ae      Holt logische Adresse
. f1a6  d0 03    bne $f1ab    Adresse nicht Tastatur ?, dann $f1ab
. f1a8  4c 8b f2 jmp $f28b    Gibt 'ILLEGAL DEVICE NUMBER' aus

. f1ab  c9 03    cmp #$03     Adresse gleich Bildschirm ?
. f1ad  f0 f9    beq $f1a8    Ja, dann Fehler
. f1af  c9 02    cmp #$02     Adresse gleich RS-232 ?
. f1b1  f0 f5    beq $f1a8    Ja, dann Fehler
. f1b3  90 7f    bcc $f234    Adresse gleich Datasette ?, dann $f234

      BUS

. f1b5  a9 61    lda #$61     Holt Sekundäradresse für SAVE
. f1b7  85 ad    sta $ad      Schreibt Sekundäradresse
. f1b9  a4 ab    ldy $ab      Holt Länge des Filenamens
. f1bb  d0 03    bne $f1c0    Filename vorhanden ?, dann $f1c0
. f1bd  4c 88 f2 jmp $f288    Gibt 'FILE NAME MISSING ERROR' aus

. f1c0  20 05 f0 jsr $f005    OPEN Bus
. f1c3  20 28 f2 jsr $f228    Gibt 'SAVING ...' aus
. f1c6  a5 ae    lda $ae      Holt Geräteadresse
. f1c8  20 2c ee jsr $ee2c    LISTEN
. f1cb  a5 ad    lda $ad      Holt Sekundäradresse
. f1cd  20 4d ee jsr $ee4d    SECOND
. f1d0  a0 00    ldy #$00
. f1d2  a5 b3    lda $b3      Setzt
. f1d4  85 9c    sta $9c      ... Save-Zeiger
. f1d6  a5 b2    lda $b2      ... gleich
. f1d8  85 9b    sta $9b      ... Save-Anfangsadresse
. f1da  a5 9b    lda $9b      Holt Save-Zeiger (Low)
. f1dc  20 df ec jsr $ecdf    (CIOUT) Gibt Byte auf Bus aus
. f1df  a5 9c    lda $9c      Holt Save-Zeiger (High)
. f1e1  20 df ec jsr $ecdf    (CIOUT) Gibt Byte auf Bus aus
. f1e4  38       sec          Subtrahiert
. f1e5  a5 9b    lda $9b      ... Save-Endadresse
. f1e7  e5 9d    sbc $9d      ... vom
. f1e9  a5 9c    lda $9c      ... Save-Zeiger ...
. f1eb  e5 9e    sbc $9e
```

```
. f1ed  b0 1f     bcs $f20e     Ergebnis nicht negativ ?, dann fertig
. f1ef  a9 9b     lda #$9b      Schreibt SAVE-Zeiger in
. f1f1  8d df 07  sta $07df     ... Routine für indirekten RAM-Zugriff
. f1f4  20 d9 07  jsr $07d9     Holt Byte aus Speicher (LDA($9b),Y)
. f1f7  20 df ec  jsr $ecdf     (CIOUT) Gibt Byte auf Bus aus
. f1fa  20 e1 ff  jsr $ffe1     STOP-Taste gedrückt ?
. f1fd  d0 07     bne $f206     Nein, dann $f206
. f1ff  20 11 f2  jsr $f211     CLOSE Bus
. f202  a9 00     lda #$00
. f204  38        sec
. f205  60        rts           ENDE

. f206  e6 9b     inc $9b       Erhöht
. f208  d0 da     bne $f1e4     ... Save-Zeiger
. f20a  e6 9c     inc $9c       ... um 1
. f20c  d0 d6     bne $f1e4     Springt immer nach $f1e4

. f20e  20 23 ef  jsr $ef23     UNLIST
. f211  24 ad     bit $ad       Bit 7 der Sekundäradresse gesetzt ?
. f213  30 11     bmi $f226     Ja, dann $f226
. f215  a5 ae     lda $ae       Holt Geräteadresse
. f217  20 2c ee  jsr $ee2c     LISTEN
. f21a  a5 ad     lda $ad       Holt Sekundäradresse
. f21c  29 ef     and #$ef
. f21e  09 e0     ora #$e0
. f220  20 4d ee  jsr $ee4d     SECOND
. f223  20 23 ef  jsr $ef23     UNLISTEN
. f226  18        clc
. f227  60        rts           ENDE

. f228  a5 9a     lda $9a       Flag für Meldung gesetzt ?
. f22a  10 38     bpl $f264     Nein, dann $f264
. f22c  a0 51     ldy #$51      Holt 'SAVING'-Code
. f22e  20 ca eb  jsr $ebca     Gibt 'SAVING' aus
. f231  4c 72 f1  jmp $f172     Gibt Filenamen aus

    KASSETTE

. f234  20 19 e3  jsr $e319     Prüft PLAY-Taste
. f237  b0 29     bcs $f262     STOP-Taste gedrückt ?, dann $f262
. f239  20 28 f2  jsr $f228     Gibt 'SAVING ...' aus
. f23c  a2 03     ldx #$03      Holt File-Typ 3
. f23e  a5 ad     lda $ad       Holt Sekundäradresse
. f240  29 01     and #$01      Ist 1 enthalten ?
. f242  d0 02     bne $f246     Ja, dann $f246
. f244  a2 01     ldx #$01      Holt File-Typ 1
. f246  86 f8     stx $f8       Schreibt File-Typ
. f248  20 6c e5  jsr $e56c     Schreibt Header auf Band
. f24b  b0 15     bcs $f262     Fehler ?,dann $f262
. f24d  a9 00     lda #$00      File-Typ
. f24f  85 f8     sta $f8       ... gleich 0
. f251  20 b0 e5  jsr $e5b0     Schreibt Speicherbereich auf Band
```

```
. f254  bØ Øc     bcs $f262    Fehler ?, dann $f262
. f256  a5 ad     lda $ad      Holt Sekundäradresse
. f258  29 Ø2     and #$Ø2     Ist 2 enthalten ?
. f25a  fØ Ø5     beq $f261    Nein, dann $f261
. f25c  2Ø fØ e5  jsr $e5fØ    Schreibt 'EOT'-Block
. f25f  bØ Ø1     bcs $f262    Fehler ?, dann $f262
. f261  18        clc          Flag für OK
. f262  a9 ØØ     lda #$ØØ
. f264  6Ø        rts          ENDE

. f265  :
. f265  *** S T O P ***
. f265  :
. f265  a5 91     lda $91      Holt STOP-Flag
. f267  c9 7f     cmp #$7f     STOP-Taste gedrückt ?
. f269  dØ Ø7     bne $f272    Nein, dann fertig
. f26b  Ø8        php
. f26c  2Ø cc ff  jsr $ffcc    (CLRCH) Schließt Ein-/Ausgabekanäle
. f26f  85 ef     sta $ef      Initialisiert Tastaturpuffer
. f271  28        plp
. f272  6Ø        rts          ENDE

. f273  :
. f273  *** I/O-FEHLER ***
. f273  :
. f273  a9 Ø1     lda #$Ø1     'TOO MANY FILES'

. f275  2c        .byte $2c

. f276  a9 Ø2     lda #$Ø2     'FILE OPEN'

. f278  2c        .byte $2c

. f279  a9 Ø3     lda #$Ø3     'FILE NOT OPEN'

. f27b  2c        .byte $2c

. f27c  a9 Ø4     lda #$Ø4     'FILE NOT FOUND'

. f27e  2c        .byte $2c

. f27f  a9 Ø5     lda #$Ø5     'DEVICE NOT PRESENT'

. f281  2c        .byte $2c

. f282  a9 Ø6     lda #$Ø6     'NOT INPUT FILE'

. f284  2c        .byte $2c

. f285  a9 Ø7     lda #$Ø7     'NOT OUTPUT FILE'

. f287  2c        .byte $2c
```

```
. f288  a9 Ø8     lda #$Ø8      'MISSING FILE NAME'

. f28a  2c        .byte $2c

. f28b  a9 Ø9     lda #$Ø9      'ILLEGAL DEVICE NUMBER'
. f28d  48        pha
. f28e  2Ø cc ff  jsr $ffcc     (CLRCH) Schließt alle Ein-/Ausgabekanäle
. f291  aØ ØØ     ldy #$ØØ
. f293  24 9a     bit $9a       Flag für Meldung gesetzt ?
. f295  5Ø Øa     bvc $f2a1     Nein, dann $f2a1
. f297  2Ø ca eb  jsr $ebca     Gibt Meldung aus
. f29a  68        pla           Kopiert Nummer
. f29b  48        pha           ... der Meldung aus Stack
. f29c  Ø9 3Ø     ora #$3Ø      In ASCII-Code umwandeln
. f29e  2Ø d2 ff  jsr $ffd2     (BSOUT) Zeichen ausgeben
. f2a1  68        pla           Holt Meldung vom Stack
. f2a2  38        sec
. f2a3  6Ø        rts           ENDE

. f2a4  :
. f2a4  *** NMI und START ***
. f2a4  :
. f2a4  a2 ff     ldx #$ff
. f2a6  78        sei           Kein Interrupt zugelassen
. f2a7  9a        txs           Initialisiert
. f2a8  d8        cld           ... Prozessor-Stack
. f2a9  2Ø a6 cf  jsr $cfa6     Legt Modultabelle an und initialisiert Module
. f2ac  2Ø Øb f3  jsr $f3Øb     (IOINIT) Initialisiert Ein-/Ausgaberegister
. f2af  2Ø 11 cf  jsr $cf11     STOP-Taste überprüfen
. f2b2  Ø8        php           Ergebnis auf Stack retten
. f2b3  3Ø Ø7     bmi $f2bc     STOP-Taste nicht gedrückt ?, dann $f2bc
. f2b5  a9 a5     lda #$a5      Speicherprüfung
. f2b7  cd Ø8 Ø5  cmp $Ø5Ø8     ... bereits durchgeführt ?
. f2ba  fØ Ø3     beq $f2bf     Ja, dann $f2bf
. f2bc  2Ø 52 f3  jsr $f352     (RAMTAS) Speicherprüfung
. f2bf  2Ø ce f2  jsr $f2ce     (RESTOR) Initialisiert Vektoren
. f2c2  2Ø 4e d8  jsr $d84e     Initialisiert Editor
. f2c5  28        plp           War STOP-Taste gedrückt ?
. f2c6  3Ø Ø3     bmi $f2cb     Nein, dann RESET
. f2c8  4c 45 f4  jmp $f445     MONITOR

. f2cb  4c ØØ 8Ø  jmp $8ØØØ     Basic-Kaltstart

. f2ce  :
. f2ce  *** RESTOR ***
. f2ce  :
. f2ce  a2 eb     ldx #$eb      Holt Adresse
. f2dØ  aØ f2     ldy #$f2      ... der Vektor-Adressen ($f2eb)
. f2d2  18        clc
. f2d3  :
. f2d3  *** VECTOR ***
```

```
. f2d3  :
. f2d3  86 b8     stx $b8       Schreibt Ziel- bzw. Quelladresse
. f2d5  84 b9     sty $b9       ... nach $b8-$b9
. f2d7  a0 1f     ldy #$1f      Holt Offset
. f2d9  b9 12 03  lda $0312,y   Holt Vektoradresse
. f2dc  b0 02     bcs $f2e0     Aufruf mit Carry = 1 ?, dann $f2e0
. f2de  b1 b8     lda ($b8),y   Holt Vektoradresse aus Zieladresse (Carry=0)
. f2e0  99 12 03  sta $0312,y   Schreibt Vektoradresse in Vektortabelle
. f2e3  90 02     bcc $f2e7     Aufruf mit Carry = 0 ?, dann $f2e7
. f2e5  91 b8     sta ($b8),y   Schreibt Vektoradresse in Zielbereich (Carry=1)
. f2e7  88        dey           Erniedrigt Offset um 1
. f2e8  10 ef     bpl $f2d9     Noch nicht Null ?, dann weiterkopieren
. f2ea  60        rts           ENDE

. f2eb  :
. f2eb  *** STANDARD-VEKTOR-ADRESSEN ***
. f2eb  :
. f2eb  42 ce     .adr $ce42    Interrupt (Uhr)
. f2ed  0e ce     .adr $ce0e    Interrupt
. f2ef  4c f4     .adr $f44c    Break Interrupt
. f2f1  53 ef     .adr $ef53    (OPEN)
. f2f3  5d ee     .adr $ee5d    (CLOSE)
. f2f5  18 ed     .adr $ed18    (CHKIN)
. f2f7  60 ed     .adr $ed60    (CHKOUT)
. f2f9  0c ef     .adr $ef0c    (CLRCH)
. f2fb  e8 eb     .adr $ebe8    (BASIN)
. f2fd  4b ec     .adr $ec4b    (BSOUT)
. f2ff  65 f2     .adr $f265    (STOP)
. f301  d9 eb     .adr $ebd9    (GETIN)
. f303  08 ef     .adr $ef08    (CLALL)
. f305  4c f4     .adr $f44c    Monitor Break
. f307  4a f0     .adr $f04a    (LOAD)
. f309  a4 f1     .adr $f1a4    (SAVE)
. f30b  :
. f30b  *** IOINIT ***
. f30b  :
. f30b  a9 0f     lda #$0f      Bit 0-3 Ausgabe, Bit 4-7 Eingabe
. f30d  85 00     sta $00       Schreibt Datenrichtungsregister
. f30f  a9 08     lda #$08      Schaltet
. f311  85 01     sta $01       ... Datasettenmotor ab
. f313  a2 ff     ldx #$ff      Schaltet Schnittstellenbaustein
. f315  8e 10 fd  stx $fd10     ... 6529 auf Ausgabe
. f318  8e f3 fe  stx $fef3     Schaltet IEC-Port #8 auf Ausgabe
. f31b  e8        inx
. f31c  8e f4 fe  stx $fef4
. f31f  8e f0 fe  stx $fef0     Schaltet IEC-Port #8 auf Daten
. f322  a9 40     lda #$40
. f324  8d f5 fe  sta $fef5
. f327  20 ea ed  jsr $edea     IOINIT-Rucksack
. f32a  bd 38 f3  lda $f338,x   Belegt
. f32d  9d 00 ff  sta $ff00,x   ... TED-Register
. f330  e8        inx           ... mit
```

```
. f331  e0 1a     cpx #$1a      ... Standard-
. f333  d0 f5     bne $f32a     ... Werten
. f335  4c 46 eb  jmp $eb46     Springt nach $eb46

. f338  :
. f338  *** TED-STANDARDWERTE ***
. f338  :
. f338  f1 39 00 00 00 00 1b 08
. f340  00 00 02 cc 00 00 00 00
. f348  00 00 04 d0 08 71 5b 75
. f350  77 6e

. f351  :
. f351  *** RAMTAS ***
. f351  :
. f351  a9 00     lda #$00
. f354  a8        tay
. f355  99 02 00  sta $0002,y   Löscht $0002-$0101
. f358  99 00 02  sta $0200,y   Löscht $0200-$02ff
. f35b  99 00 03  sta $0300,y   Löscht $0300-$03ff
. f35e  99 00 04  sta $0400,y   Löscht $0400-$04ff
. f361  99 00 07  sta $0700,y   Löscht $0700-$07ff
. f364  c8        iny
. f365  d0 ee     bne $f355
. f367  a2 08     ldx #$08
. f369  86 9f     stx $9f
. f36b  bd f5 ff  lda $fff5,x   Ermittelt MEMTOP
. f36e  9d f5 ff  sta $fff5,x
. f371  dd f5 3f  cmp $3ff5,x
. f374  d0 01     bne $f377
. f376  c8        iny
. f377  dd f5 7f  cmp $7ff5,x
. f37a  d0 02     bne $f37e
. f37c  c6 9f     dec $9f
. f37e  ca        dex
. f37f  d0 ea     bne $f36b
. f381  c0 08     cpy #$08
. f383  f0 07     beq $f38c
. f385  a5 9f     lda $9f
. f387  d0 08     bne $f391
. f389  a0 7f     ldy #$7f      MEMTOP = $7ff6

. f38b  2c        .byte $2c

. f38c  a0 3f     ldy #$3f      MEMTOP = $3ff6
. f38e  a2 f6     ldx #$f6

. f390  2c        .byte $2c

. f391  a0 fd     ldy #$fd      MEMTOP = $fdf6
. f393  18        clc
. f394  20 2f f4  jsr $f42f     (MEMTOP) Setzt Zeiger für obere Speichergrenze
```

```
. f397  a9 10     lda #$10     MEMBOT = $1000
. f399  8d 32 05  sta $0532    Setzt Zeiger für untere Speichergrenze
. f39c  a2 3a     ldx #$3a     Programmierung
. f39e  bd d1 f3  lda $f3d1,x  ... der
. f3a1  9d 5e 05  sta $055e,x  ... Funktionstasten...
. f3a4  ca        dex
. f3a5  d0 f7     bne $f39e
. f3a7  8e 5d 05  stx $055d    Initialisiert Zähler für Funktionstastenlänge
. f3aa  a2 0b     ldx #$0b     Kopiert
. f3ac  bd b3 cf  lda $cfb3,x  ... Routine
. f3af  9d d9 07  sta $07d9,x  ... für
. f3b2  ca        dex          ... indirekten RAM-Zugriff
. f3b3  10 f7     bpl $f3ac    ... nach $07d9
. f3b5  a2 0f     ldx #$0f     Kopiert
. f3b7  bd 43 e1  lda $e143,x  ... Farbcodes
. f3ba  9d 13 01  sta $0113,x  ... ins
. f3bd  ca        dex          ... RAM
. f3be  10 f7     bpl $f3b7
. f3c0  a9 a5     lda #$a5     Setzt Flag
. f3c2  8d 08 05  sta $0508    ... für ausgeführtes RAMTAS
. f3c5  a9 04     lda #$04     Initialisiert
. f3c7  8d fa 07  sta $07fa    ... ROM-Maske für geteilten Bildschirm
. f3ca  a9 18     lda #$18     Initialisiert
. f3cc  8d fb 07  sta $07fb    ... Video-RAM-Maske für geteilten Bildschirm
. f3cf  60        rts          ENDE

. f3d0  ea        nop
. f3d1  ea        nop

. f3d2  :
. f3d2  *** FUNKTIONSTASTEN ***
. f3d2  :
. f3d2  07 06 0a 07 06 04 05 05   Länge der Texte

. f3da  .byte 'graphic'           Funktionstaste 1
. f3e1  .byte 'dload"'            Funktionstaste 2
. f3e7  .byte 'directory' $0d     Funktionstaste 3
. f3f1  .byte 'scnclr' $0d        Funktionstaste 4
. f3f8  .byte 'dsave"'            Funktionstaste 5
. f3fe  .byte 'run' $0d           Funktionstaste 6
. f402  .byte 'list' $0d          Funktionstaste 7
. f407  .byte 'help' $0d          HELP-Taste

. f40c  :
. f40c  *** SETNAM ***
. f40c  :
. f40c  85 ab     sta $ab         Schreibt Länge des Filenamens
. f40e  86 af     stx $af         Schreibt Adresse
. f410  84 b0     sty $b0         ... des Filenamens
. f412  60        rts             ENDE

. f413  :
```

```
. f413  *** SETNAM ***
. f413  :
. f413  85 ac     sta $ac          Schreibt logische Adresse
. f415  86 ae     stx $ae          Schreibt Geräteadresse
. f417  84 ad     sty $ad          Schreibt Sekundäradresse
. f419  60        rts              ENDE

. f41a  :
. f41a  *** SETMSG ***
. f41a  :
. f41a  85 9a     sta $9a          Schreibt Flag für Systemmeldungen
. f41c  :
. f41c  *** READST ***
. f41c  :
. f41c  a5 90     lda $90          Liest Statuswort (ST)
. f41e  05 90     ora $90          ODER-Verknüpfung von Akku und Status
. f420  85 90     sta $90          Status zurückschreiben
. f422  60        rts              ENDE

. f423  :
. f423  *** SETTMO ***
. f423  :
. f423  8d 35 05 sta $0535         Setzt Timeout-Flag für IEC-Bus
. f426  60        rts              ENDE

. f427  :
. f427  *** MEMTOP ***
. f427  :
. f427  90 06     bcc $f42f        Carry = 0 ?, dann Obergrenze setzen
. f429  ae 33 05 ldx $0533         Holt
. f42c  ac 34 05 ldy $0534         ... RAM-Obergrenze
. f42f  8e 33 05 stx $0533         Setzt
. f432  8c 34 05 sty $0534         ... Obergrenze
. f435  60        rts              ENDE

. f436  :
. f436  *** MEMBOT ***
. f436  :
. f436  90 06     bcc $f43e        Carry = 0 ?, dann Untergrenze setzen
. f438  ae 31 05 ldx $0531         Holt
. f43b  ac 32 05 ldy $0532         ... RAM-Untergrenze
. f43e  8e 31 05 stx $0531         Setzt
. f441  8c 32 05 sty $0532         ... Untergrenze
. f444  60        rts              ENDE

. f445  :
. f445  *** M O N I T O R ***
. f445  :
. f445  a2 00     ldx #$00      Initialisiert
. f447  8e 54 05 stx $0554      ... Prozessor-Flags
. f44a  f0 0c     beq $f458     Springt immer nach $f458
```

```
. f44c  :
. f44c  *** MONITOR-BREAK ***
. f44c  :
. f44c  d8       cld          Kopiert
. f44d  a2 05    ldx #$05     ... Break-Adr., PC, Akku, X-Reg. und Y-Reg.
. f44f  68       pla          ... vom Stack
. f450  9d 52 05 sta $0552,x  ... nach $0552-$0557
. f453  ca       dex
. f454  10 f9    bpl $f44f
. f456  a2 09    ldx #$09     Setzt Flag
. f458  8e f4 07 stx $07f4    ... für 'BREAK'
. f45b  a9 c0    lda #$c0     Setzt Flag
. f45d  85 9a    sta $9a      ... für 'alle Meldungen zugelassen'
. f45f  ba       tsx          Speichert
. f460  8e 58 05 stx $0558    ... Stack-Zeiger
. f463  ae f4 07 ldx $07f4    Holt Meldung
. f466  20 66 cf jsr $cf66    Gibt 'MONITOR' bzw. 'BREAK' aus
. f469  ad 06 ff lda $ff06    Schaltet
. f46c  09 10    ora #$10     ... HIRES-Modus
. f46e  8d 06 ff sta $ff06    ... ab
. f471  a9 00    lda #$00     Löscht
. f473  85 a1    sta $a1      ... temporären
. f475  85 a2    sta $a2      ... Datenspeicher
. f477  58       cli          Interrupt zugelassen
. f478  :
. f478  *** REGISTER-BEFEHL (R) ***
. f478  :
. f478  a2 0f    ldx #$0f
. f47a  20 66 cf jsr $cf66    Gibt Register-Kopfleiste aus
. f47d  ad 52 05 lda $0552    Holt Break-Adresse (High)
. f480  20 10 fb jsr $fb10    Gibt Break-Adresse aus
. f483  a0 00    ldy #$00
. f485  b9 53 05 lda $0553,y  Holt Break-Adresse (Low), SR, AC, XR, YR & SP
. f488  20 05 fb jsr $fb05    Gibt Bytes mit Leerzeichen aus
. f48b  c8       iny
. f48c  c0 06    cpy #$06
. f48e  90 f5    bcc $f485
. f490  b0 03    bcs $f495    Springt nach $f495

. f492  :
. f492  *** EINGABESCHLEIFE ***
. f492  :
. f492  20 0b fb jsr $fb0b    Gibt '?' aus
. f495  20 3a fb jsr $fb3a    Gibt CR-Code aus
. f498  a2 00    ldx #$00     Initialisiert
. f49a  86 f3    stx $f3      ... Zeiger im Eingabepuffer
. f49c  20 cf ff jsr $ffcf    (BASIN) Holt Zeichen
. f49f  9d 00 02 sta $0200,x  Speichert Zeichen im Eingabepuffer
. f4a2  e8       inx          Erhöht Zeiger im Eingabepuffer
. f4a3  c9 0d    cmp #$0d     Zeichen gleich CR-Code ?
. f4a5  d0 f5    bne $f49c    Nein, dann $f49c
. f4a7  ca       dex          Erniedrigt Zeiger im Eingabepuffer
```

```
. f4a8  86 f4     stx $f4       Schreibt Puffer-Endezeiger
. f4aa  20 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
. f4ad  f0 e6     beq $f495     Endmarke oder ':' gefunden ?, dann $f495
. f4af  c9 20     cmp #$20      Leerzeichen gefunden ?
. f4b1  f0 f7     beq $f4aa     Ja, dann überlesen
. f4b3  a2 0f     ldx #$0f      Sucht Zeichen
. f4b5  dd 70 f5  cmp $f570,x   ... in Kommando-Tabelle
. f4b8  f0 05     beq $f4bf     Kommando erkannt ?, dann $f4bf
. f4ba  ca        dex
. f4bb  10 f8     bpl $f4b5
. f4bd  30 d3     bmi $f492     Zeichen nicht in Tabelle ?, dann $f492

. f4bf  e0 0d     cpx #$0d      LOAD-, SAVE- oder VERIFY-Kommando ?
. f4c1  b0 0e     bcs $f4d1     Ja, dann $f4d1
. f4c3  8a        txa           Berechnet
. f4c4  0a        asl           ... Offset
. f4c5  aa        tax           ... in Adreßtabelle
. f4c6  bd 81 f5  lda $f581,x   Holt
. f4c9  48        pha           ... Adresse des Kommandos
. f4ca  bd 80 f5  lda $f580,x   ... und legt diese
. f4cd  48        pha           ... auf Stack
. f4ce  4c ad fa  jmp $faad     Übernimmt Adresse und ruft Routine auf

. f4d1  8d 5b 05  sta $055b     Schreibt LOAD-, SAVE- oder VERIFY-Kommando
. f4d4  4c 6e f6  jmp $f66e     Führt Kommando aus

. f4d7  :
. f4d7  *** MEMORY-DUMP (M) ***
. f4d7  :
. f4d7  b0 08     bcs $f4e1     Keine Adresse ?, dann $f4e1
. f4d9  20 5b fb  jsr $fb5b     Kopiert Adresse nach Zeiger ($a1-$a2)
. f4dc  20 ad fa  jsr $faad     Übernimmt 2. Adresse
. f4df  90 06     bcc $f4e7     Adresse angegeben ?, dann $f4e7
. f4e1  a9 0b     lda #$0b      Setzt Zeilenzähler
. f4e3  85 f1     sta $f1       ... auf 11 Zeilen
. f4e5  d0 0e     bne $f4f5     Springt immer nach $f4f5

. f4e7  20 64 fb  jsr $fb64     Berechnet Länge des Bereiches
. f4ea  4a        lsr           Dividiert
. f4eb  66 f1     ror $f1       ... Länge
. f4ed  4a        lsr           ... durch 8
. f4ee  66 f1     ror $f1       ... und
. f4f0  4a        lsr           ... schreibt
. f4f1  66 f1     ror $f1       ... das Ergebnis
. f4f3  85 f2     sta $f2       ... in Zeilenzähler ($f1-$f2)
. f4f5  20 e1 ff  jsr $ffe1     Prüft STOP-Taste
. f4f8  f0 0d     beq $f507     STOP-Taste gedrückt ?, dann $f507
. f4fa  20 9a f5  jsr $f59a     Gibt Zeile, Bytes und Zeichen aus
. f4fd  a9 08     lda #$08      Addiert 8 Byte
. f4ff  20 96 fb  jsr $fb96     ... zum Zeiger
. f502  20 72 fb  jsr $fb72     Erniedrigt Zeilenzähler um 1 Zeile
. f505  b0 ee     bcs $f4f5     Noch nicht fertig ?, dann $f4f5
```

```
. f507  4c 95 f4 jmp $f495    Eingabeschleife

. f50a  :
. f50a  *** REGISTER-ÄNDERN (;) ***
. f50a  :
. f50a  b0 fb    bcs $f507    Keine Adresse ?, dann $f507
. f50c  a5 f1    lda $f1      Holt
. f50e  a4 f2    ldy $f2      ... Adresse
. f510  8d 53 05 sta $0553    Schreibt
. f513  8c 52 05 sty $0552    ... Programmzähler (PC)
. f516  a0 00    ldy #$00     Initialisiert Zähler
. f518  20 ad fa jsr $faad    Übernimmt Byte
. f51b  b0 ea    bcs $f507    Zeilenende ?, dann $f507
. f51d  a5 f1    lda $f1      Holt übernommenes Byte
. f51f  99 54 05 sta $0554,y  Schreibt SR, AC, XR, YR bzw. SP
. f522  c8       iny          Erhöht Zähler um 1
. f523  c0 05    cpy #$05     Schon alle 5 Register übernommen ?
. f525  90 f1    bcc $f518    Nein, dann $f518
. f527  b0 de    bcs $f507    Springt immer nach $f507

. f529  :
. f529  *** MEMORY-DUMP EDITIEREN (>) ***
. f529  :
. f529  b0 13    bcs $f53e    Keine Adresse ?, dann $f53e
. f52b  20 5b fb jsr $fb5b    Kopiert Adresse nach Zeiger ($a1-$a2)
. f52e  a0 00    ldy #$00     Initialisiert Zähler
. f530  20 ad fa jsr $faad    Übernimmt Byte
. f533  b0 09    bcs $f53e    Zeilenende ?, dann $f53e
. f535  a5 f1    lda $f1      Holt übernommenes Byte
. f537  91 a1    sta ($a1),y  Speichert Byte an angegebener Adresse
. f539  c8       iny          Erhöht Zähler um 1
. f53a  c0 08    cpy #$08     Schon 8 Byte übernommen ?
. f53c  90 f2    bcc $f530    Nein, dann $f530
. f53e  20 d8 fb jsr $fbd8    Gibt Meldung aus

. f541  1b 4f 91              'ESC o' und 'CURSOR UP'-Code
. f544  00       .byte $00    Endmarke der Meldung

. f545  20 9a f5 jsr $f59a    Gibt Adresse, 8 Byte und Zeichen aus
. f548  4c 95 f4 jmp $f495    Eingabeschleife

. f54b  :
. f54b  *** GOTO (G) ***
. f54b  :
. f54b  b0 0a    bcs $f557    Keine Adresse ?, dann $f557
. f54d  a5 f1    lda $f1      Kopiert
. f54f  8d 53 05 sta $0553    ... angegebene Adresse
. f552  a5 f2    lda $f2      ... nach
. f554  8d 52 05 sta $0552    ... Programmadresse
. f557  ae 58 05 ldx $0558    Setzt
. f55a  9a       txs          ... Stackzeiger
. f55b  a2 00    ldx #$00     Kopiert
```

```
. f55d  bd 52 Ø5 lda $Ø552,x  ... Programmadresse
. f56Ø  48       pha          ... und
. f561  e8       inx          ... Status
. f562  eØ Ø3    cpx #$Ø3     ... auf
. f564  dØ f7    bne $f55d    ... Stack
. f566  ae 56 Ø5 ldx $Ø556    Holt X-Reg.
. f569  ac 57 Ø5 ldy $Ø557    Holt Y-Reg.
. f56c  ad 55 Ø5 lda $Ø555    Holt Akku
. f56f  4Ø       rti          Programmaufruf

. f57Ø  :
. f57Ø  *** KOMMANDO-TABELLE ***
. f57Ø  :
. f57Ø  .byte 'xmrgtcda.hf>;lsv'

. f58Ø  Ø2 8Ø    .adr $8ØØ2   (X) Exit
. f582  d6 f4    .adr $f4d6   (M) Memory-Dump
. f584  77 f4    .adr $f477   (R) Register-Befehl
. f586  4a f5    .adr $f54a   (G) Goto
. f588  dØ f5    .adr $f5dØ   (T) Transfer
. f58a  cd f5    .adr $f5cd   (C) Compare
. f58c  23 f7    .adr $f723   (D) Disassemble
. f58e  1e f9    .adr $f91e   (A) Assemble
. f59Ø  1e f9    .adr $f91e   (.) Assemble
. f592  Ød f6    .adr $f6Ød   (H) Hunt
. f594  Ø9 f7    .adr $f7Ø9   (F) Fill
. f596  28 f5    .adr $f528   (>) Memory-Dump editieren
. f598  Ø9 f5    .adr $f5Ø9   (;) Register ändern

. f59a  :
. f59a  *** ADRESSE, BYTES UND ZEICHEN DARSTELLEN ***
. f59a  :
. f59a  2Ø 3a fb jsr $fb3a    Gibt CR-Code aus
. f59d  a9 3e    lda #$3e     Holt '>'-Code
. f59f  2Ø d2 ff jsr $ffd2    (BSOUT) Gibt Zeichen aus
. f5a2  2Ø fb fa jsr $fafb    Gibt Adresse aus
. f5a5  aØ ØØ    ldy #$ØØ     Initialisiert Zähler
. f5a7  2Ø 96 cf jsr $cf96    Holt Byte (LDA($A1),Y)
. f5aa  2Ø Ø5 fb jsr $fbØ5    Gibt Byte und Leerzeichen aus
. f5ad  c8       iny          Erhöht Zähler um 1
. f5ae  cØ Ø8    cpy #$Ø8     Schon 8 Byte ausgegeben ?
. f5bØ  9Ø f5    bcc $f5a7    Nein, dann $f5a7
. f5b2  2Ø d8 fb jsr $fbd8    Gibt Meldung aus

. f5b5  3a 12   .byte $3a $12 ':' und 'RVS ON'
. f5b7  ØØ      .byte $ØØ     Endmarke

. f5b8  aØ ØØ    ldy #$ØØ     Initialisiert Zähler
. f5ba  2Ø 96 cf jsr $cf96    Holt Byte (LDA($A1),Y)
. f5bd  29 7f    and #$7f     Löscht Bit 7
. f5bf  c9 2Ø    cmp #$2Ø     Zeichen gleich Steuercode ?
. f5c1  bØ Ø2    bcs $f5c5    Nein, dann $f5c5
```

```
. f5c3  a9 2e     lda #$2e      Sonst durch Punkt '.' ersetzen
. f5c5  20 d2 ff  jsr $ffd2     (BSOUT) Zeichen ausgeben
. f5c8  c8        iny           Zähler um 1 erhöhen
. f5c9  c0 08     cpy #$08      Schon 8 Zeichen ausgegeben ?
. f5cb  90 ed     bcc $f5ba     Nein, dann $f5ba
. f5cd  60        rts           ENDE

. f5ce  :
. f5ce  *** COMPARE (C) ***
. f5ce  :
. f5ce  a9 00     lda #$00      Holt Flag für 'COMPARE'

. f5d0  2c        .byte $2c
. f5d1  :
. f5d1  *** TRANSFER (C) ***
. f5d1  :
. f5d1  a9 80     lda #$80      Holt Flag für 'TRANSFER'
. f5d3  85 bb     sta $bb       Setzt Flag für Compare/Transfer
. f5d5  20 a0 fb  jsr $fba0     Holt Quelladresse und Blocklänge
. f5d8  b0 30     bcs $f60a     Keine Adresse ?, dann Fehler
. f5da  20 ad fa  jsr $faad     Übernimmt Zieladresse
. f5dd  b0 2b     bcs $f60a     Keine Zieladresse ?, dann Fehler
. f5df  20 3a fb  jsr $fb3a     Gibt CR-Code aus
. f5e2  a0 00     ldy #$00
. f5e4  20 96 cf  jsr $cf96     Holt Byte (LDA($A1),Y)
. f5e7  24 bb     bit $bb       Überprüft Compare/Transfer-Flag
. f5e9  10 02     bpl $f5ed     Compare ?, dann $f5ed
. f5eb  91 f1     sta ($f1),y   Schreibt Byte an Zieladresse
. f5ed  d1 f1     cmp ($f1),y   Vergleicht Byte mit Byte in Zieladresse
. f5ef  f0 08     beq $f5f9     Bytes identisch ?, dann $f5f9
. f5f1  20 e1 ff  jsr $ffe1     STOP-Taste gedrückt ?
. f5f4  f0 11     beq $f607     Ja, dann $f607
. f5f6  20 fb fa  jsr $fafb     Gibt Quelladresse aus
. f5f9  e6 f1     inc $f1       Erhöht
. f5fb  d0 02     bne $f5ff     ... Zieladresse
. f5fd  e6 f2     inc $f2       ... um 1
. f5ff  20 94 fb  jsr $fb94     Erhöht Quelladresse um 1
. f602  20 86 fb  jsr $fb86     Erniedrigt Zeichenzähler um 1
. f605  b0 dd     bcs $f5e4     Kein Unterlauf ?, dann $f5e4
. f607  4c 95 f4  jmp $f495     Eingabeschleife

. f60a  4c 92 f4  jmp $f492     Fehler und Eingabeschleife

. f60d  ea        nop

. f60e  :
. f60e  *** HIND (H) ***
. f60e  :
. f60e  20 a0 fb  jsr $fba0     Holt Adresse und Blocklänge
. f611  b0 f7     bcs $f60a     Keine Adresse ?, dann Fehler
. f613  a0 00     ldy #$00      Initialisiert Zähler
. f615  20 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
```

```
. f618  c9 27     cmp #$27      Zeichen gleich Hochkomma (') ?
. f61a  d0 12     bne $f62e     Nein, dann $f62e
. f61c  20 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
. f61f  99 5d 02  sta $025d,y   Schreibt Zeichen in Suchstring
. f622  c8        iny           Erhöht Zähler um 1
. f623  20 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
. f626  f0 1b     beq $f643     Trennzeichen ?, dann $f643
. f628  c0 20     cpy #$20      Schon 32 Zeichen übernommen ?
. f62a  d0 f3     bne $f61f     Nein, dann $f61f
. f62c  f0 15     beq $f643     Springt immer nach $f643

. f62e  8c 5c 05  sty $055c     Y-Reg. gleich 0
. f631  20 ab fa  jsr $faab     Übernimmt Byte
. f634  a5 f1     lda $f1       Holt übernommenes Byte
. f636  99 5d 02  sta $025d,y   Schreibt Byte in Suchstring
. f639  c8        iny           Erhöht Zähler um 1
. f63a  20 ad fa  jsr $faad     Übernimmt Byte
. f63d  b0 04     bcs $f643     Zeilenende ?, dann $f643
. f63f  c0 20     cpy #$20      Schon 32 Byte übernommen ?
. f641  d0 f1     bne $f634     Nein, dann $f634
. f643  8c 5b 05  sty $055b     Schreibt Zähler
. f646  20 3a fb  jsr $fb3a     Gibt CR-Code aus
. f649  a2 00     ldx #$00
. f64b  a0 00     ldy #$00
. f64d  20 96 cf  jsr $cf96     Holt Byte aus Quelladresse
. f650  dd 5d 02  cmp $025d,x   Gleich Byte im Suchstring ?
. f653  d0 0f     bne $f664     Nein, dann $f664
. f655  c8        iny
. f656  e8        inx
. f657  ec 5b 05  cpx $055b     Schon alle Zeichen verglichen
. f65a  d0 f1     bne $f64d     Nein, dann $f64d
. f65c  20 e1 ff  jsr $ffe1     STOP-Taste gedrückt ?
. f65f  f0 a6     beq $f607     Ja, dann $f607
. f661  20 fb fa  jsr $fafb     Gibt Adresse aus
. f664  20 94 fb  jsr $fb94     Erhöht Quelladresse um 1
. f667  20 86 fb  jsr $fb86     Erniedrigt Blocklänge um 1
. f66a  b0 dd     bcs $f649     Kein Unterlauf ?, dann $f649
. f66c  90 99     bcc $f607     Springt immer nach $f607

. f66e  :
. f66e  *** LOAD (L), SAVE (S), VERIFY (V) ***
. f66e  :
. f66e  a0 01     ldy #$01      Vorbesetzung für Geräteadresse (Datasette)
. f670  84 ae     sty $ae       Schreibt Geräteadresse
. f672  84 ad     sty $ad       Schreibt Sekundäradresse
. f674  88        dey           Vorbesetzung 0 für Status, Filename & Load
. f675  84 ab     sty $ab       Schreibt Länge des Filenamens
. f677  84 90     sty $90       Schreibt Status
. f679  84 93     sty $93       Schreibt LOAXD/VERIFY-Flag
. f67b  a9 02     lda #$02      Schreibt
. f67d  85 b0     sta $b0       ... Adresse
. f67f  a9 5d     lda #$5d      ... des
```

```
. f681  85 af     sta $af       ... Filenamens ($Ø25d)
. f683  2Ø 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
. f686  fØ 5e     beq $f6e6     Zeichen gleich Trennzeichen ?, dann $f6e6
. f688  c9 2Ø     cmp #$2Ø      Zeichen gleich Leerzeichen ?
. f68a  fØ f7     beq $f683     Ja, dann überlesen
. f68c  c9 22     cmp #$22      Zeichen gleich Anführungszeichen ?
. f68e  dØ 17     bne $f6a7     Nein, dann Fehler
. f69Ø  a6 f3     ldx $f3       Holt Zeiger im Eingabepuffer
. f692  e4 f4     cpx $f4       Pufferende erreicht ?
. f694  bØ 5Ø     bcs $f6e6     Ja, dann $f6e6
. f696  bd ØØ Ø2  lda $Ø2ØØ,x   Holt Zeichen aus Puffer
. f699  e8        inx           Erhöht Offset im Puffer
. f69a  c9 22     cmp #$22      Ende des Filenamens erreicht ?
. f69c  fØ Ød     beq $f6ab     Ja, dann $f6ab
. f69e  91 af     sta ($af),y   Schreibt Filenamen in Monitor-Puffer
. f6aØ  e6 ab     inc $ab       Erhöht Länge des Filenamens um 1
. f6a2  c8        iny           Erhöht Offset um 1
. f6a3  cØ 11     cpy #$11      Schon 17 Zeichen gelesen ?
. f6a5  9Ø eb     bcc $f692     Nein, dann $f692
. f6a7  4c 92 f4  jmp $f492     Fehler und Eingabeschleife

. f6aa  ea        nop

. f6ab  86 f3     stx $f3       Schreibt Zeiger im Eingabepuffer
. f6ad  2Ø 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
. f6bØ  2Ø ad fa  jsr $faad     Holt Geräteadresse
. f6b3  bØ 31     bcs $f6e6     Keine Geräteadresse ?, dann $f6e6
. f6b5  a5 f1     lda $f1       Holt Geräteadresse
. f6b7  fØ ee     beq $f6a7     Geräteadresse gleich Ø ?, dann Fehler
. f6b9  c9 Ø3     cmp #$Ø3      Geräteadresse gleich 3 ?
. f6bb  fØ ea     beq $f6a7     Ja, dann Fehler
. f6bd  85 ae     sta $ae       Schreibt Geräteadresse
. f6bf  2Ø ad fa  jsr $faad     Übernimmt Anfangsadresse
. f6c2  bØ 22     bcs $f6e6     Keine Anfangsadresse ?, dann $f6e6
. f6c4  2Ø 5b fb  jsr $fb5b     Kopiert Adresse nach Zeiger ($a1-$a2)
. f6c7  2Ø ad fa  jsr $faad     Übernimmt Endadresse
. f6ca  bØ db     bcs $f6a7     Keine Endadresse ?, dann Fehler
. f6cc  2Ø 3a fb  jsr $fb3a     Gibt CR-Code aus
. f6cf  a6 f1     ldx $f1       Holt
. f6d1  a4 f2     ldy $f2       ... Endadresse
. f6d3  ad 5b Ø5  lda $Ø55b     Holt Kommando
. f6d6  c9 53     cmp #$53      SAVE-Kommando ?
. f6d8  dØ cd     bne $f6a7     Nein, dann Fehler
. f6da  a9 ØØ     lda #$ØØ      Initialisiert
. f6dc  85 ad     sta $ad       ... Sekundäradresse
. f6de  a9 a1     lda #$a1      Holt Zeiger auf Anfangsadresse
. f6eØ  2Ø d8 ff  jsr $ffd8     SAVE
. f6e3  4c 95 f4  jmp $f495     Eingabeschleife

. f6e6  ad 5b Ø5  lda $Ø55b     Holt Kommando
. f6e9  c9 56     cmp #$56      VERIFY-Kommando ?
. f6eb  fØ Ø6     beq $f6f3     Ja, dann $f6f3
```

```
. f6ed  c9 4c     cmp #$4c      LOAD-Kommando ?
. f6ef  d0 b6     bne $f6a7     Nein, dann Fehler
. f6f1  a9 00     lda #$00      Holt Flag für 'LOAD'
. f6f3  20 d5 ff  jsr $ffd5     LOAD/VERIFY
. f6f6  a5 90     lda $90       Holt Status
. f6f8  29 10     and #$10      LOAD- oder VERIFY-ERROR ?
. f6fa  f0 e7     beq $f6e3     Nein, dann zur Eingabeschleife
. f6fc  ad 5b 05  lda $055b     Holt Kommando
. f6ff  c9 4c     cmp #$4c      LOAD-Kommando ?
. f701  f0 a4     beq $f6a7     Ja, dann Fehler
. f703  a2 2a     ldx #$2a      Holt 'ERROR'-Code
. f705  20 66 cf  jsr $cf66     Gibt Meldung aus
. f708  30 d9     bmi $f6e3     Springt immer zur Eingabeschleife

. f70a  :
. f70a  *** FILL (F) ***
. f70a  :
. f70a  20 a0 fb  jsr $fba0     Holt Zieladresse und Blocklänge
. f70d  b0 98     bcs $f6a7     Keine Angaben ?, dann Fehler
. f70f  20 ad fa  jsr $faad     Holt Füll-Code
. f712  b0 93     bcs $f6a7     Kein Füll-Code angegeben ?, dann Fehler
. f714  a0 00     ldy #$00
. f716  a5 f1     lda $f1       Holt Füllcode
. f718  91 a1     sta ($a1),y   Schreibt Byte in Zielzeiger
. f71a  20 94 fb  jsr $fb94     Erhöht Zieladresse um 1
. f71d  20 86 fb  jsr $fb86     Erniedrigt Blocklänge um 1
. f720  b0 f4     bcs $f716     Kein Unterlauf ?, dann $f716
. f722  90 bf     bcc $f6e3     Springt immer zur Eingabeschleife

. f724  :
. f724  *** DISASSEMBLE (D) ***
. f724  :
. f724  b0 08     bcs $f72e     Keine Adresse ?, dann $f72e
. f726  20 5b fb  jsr $fb5b     Kopiert Anfangsadresse nach Zeiger ($a1-$a2)
. f729  20 ad fa  jsr $faad     Holt Endadresse
. f72c  90 06     bcc $f734     Keine Endadresse ?, dann $f734
. f72e  a9 14     lda #$14      Setzt Zeilenzähler
. f730  85 f1     sta $f1       ... auf 20 Zeilen
. f732  d0 03     bne $f737     Springt immer nach $f737

. f734  20 64 fb  jsr $fb64     Berechnet Blocklänge
. f737  20 3a fb  jsr $fb3a     Gibt CR-Code aus
. f73a  20 e1 ff  jsr $ffe1     STOP-Taste gedrückt ?
. f73d  f0 a4     beq $f6e3     Ja, dann $f6e3
. f73f  20 52 f7  jsr $f752     Disassembliert Befehl
. f742  e6 f6     inc $f6       Erhöht Operandenlänge um 1
. f744  a5 f6     lda $f6       Holt Befehlslänge
. f746  20 96 fb  jsr $fb96     Erhöht Adresse um Befehlslänge
. f749  a5 f6     lda $f6       Holt Befehlslänge
. f74b  20 74 fb  jsr $fb74     Erniedrigt Blocklänge um Befehlslänge
. f74e  b0 e7     bcs $f737     Kein Unterlauf ?, dann $f737
. f750  90 91     bcc $f6e3     Springt immer zur Eingabeschleife
```

```
      DISASSEMBLIERT BEFEHL

. f752  a9 2e     lda #$2e      Holt Punkt (.)
. f754  20 d2 ff  jsr $ffd2     (BSOUT) Gibt Zeichen aus
. f757  20 08 fb  jsr $fb08     Gibt Leerzeichen aus
. f75a  20 fb fa  jsr $fafb     Gibt Adresse aus
. f75d  20 08 fb  jsr $fb08     Gibt Leerzeichen aus
. f760  a0 00     ldy #$00
. f762  20 96 cf  jsr $cf96     Holt Opcode (LDA($a1),Y)
. f765  20 d4 f7  jsr $f7d4     Analysiert Opcode
. f768  48        pha           Rettet Tabellenoffset auf Stack
. f769  a6 f6     ldx $f6       Holt Anzahl der Operandenbytes
. f76b  e8        inx           Berechnet Befehlslänge
. f76c  ca        dex           Schon alle Bytes ausgegeben ?
. f76d  10 0b     bpl $f77a     Nein, dann $f77a
. f76f  20 d8 fb  jsr $fbd8     Gibt 3 Leerzeichen aus

. f772  20 20 20  .byte '   '
. f775  00        .byte $00     Endmarke

. f776  4c 80 f7  jmp $f780     Springt immer nach $f780

. f779  ea        nop

. f77a  20 96 cf  jsr $cf96     Holt Objekt-Byte (LDA($a1),Y)
. f77d  20 05 fb  jsr $fb05     Gibt Byte als HEX-Ziffer aus
. f780  c8        iny           Maximal 3 Byte ausgeben
. f781  c0 03     cpy #$03      Schon Maximalwert erreicht ?
. f783  90 e7     bcc $f76c     Nein, dann $f76c
. f785  68        pla           Holt Tabellenoffset vom Stack
. f786  a2 03     ldx #$03
. f788  20 1b f8  jsr $f81b     Gibt MNEMO zum Opcode aus
. f78b  a2 06     ldx #$06      Erzeugt Operandenwort
. f78d  e0 03     cpx #$03
. f78f  d0 14     bne $f7a5
. f791  a4 f6     ldy $f6       Operandenlänge gleich Null ?
. f793  f0 10     beq $f7a5     Ja, dann $f7a5
. f795  ad 4b 05  lda $054b     Holt Adressierungs-Flag
. f798  c9 e8     cmp #$e8      Überprüft ob Branch-Befehl
. f79a  20 96 cf  jsr $cf96     Holt Operandenbyte (LDA($a1),Y)
. f79d  b0 1d     bcs $f7bc     Branch-Befehl ?, dann $f7bc
. f79f  20 10 fb  jsr $fb10     Ausgabe
. f7a2  88        dey           Erniedrigt Operandenlänge um 1
. f7a3  d0 f0     bne $f795     Noch nicht Null ?, dann $f795
. f7a5  0e 4b 05  asl $054b     Adressierungs-Flag gesetzt ?
. f7a8  90 0e     bcc $f7b8     Nein, dann $f7b8
. f7aa  bd 8e f8  lda $f88e,x   Holt Symbol aus Tabelle
. f7ad  20 d2 ff  jsr $ffd2     (BSOUT) Gibt Zeichen aus
. f7b0  bd 94 f8  lda $f894,x   Holt 2. Symbol aus Tabelle
. f7b3  f0 03     beq $f7b8     Symbol gleich Null ?, dann nicht ausgeben
. f7b5  20 d2 ff  jsr $ffd2     (BSOUT) Gibt Zeichen aus
```

```
. f7b8  ca       dex          Schon alle Zeichen ausgegeben ?
. f7b9  d0 d2    bne $f78d    Nein, dann $f78d
. f7bb  60       rts          ENDE

    ZIELADRESSE FÜR BRANCH-BEFEHL

. f7bc  20 c8 f7 jsr $f7c8    Addiert Opcode zur Adresse
. f7bf  18       clc          Addiert 1
. f7c0  69 01    adc #$01     ... zum Ergebnis
. f7c2  d0 01    bne $f7c5
. f7c4  e8       inx
. f7c5  4c ff fa jmp $faff    Gibt Zieladresse aus

. f7c8  a6 a2    ldx $a2      Holt Adresse (High)
. f7ca  a8       tay          Operand kleiner als 128 ?
. f7cb  10 01    bpl $f7ce    Ja, dann $f7ce
. f7cd  ca       dex          Sonst: Rückwärtssprung
. f7ce  65 a1    adc $a1      Addiert Operand + 1 zur Adresse (Low)
. f7d0  90 01    bcc $f7d3    Kein Übertrag ?, dann $f7d3
. f7d2  e8       inx          Erhöht Adresse (High) um 1
. f7d3  60       rts          ENDE

    OPCODE ANALYSIEREN

. f7d4  a8       tay          Holt Befehls-Code
. f7d5  4a       lsr          Bit 0 gesetzt ?
. f7d6  90 0b    bcc $f7e3    Nein, dann $f7e3
. f7d8  4a       lsr          Bit 1 gesetzt ?
. f7d9  b0 17    bcs $f7f2    Ja, dann Fehler
. f7db  c9 22    cmp #$22     Opcode $89 ?
. f7dd  f0 13    beq $f7f2    Ja, dann Fehler
. f7df  29 07    and #$07
. f7e1  09 80    ora #$80
. f7e3  4a       lsr
. f7e4  aa       tax
. f7e5  bd 3d f8 lda $f83d,x  Holt Offset für Adressierungs-Flag
. f7e8  b0 04    bcs $f7ee    Bit 0 gesetzt ?, dann unteres Halbbyte
. f7ea  4a       lsr          Schiebt
. f7eb  4a       lsr          ... oberes
. f7ec  4a       lsr          ... Halbbyte
. f7ed  4a       lsr          ... nach unten
. f7ee  29 0f    and #$0f     Isoliert unteres Halbbyte
. f7f0  d0 04    bne $f7f6    Ergebnis ungleich 0 ?, dann OK
. f7f2  a0 80    ldy #$80     Einsprung bei Fehler
. f7f4  a9 00    lda #$00
. f7f6  aa       tax
. f7f7  bd 81 f8 lda $f881,x  Holt Adressierungs-Flag
. f7fa  8d 4b 05 sta $054b    Speichert Adressierungs-Flag
. f7fd  29 03    and #$03     Addiert 3
. f7ff  85 f6    sta $f6      Schreibt Operandenlänge
. f801  98       tya          Holt Opcode nach Akku
. f802  29 8f    and #$8f
```

```
. f8Ø4  aa       tax
. f8Ø5  98       tya
. f8Ø6  aØ Ø3    ldy #$Ø3      Berechnet
. f8Ø8  eØ 8a    cpx #$8a      ... Offset
. f8Øa  fØ Øb    beq $f817     ... des
. f8Øc  4a       lsr           ... Opcodes
. f8Ød  9Ø Ø8    bcc $f817
. f8Øf  4a       lsr
. f81Ø  4a       lsr
. f811  Ø9 2Ø    ora #$2Ø
. f813  88       dey
. f814  dØ fa    bne $f81Ø
. f816  c8       iny
. f817  88       dey
. f818  dØ f2    bne $f8Øc
. f81a  6Ø       rts           ENDE

      MNEMO ZUR OPCODE-AUSGABE

. f81b  a8       tay           Holt Opcode-Offset nach Y-Reg.
. f81c  b9 9b f8 lda $f89b,y   Holt Byte aus Tabelle
. f81f  85 9f    sta $9f       Schreibt Byte nach $9f
. f821  b9 db f8 lda $f8db,y   Holt 2. Byte aus Tabelle
. f824  85 aØ    sta $aØ       Schreibt Byte nach $aØ
. f826  a9 ØØ    lda #$ØØ
. f828  aØ Ø5    ldy #$Ø5      Folgende Routine wandelt
. f82a  Ø6 aØ    asl $aØ       ... die beiden Bytes $aØ und $9f
. f82c  26 9f    rol $9f       ... in ein Mnemo
. f82e  2a       rol           ... mit 3 Buchstaben um
. f82f  88       dey           ... Dabei ergeben 5 Bit
. f83Ø  dØ f8    bne $f82a     ... einen Buchstaben
. f832  69 3f    adc #$3f      Ergibt Buchstabencode oder '?' bei Ø !
. f834  2Ø d2 ff jsr $ffd2     (BSOUT) Zeichen ausgeben
. f837  ca       dex
. f838  dØ ec    bne $f826
. f83a  4c Ø8 fb jmp $fbØ8     Leerzeichen ausgeben

. f83d  :
. f83d  *** TABELLEN ***
. f83d  :

      OFFSET FÜR ADRESSIERUNGS-FLAGS

. f83d 4Ø Ø2 45 Ø3 dØ Ø8 4Ø Ø9
. f845 3Ø 22 45 33 dØ Ø8 4Ø Ø9
. f84d 4Ø Ø2 45 33 dØ Ø8 4Ø Ø9
. f855 4Ø Ø2 45 b3 dØ Ø8 4Ø Ø9
. f85d ØØ 22 44 33 dØ 8c 44 ØØ
. f865 11 22 44 33 dØ 8c 44 9a
. f86d 1Ø 22 44 33 dØ Ø8 4Ø Ø9
. f875 1Ø 22 44 33 dØ Ø8 4Ø Ø9
. f87d 62 13 78 a9
```

```
      ADRESSIERUNGS-FLAGS

. f881 ØØ 21 81 82 ØØ ØØ 59 4d
. f889 91 92 86 4a 85 9d

      SYMBOL-TABELLE 1

. f88f 2c 29 2c 23 28 24       .byte ',),#($'

      SYMBOL-TABELLE 2

. f895 59 ØØ 58 24 24 ØØ       .byte 'y' $ØØ 'x$$' $ØØ

      MNEMO-TABELLE 1

. f89b 1c 8a 1c 23 5d 8b 1b a1
. f8a3 9d 8a 1d 23 9d 8b 1d a1
. f8ab ØØ 29 19 ae 69 a8 19 23
. f8b3 24 53 1b 23 24 53 19 a1
. f8bb ØØ 1a 5b 5b a5 69 24 24
. f8c3 ae ae a8 ad 29 ØØ 7c ØØ
. f8cb 15 9c 6d 9c a5 69 29 53
. f8d3 84 13 34 11 a5 69 23 aØ

      MNEMO-TABELLE 2

. f8db d8 62 5a 48 26 62 94 88
. f8e3 54 44 c8 54 68 44 e8 94
. f8eb ØØ b4 Ø8 84 74 b4 28 6e
. f8f3 74 f4 cc 4a 72 f2 a4 8a
. f8fb ØØ aa a2 a2 74 74 74 72
. f9Ø3 44 68 b2 32 b2 ØØ 22 ØØ
. f9Øb 1a 1a 26 26 72 72 88 c8
. f913 c4 ca 26 48 44 44 a2 c8

. f91b Ød 2Ø 2Ø 2Ø

. f91f  :
. f91f  *** ASSEMBLE (A/.) ***
. f91f  :
. f91f  9Ø Ø3    bcc $f924    Adresse angegeben ?, dann $f924
. f921  4c 92 f4 jmp $f492    Eingabeschleife

. f924  2Ø 5b fb jsr $fb5b    Kopiert Adresse nach Zeiger ($a1-$a2)
. f927  a2 ØØ    ldx #$ØØ
. f929  86 78    stx $78
. f92b  2Ø 3f fb jsr $fb3f    Holt Zeichen aus Puffer
. f92e  dØ Ø7    bne $f937    Kein Trennzeichen ?, dann $f937
. f93Ø  eØ ØØ    cpx #$ØØ     Trennzeichen am Puffer-Anfang ?
. f932  dØ Ø3    bne $f937    Nein, dann $f937
. f934  4c 95 f4 jmp $f495    Eingabeschleife
```

```
. f937  c9 20     cmp #$20      Zeichen gleich Leerzeichen ?
. f939  f0 ec     beq $f927     Ja, dann überlesen
. f93b  9d 4c 05  sta $054c,x   Kopiert
. f93e  e8        inx           ... Befehlswort
. f93f  e0 03     cpx #$03      ... nach
. f941  d0 e8     bne $f92b     ... ($054c-$054e)
. f943  ca        dex
. f944  30 12     bmi $f958
. f946  bd 4c 05  lda $054c,x   Jedes Byte des Befehlswortes
. f949  38        sec           ... wird in den Bereich $01-$1b
. f94a  e9 3f     sbc #$3f      ... transformiert
. f94c  a0 05     ldy #$05      ... Die Buchstaben benötigen so nur noch
. f94e  4a        lsr           ... 5 Bit und können bequem
. f94f  66 78     ror $78       ... in 2 Byte
. f951  66 77     ror $77       ... $77-$78
. f953  88        dey           ... untergebracht
. f954  d0 f8     bne $f94e     ... werden
. f956  f0 eb     beq $f943     Springt immer nach $f943

. f958  a2 02     ldx #$02
. f95a  20 3f fb  jsr $fb3f     Holt Zeichen aus Puffer
. f95d  f0 1e     beq $f97d     Zeichen gleich Trennzeichen ?, dann $f97d
. f95f  c9 20     cmp #$20      Zeichen gleich Leerzeichen ?
. f961  f0 f7     beq $f95a     Ja, dann überlesen
. f963  20 7d fa  jsr $fa7d     Zeichen gleich HEX-Ziffer ?
. f966  b0 0e     bcs $f976     Nein, dann $f976
. f968  20 8b fa  jsr $fa8b     Wandelt 2 HEX-Ziffern in Byte
. f96b  a4 f1     ldy $f1       Vorheriges Byte
. f96d  84 f2     sty $f2       ... hochschieben
. f96f  85 f1     sta $f1       ... und neues Byte speichern
. f971  a9 30     lda #$30      Holt '0'-Code
. f973  95 77     sta $77,x
. f975  e8        inx
. f976  95 77     sta $77,x
. f978  e8        inx
. f979  e0 0a     cpx #$0a
. f97b  90 dd     bcc $f95a
. f97d  86 9f     stx $9f       Merkt sich Ende-Offset
. f97f  a2 00     ldx #$00
. f981  8e 4f 05  stx $054f     Schreibt Test-Code
. f984  a2 00     ldx #$00
. f986  8e 50 05  stx $0550     Schreibt Vergleichs-Offset
. f989  ad 4f 05  lda $054f     Holt Test-Code
. f98c  20 d4 f7  jsr $f7d4     Disassembliert Test-Code
. f98f  ae 4b 05  ldx $054b     Holt Adressierungsflag
. f992  86 a0     stx $a0       Flag sichern
. f994  aa        tax           Holt Opcode-Offset
. f995  bd db f8  lda $f8db,x   Holt Bytes aus Mnemo-Tabelle 2
. f998  20 5e fa  jsr $fa5e     Vergleicht beide Bytes
. f99b  bd 9b f8  lda $f89b,x   Holt Bytes aus Mnemo-Tabelle 1
. f99e  20 5e fa  jsr $fa5e     Vergleicht beide Bytes
. f9a1  a2 06     ldx #$06      Beide Bytes waren identisch!
```

```
. f9a3  e0 03    cpx #$03
. f9a5  d0 13    bne $f9ba
. f9a7  a4 f6    ldy $f6       Operandenlänge gleich Null ?
. f9a9  f0 0f    beq $f9ba     Ja, dann $f9ba
. f9ab  ad 4b 05 lda $054b     Holt Adressierungs-Flag
. f9ae  c9 e8    cmp #$e8      Handelt es sich um einen Branch-Befehl ?
. f9b0  a9 30    lda #$30
. f9b2  b0 1e    bcs $f9d2     Ja, dann $f9d2
. f9b4  20 5b fa jsr $fa5b     Vergleich Test-Code mit Eingabe
. f9b7  88       dey           Verringert Operandenlänge um 1
. f9b8  d0 f1    bne $f9ab     Noch nicht 0 ?, dann $f9ab
. f9ba  0e 4b 05 asl $054b     Schiebt Adressierungs-Flag bitweise ins Carry
. f9bd  90 0e    bcc $f9cd     Bit nicht gesetzt ?, dann $f9cd
. f9bf  bd 8e f8 lda $f88e,x   Holt Symbol aus Tabelle
. f9c2  20 5e fa jsr $fa5e     Symbol mit Test-Code vergleichen
. f9c5  bd 94 f8 lda $f894,x   Holt 2. Symbol aus Tabelle
. f9c8  f0 03    beq $f9cd     Kein 2. Symbol nötig ?, dann $f9cd
. f9ca  20 5e fa jsr $fa5e     Symbol mit Test-Code vergleichen
. f9cd  ca       dex
. f9ce  d0 d3    bne $f9a3
. f9d0  f0 06    beq $f9d8     Springt immer nach $f9d8

. f9d2  20 5b fa jsr $fa5b     Vergleich Test-Code mit Eingabe
. f9d5  20 5b fa jsr $fa5b     Vergleich Test-Code mit Eingabe
. f9d8  a5 9f    lda $9f       Holt Ende-Offset
. f9da  cd 50 05 cmp $0550     Ende-Offset gleich Vergleichs-Offset ?
. f9dd  f0 03    beq $f9e2     Ja, dann $f9e2
. f9df  4c 6a fa jmp $fa6a     Erhöht Test-Code um 1

. f9e2  a4 f6    ldy $f6       Operandenlänge gleich Null ?
. f9e4  f0 34    beq $fa1a     Ja, dann $fa1a
. f9e6  a5 a0    lda $a0       Holt Adressierungs-Flag
. f9e8  c9 9d    cmp #$9d      Branch-Befehl ?
. f9ea  d0 26    bne $fa12     Nein, dann $fa12
. f9ec  a5 f1    lda $f1       Subtrahiert
. f9ee  e5 a1    sbc $a1       ... Befehlsadresse
. f9f0  8d 51 05 sta $0551     ... von Zieladresse.
. f9f3  a5 f2    lda $f2       ... Ergebnis gleich
. f9f5  e5 a2    sbc $a2       ... Operand
. f9f7  90 09    bcc $fa02     Ergebnis negativ ?, dann Rückwärtssprung
. f9f9  d0 77    bne $fa72     Ergebnis ungleich 0 ?, dann fertig
. f9fb  ae 51 05 ldx $0551     Holt Operand
. f9fe  30 72    bmi $fa72     Operand negativ ?, dann fertig
. fa00  10 09    bpl $fa0b     Springt immer nach $fa0b

. fa02  a8       tay           Holt High-Byte
. fa03  c8       iny           High-Byte gleich $ff ?
. fa04  d0 6c    bne $fa72     Nein, dann fertig
. fa06  ae 51 05 ldx $0551     Holt Low-Byte
. fa09  10 67    bpl $fa72     Low-Byte kleiner 128 ?, dann fertig
. fa0b  ca       dex
. fa0c  ca       dex
```

```
. fa0d  8a       txa
. fa0e  a4 f6    ldy $f6       Holt Operandenlänge
. fa10  d0 03    bne $fa15     Länge ungleich Null ?, dann $fa15
. fa12  b9 f0 00 lda $00f0,y   Holt Operand
. fa15  91 a1    sta ($a1),y   Schreibt Operand an Adresse
. fa17  88       dey           Erniedrigt Operandenlänge
. fa18  d0 f8    bne $fa12     Noch nicht Null ?, dann weiter kopieren
. fa1a  ad 4f 05 lda $054f     Holt Opcode
. fa1d  91 a1    sta ($a1),y   Schreibt Opcode an Adresse
. fa1f  20 35 fb jsr $fb35     Setzt Cursor an Zeilenanfang zurück
. fa22  a2 28    ldx #$28      Gibt Monitor-Meldung
. fa24  20 66 cf jsr $cf66     ... 'A 'aus
. fa27  20 5a f7 jsr $f75a     Gibt Adresse aus
. fa2a  e6 f6    inc $f6       Befehlslänge gleich
. fa2c  a5 f6    lda $f6       ... Operandenlänge plus 1
. fa2e  20 96 fb jsr $fb96     Erhöht Adresse um Befehlslänge
. fa31  a9 41    lda #$41      Schreibt 'A'
. fa33  8d 27 05 sta $0527     ... an 1. Stelle des Tastaturpuffers
. fa36  a9 20    lda #$20      Schreibt Leerzeichen ' '
. fa38  8d 28 05 sta $0528     ... an 2. und
. fa3b  8d 2d 05 sta $052d     ... 7. Stelle des Tastaturpuffers
. fa3e  a5 a2    lda $a2       Wandelt High-Byte
. fa40  20 20 fb jsr $fb20     ... in 2 HEX-Ziffern
. fa43  8d 29 05 sta $0529     ... und schreibt diese an 3.
. fa46  8e 2a 05 stx $052a     ... und 4. Stelle des Tastaturpuffers
. fa49  a5 a1    lda $a1       Wandelt Low-Byte
. fa4b  20 20 fb jsr $fb20     ... in 2 HEX-Ziffern
. fa4e  8d 2b 05 sta $052b     ... und schreibt diese an 5.
. fa51  8e 2c 05 stx $052c     ... und 6. Stelle des Tastaturpuffers
. fa54  a9 07    lda #$07      Setzt Tastaturpuffer-Index
. fa56  85 ef    sta $ef       ... auf 7 Zeichen
. fa58  4c 95 f4 jmp $f495     Eingabeschleife

      VERGLEICH VON TEST-CODE MIT EINGABE

. fa5b  20 5e fa jsr $fa5e     Vergleicht Bytes
. fa5e  8e f3 07 stx $07f3     Rettet X-Register
. fa61  ae 50 05 ldx $0550     Holt Vergleichs-Offset
. fa64  d5 77    cmp $77,x     Vergleicht Akku mit Eingabe
. fa66  f0 0d    beq $fa75     Beide Bytes identisch ?, dann $fa75
. fa68  68       pla           Entfernt Rücksprungadresse
. fa69  68       pla           ... vom Stack
. fa6a  ee 4f 05 inc $054f     Erhöht Test-Code um 1
. fa6d  f0 03    beq $fa72     Alle Codes getestet ?, dann fertig
. fa6f  4c 84 f9 jmp $f984     Testet nächsten Code

. fa72  4c 92 f4 jmp $f492     Eingabeschleife

. fa75  e8       inx           Erhöht Vergleichs-Offset um 1
. fa76  8e 50 05 stx $0550     Schreibt Vergleichs-Offset
. fa79  ae f3 07 ldx $07f3     Holt X-Register
. fa7c  60       rts           ENDE
```

```
      PRÜFT AUF HEX-ZIFFER

. fa7d  c9 41     cmp #$41      Zeichen gleich 'a' ?
. fa7f  90 03     bcc $fa84
. fa81  c9 47     cmp #$47      Zeichen gleich 'g' ?
. fa83  60        rts           ENDE

. fa84  c9 30     cmp #$30      Zeichen gleich '0' ?
. fa86  90 16     bcc $fa9e
. fa88  c9 3a     cmp #$3a      Zeichen gleich ':' ?
. fa8a  6       rts           ENDE

      WANDELT 2 HEX-ZIFFERN IN 1 BYTE

. fa8b  20 a0 fa jsr $faa0    Holt 1. Ziffer ins untere Halbbyte
. fa8e  0a        asl           Schiebt
. fa8f  0a        asl           ... unteres
. fa90  0a        asl           ... Halbbyte
. fa91  0a        asl           ... nach oben
. fa92  8d 5c 05 sta $055c     Ergebnis zwischenspeichern
. fa95  20 3f fb jsr $fb3f    Holt Zeichen aus Puffer
. fa98  20 a0 fa jsr $faa0    Holt 2. Ziffer ins untere Halbbyte
. fa9b  0d 5c 05 ora $055c     Kombiniert beide Halbbytes zu einem Byte
. fa9e  38        sec
. fa9f  60        rts           ENDE

. faa0  c9 3a     cmp #$3a      Zeichen gleich Dezimalziffer ?
. faa2  08        php           Antwort merken
. faa3  29 0f     and #$0f      Isoliert unteres Halbbyte
. faa5  28        plp           Zeichen gleich Dezimalziffer ?
. faa6  90 02     bcc $faaa     Ja, dann fertig
. faa8  69 08     adc #$08      Addiert 8 + 1(Carry) => a-f := 10-15
. faaa  60        rts           ENDE

. faab  :
. faab  *** ADRESSE ÜBERNEHMEN ***
. faab  :
. faab  c6 f3     dec $f3       Erniedrigt Puffer-Zeiger um 1
. faad  a9 00     lda #$00      Initialisiert
. faaf  85 f1     sta $f1       ... Adressen-
. fab1  85 f2     sta $f2       ... speicher
. fab3  8d f4 07 sta $07f4     Initialisiert Ziffernzähler
. fab6  20 3f fb jsr $fb3f    Holt Zeichen aus Puffer
. fab9  f0 3a     beq $faf5     Zeichen gleich Trennzeichen ?, dann $faf5
. fabb  c9 20     cmp #$20      Zeichen gleich Leerzeichen ?
. fabd  f0 f7     beq $fab6     Ja, dann überlesen
. fabf  c9 20     cmp #$20      Zeichen gleich Leerzeichen ?
. fac1  f0 2e     beq $faf1     Ja, dann fertig
. fac3  c9 2c     cmp #$2c      Zeichen gleich Komma ?
. fac5  f0 2a     beq $faf1     Ja, dann fertig
. fac7  c9 30     cmp #$30      Zeichen gleich HEX-Ziffer ?
```

```
. fac9  90 2b    bcc $faf6    Nein, dann Fehler
. facb  c9 47    cmp #$47     Zeichen gleich HEX-Ziffer ?
. facd  b0 27    bcs $faf6    Nein, dann Fehler
. facf  c9 3a    cmp #$3a     Zeichen gleich HEX-Ziffer ?
. fad1  90 06    bcc $fad9    Ja, dann $fad9
. fad3  c9 41    cmp #$41     Zeichen gleich HEX-Ziffer ?
. fad5  90 1f    bcc $faf6    Nein, dann Fehler
. fad7  e9 08    sbc #$08     Verschiebt Zahl
. fad9  e9 2f    sbc #$2f     ... in den Bereich 0-15
. fadb  0a       asl          Schiebt
. fadc  0a       asl          ... Halbbyte
. fadd  0a       asl          ... nach
. fade  0a       asl          ... oben
. fadf  a2 04    ldx #$04     Rotiert
. fae1  0a       asl          ... die 4 Bit
. fae2  26 f1    rol $f1      ... des oberen
. fae4  26 f2    rol $f2      ... Halbbytes
. fae6  ca       dex          ... in den Adressenspeicher
. fae7  d0 f8    bne $fae1    ... ein
. fae9  ee f4 07 inc $07f4    Erhöht Ziffernzähler um 1
. faec  20 3f fb jsr $fb3f    Holt Zeichen aus Puffer
. faef  d0 ce    bne $fabf    Keine Endmarke ?, dann nächste Ziffer bearbeiten
. faf1  ad f4 07 lda $07f4    Holt Anzahl der Ziffern
. faf4  18       clc
. faf5  60       rts          ENDE

. faf6  68       pla          Holt Rücksprungadresse
. faf7  68       pla          ... vom Stack
. faf8  4c 92 f4 jmp $f492    Fehler und Eingabeschleife

. fafb  :
. fafb  *** ADRESSE AUSGEBEN ***
. fafb  :
. fafb  a5 a1    lda $a1      Holt Adresse (Low)
. fafd  a6 a2    ldx $a2      Holt Adresse (High)
. faff  48       pha          Retten Low-Byte auf Stack
. fb00  8a       txa          Holt High-Byte nach Akku
. fb01  20 10 fb jsr $fb10    Gibt Byte als 2 HEX-Ziffern aus
. fb04  68       pla          Holt Low-Byte vom Stack
. fb05  20 10 fb jsr $fb10    Gibt Byte als 2 HEX-Ziffern aus
. fb08  a9 20    lda #$20     Holt Leerzeichen

. fb0a  2c       .byte $2c

. fb0b  a9 3f    lda #$3f     Holt Fragezeichen (?)
. fb0d  4c d2 ff jmp $ffd2    (BSOUT) Gibt Zeichen aus

     HEX-ZAHL AUSGEBEN

. fb10  8e f3 07 stx $07f3    Rettet X-Register
. fb13  20 20 fb jsr $fb20    Wandelt Byte in 2 HEX-Ziffern
. fb16  20 d2 ff jsr $ffd2    (BSOUT) Gibt 1. Ziffer aus
```

```
. fb19  8a       txa          Holt 2. Ziffer nach Akku
. fb1a  ae f3 Ø7 ldx $Ø7f3    Holt X-Register
. fb1d  4c d2 ff jmp $ffd2    (BSOUT) Gibt 2. Ziffer aus

     BYTE IN 2 HEX-ZIFFERN WANDELN

. fb2Ø  48       pha          Legt Byte auf Stack
. fb21  2Ø 2a fb jsr $fb2a    Wandelt unteres Halbbyte in HEX-Ziffer
. fb24  aa       tax          2. HEX-Ziffer nach X-Reg.
. fb25  68       pla          Holt Byte vom Stack
. fb26  4a       lsr          Schiebt
. fb27  4a       lsr          ... oberes Halbbyte
. fb28  4a       lsr          ... nach
. fb29  4a       lsr          ... unten
. fb2a  29 Øf    and #$Øf     Isoliert unteres Halbbyte
. fb2c  c9 Øa    cmp #$Øa     Byte kleiner als 1Ø ?
. fb2e  9Ø Ø2    bcc $fb32    Ja, dann $fb32
. fb3Ø  69 Ø6    adc #$Ø6     Sonst 7 addieren (Carry=1)
. fb32  69 3Ø    adc #$3Ø     Erzeugt ASCII-Code
. fb34  6Ø       rts          ENDE

. fb35  a9 91    lda #$91     Holt 'CURSOR UP'-Code
. fb37  2Ø d2 ff jsr $ffd2    (BSOUT) Gibt Zeichen aus
. fb3a  a9 Ød    lda #$Ød     Holt CR-Code
. fb3c  4c d2 ff jmp $ffd2    (BSOUT) Gibt Zeichen aus

     LIEST ZEICHEN AUS PUFFER

. fb3f  8e f3 Ø7 stx $Ø7f3    X-Register zwischenspeichern
. fb42  a6 f3    ldx $f3      Holt Pufferzeiger
. fb44  e4 f4    cpx $f4      Pufferzeiger gleich Puffer-Ende ?
. fb46  bØ Øf    bcs $fb57    Ja, dann $fb57
. fb48  bd ØØ Ø2 lda $Ø2ØØ,x  Holt Zeichen aus Puffer
. fb4b  c9 3a    cmp #$3a     Zeichen gleich Trennzeichen ?
. fb4d  fØ Ø8    beq $fb57    Ja, dann $fb57
. fb4f  e6 f3    inc $f3      Erhöht Pufferzeiger um 1
. fb51  Ø8       php
. fb52  ae f3 Ø7 ldx $Ø7f3    X-Register wiederherstellen
. fb55  28       plp
. fb56  6Ø       rts          ENDE

. fb57  a9 ØØ    lda #$ØØ     Code für 'Trennzeichen'
. fb59  fØ f6    beq $fb51    Springt immer nach $fb51

     ADRESSE IN ZEIGER ($A1-$A2) KOPIEREN

. fb5b  a5 f1    lda $f1      Kopiert
. fb5d  85 a1    sta $a1      ... Adresse ($f1-$f2)
. fb5f  a5 f2    lda $f2      ... in
. fb61  85 a2    sta $a2      ... Zeiger ($a1-$a2)
. fb63  6Ø       rts          ENDE
```

```
      ZEIGER ($A1-$A2) VON ADRESSE SUBTRAHIEREN

. fb64  38       sec          Subtrahiert
. fb65  a5 f1    lda $f1      ... Zeiger ($a1-$a2)
. fb67  e5 a1    sbc $a1      ... von
. fb69  85 f1    sta $f1      ... Adresse ($f1-$f2)...
. fb6b  a5 f2    lda $f2
. fb6d  e5 a2    sbc $a2
. fb6f  85 f2    sta $f2
. fb71  60       rts          ENDE

      ADRESSE ERNIEDRIGEN

. fb72  a9 01    lda #$01     Adresse um 1 erniedrigen      (EINSPRUNG)
. fb74  8d f3 07 sta $07f3    Adresse um (AKKU) erniedrigen (EINSPRUNG)
. fb77  38       sec          Erniedrigt
. fb78  a5 f1    lda $f1      ... Adresse ($f1-$f2)
. fb7a  ed f3 07 sbc $07f3    ... um
. fb7d  85 f1    sta $f1      ... den
. fb7f  a5 f2    lda $f2      ... Wert
. fb81  e9 00    sbc #$00     ... der
. fb83  85 f2    sta $f2      ... Speicherstelle $07f3
. fb85  60       rts          ENDE

      BLOCKLÄNGE ERNIEDRIGEN

. fb86  38       sec          Erniedrigt
. fb87  a5 9f    lda $9f      ... Blocklänge
. fb89  e9 01    sbc #$01     ... um 1...
. fb8b  85 9f    sta $9f
. fb8d  a5 a0    lda $a0
. fb8f  e9 00    sbc #$00
. fb91  85 a0    sta $a0
. fb93  60       rts          ENDE

      ZEIGER ERHÖHEN

. fb94  a9 01    lda #$01     Zeiger um 1 erhöhen      (EINSPRUNG)
. fb96  18       clc          Zeiger um (AKKU) erhöhen (EINSPRUNG)
. fb97  65 a1    adc $a1      Addiert
. fb99  85 a1    sta $a1      ... Akku
. fb9b  90 02    bcc $fb9f    ... zum
. fb9d  e6 a2    inc $a2      ... Zeiger ($a1-$a2)
. fb9f  60       rts          ENDE

      ADRESSE UND BLOCKLÄNGE HOLEN

. fba0  b0 14    bcs $fbb6    Keine Adresse ?, dann $fbb6
. fba2  20 5b fb jsr $fb5b    Kopiert Adresse in Zeiger ($a1-$a2)
. fba5  20 ad fa jsr $faad    Übernimmt Adresse
. fba8  b0 0c    bcs $fbb6    Keine Adresse ?, dann $fbb6
. fbaa  20 64 fb jsr $fb64    Subtrahiert 1. Adresse von 2. Adresse
```

```
. fbad  a5 f1     lda $f1       Schreibt
. fbaf  85 9f     sta $9f       ... Ergebnis
. fbb1  a5 f2     lda $f2       ... als Blocklänge
. fbb3  85 aØ     sta $aØ       ... nach $9f-$aØ
. fbb5  18        clc
. fbb6  6Ø        rts           ENDE

      REGISTER RETTEN

. fbb7  8d 1Ø Ø1  sta $Ø11Ø     Rettet Akku
. fbba  8e 12 Ø1  stx $Ø112     Rettet X-Register
. fbbd  8c 11 Ø1  sty $Ø111     Rettet Y-Register
. fbcØ  6Ø        rts           ENDE

      REGISTER WIEDERHERSTELLEN

. fbc1  ad 1Ø Ø1  lda $Ø11Ø     Holt Akku
. fbc4  ae 12 Ø1  ldx $Ø112     Holt X-Register
. fbc7  ac 11 Ø1  ldy $Ø111     Holt Y-Register
. fbca  6Ø        rts           ENDE

      STOP-TASTE PRÜFEN

. fbcb  86 fa     stx $fa       X-Register zwischenspeichern
. fbcd  2Ø 11 cf  jsr $cf11     Dekodiert STOP-Taste
. fbdØ  a6 fa     ldx $fa       Holt X-Register
. fbd2  49 8Ø     eor #$8Ø      War STOP-Taste gedrückt,
. fbd4  Øa        asl           ... so wird Carry gesetzt
. fbd5  a9 ØØ     lda #$ØØ
. fbd7  6Ø        rts           ENDE

. fbd8  :
. fbd8  *** MELDUNG AUSGEBEN ***
. fbd8  :
. fbd8  48        pha           Rettet Akku auf Stack
. fbd9  98        tya           Rettet Y-Register
. fbda  48        pha           ... auf Stack
. fbdb  8a        txa           Rettet X-Register
. fbdc  48        pha           ... auf Stack
. fbdd  ba        tsx           Berechnet
. fbde  e8        inx           ... Position
. fbdf  e8        inx           ... der
. fbeØ  e8        inx           ... Rücksprungadresse
. fbe1  e8        inx           ... im Stack
. fbe2  bd ØØ Ø1  lda $Ø1ØØ,x   Kopiert
. fbe5  85 bc     sta $bc       ... Rücksprungadresse
. fbe7  e8        inx           ... vom
. fbe8  bd ØØ Ø1  lda $Ø1ØØ,x   ... Stack
. fbeb  85 bd     sta $bd       ... nach $bc-$bd
. fbed  e6 bc     inc $bc       Erhöht Zeiger
. fbef  dØ Ø2     bne $fbf3     ... um 1...
. fbf1  e6 bd     inc $bd       ... Ergebnis gleich Anfang der Meldung
```

```
. fbf3  aØ ØØ     ldy #$ØØ      Initialisiert Offset in Meldung
. fbf5  b1 bc     lda ($bc),y   Holt Zeichen aus Meldung
. fbf7  fØ Ø6     beq $fbff     Endmarke ?, dann $fbff
. fbf9  2Ø d2 ff  jsr $ffd2     (BSOUT) Gibt Zeichen aus
. fbfc  c8        iny           Erhöht Offset in Meldung um 1
. fbfd  dØ f6     bne $fbf5     Springt immer nach $fbf5

. fbff  98        tya           Holt Länge der Meldung nach Akku
. fcØØ  ba        tsx           Berechnet
. fcØ1  e8        inx           ... die
. fcØ2  e8        inx           ... Rücksprungadresse
. fcØ3  e8        inx           ... im
. fcØ4  e8        inx           ... Stack
. fcØ5  18        clc           Addiert
. fcØ6  65 bc     adc $bc       ... Länge der Meldung
. fcØ8  9d ØØ Ø1  sta $Ø1ØØ,x   ... zur Anfangsadresse
. fcØb  a9 ØØ     lda #$ØØ      ... der Meldung
. fcØd  65 bd     adc $bd       ... Ergebnis gleich
. fcØf  e8        inx           ... neue
. fc1Ø  9d ØØ Ø1  sta $Ø1ØØ,x   ... Rücksprungadresse!!
. fc13  68        pla           Holt X-Register
. fc14  aa        tax           ... vom Stack
. fc15  68        pla           Holt Y-Register
. fc16  a8        tay           ... vom Stack
. fc17  68        pla           Holt Akku vom Stack
. fc18  6Ø        rts           ENDE

. fc19  :
. fc19  *** IOBASE ***
. fc19  :
. fc19  a2 ØØ     ldx #$ØØ      Holt Basisadresse der
. fc1b  aØ fd     ldy #$fd      ... Ein-/Ausgaberegister ($fdØØ)
. fc1d  6Ø        rts           ENDE

. fc1e  :
. fc1e  *** MODUL-RESET ***
. fc1e  :
. fc1e  a2 Ø3     ldx #$Ø3      Initialisiert
. fc2Ø  86 96     stx $96       ... Modulzeiger
. fc22  a9 ØØ     lda #$ØØ      Löscht
. fc24  9d ec Ø5  sta $Ø5ec,x   ... Modul-
. fc27  ca        dex           ... Tabelle...
. fc28  1Ø fa     bpl $fc24
. fc2a  a6 96     ldx $96       Holt Modul-Zeiger
. fc2c  bd 7b fc  lda $fc7b,x   Holt $ØØ, $Ø5, $Øa, $Øf
. fc2f  aa        tax           Holt $ØØ, $Ø5, $Øa, $Øf nach X-Reg.
. fc3Ø  9d dØ fd  sta $fddØ,x   Schaltet Modul ein
. fc33  aØ Ø2     ldy #$Ø2      Sucht nach
. fc35  b9 Ø7 8Ø  lda $8Ø07,y   ... 'cbm'-Code
. fc38  d9 56 fc  cmp $fc56,y   ... im Bereich $8Ø07-$8ØØ9
. fc3b  dØ 14     bne $fc51     Nicht vorhanden ?, dann $fc51
. fc3d  88        dey
```

```
. fc3e  1Ø f5    bpl $fc35
. fc4Ø  ad Ø6 8Ø lda $8ØØ6    Holt Modul-Nummer (Basic-Interpreter: Ø)
. fc43  a6 96    ldx $96      Holt Modul-Zeiger
. fc45  9d ec Ø5 sta $Ø5ec,x  Schreibt Modul-Nummer in Tabelle
. fc48  c9 Ø1    cmp #$Ø1     Modul-Nummer 1 ?
. fc4a  dØ Ø5    bne $fc51    Nein, dann $fc51
. fc4c  86 fb    stx $fb      Schreibt Modul-Zgr. in akt. Modul-Konfigurat.
. fc4e  2Ø ØØ 8Ø jsr $8ØØØ    Initialisierung des Moduls
. fc51  c6 96    dec $96      Erniedrigt Modul-Zeiger um 1
. fc53  1Ø d5    bpl $fc2a    Kein Unterlauf ?, dann $fc2a
. fc55  6Ø       rts          ENDE

. fc56  43 42 4d .byte 'cbm'

. fc59  :
. fc59  *** MODULE INITIALISIEREN ***
. fc59  :
. fc59  78       sei          Kein Interrupt zugelassen
. fc5a  a2 Ø3    ldx #$Ø3     Offset
. fc5c  bd ec Ø5 lda $Ø5ec,x  Modul (X-Reg.) vorhanden ?
. fc5f  fØ 1Ø    beq $fc71    Nein, dann $fc71
. fc61  8a       txa          Offset nach Akku
. fc62  48       pha          Rettet Offset auf Stack
. fc63  bd 7b fc lda $fc7b,x  Holt $ØØ, $Ø5, $Øa, $Øf
. fc66  aa       tax          Kopiert $ØØ, $Ø5, $Øa, $Øf nach X-Register
. fc67  9d dØ fd sta $fddØ,x  Schaltet Modul ein
. fc6a  86 fb    stx $fb      Schreibt Modul-Zgr. in akt. Modul-Konfigurat.
. fc6c  2Ø ØØ 8Ø jsr $8ØØØ    Initialisierung des Moduls
. fc6f  68       pla          Holt Offset
. fc7Ø  aa       tax          ... vom Stack
. fc71  ca       dex          Erniedrigt Offset um 1
. fc72  dØ e8    bne $fc5c    Noch nicht Null ?, dann $fc5c
. fc74  8d dØ fd sta $fddØ    Schaltet Basic-Interpreter ein
. fc77  86 fb    stx $fb      Schreibt Modul-Zgr. in akt. Modul-Konfigurat.
. fc79  58       cli          Interrupt zugelassen
. fc7a  6Ø       rts          ENDE

. fc7b  ØØ Ø5 Øa Øf           .byte $ØØ, $Ø5, $Øa, $Øf

. fc7f  :
. fc7f  *** MODUL-ZUGRIFF ***
. fc7f  :
. fc7f  9d dØ fd sta $fddØ,x  Schaltet Modul (X-Reg.) ein
. fc82  aa       tax          Übergibt ursprüngliches Modul an X-Reg.
. fc83  b1 be    lda ($be),y  Zugriff auf Modul
. fc85  9d dØ fd sta $fddØ,x  Schaltet ursprüngliches Modul wieder ein
. fc88  6Ø       rts          ENDE

. fc89  :
. fc89  *** MODUL-AUFRUF ***
. fc89  :
. fc89  48       pha          Rettet aktuelle Modul-Nummer auf Stack
```

```
. fc8a  86 fb     stx $fb       Schreibt Modul-Zgr. in akt. Modul-Konfigurat.
. fc8c  9d dØ fd sta $fddØ,x    Schaltet Modul (X-Reg.) ein
. fc8f  ae f3 Ø5 ldx $Ø5f3      Holt X-Register
. fc92  ad f4 Ø5 lda $Ø5f4      Holt Status-Register
. fc95  48        pha           ... Rettet Status-Reg. auf Stack
. fc96  ad f2 Ø5 lda $Ø5f2      Holt Akkumulator
. fc99  28        plp
. fc9a  2Ø bØ fc jsr $fcbØ      Routinenaufruf
. fc9d  8d f2 Ø5 sta $Ø5f2      Schreibt Akkumulator
. fcaØ  Ø8        php
. fca1  68        pla           Holt Status-Register vom Stack
. fca2  8d f4 Ø5 sta $Ø5f4      Schreibt Status-Register
. fca5  8e f3 Ø5 stx $Ø5f3      Schreibt X-Register
. fca8  68        pla           Holt ursprüngliche Modul-Nummer
. fca9  85 fb     sta $fb       Schreibt Modul-Zgr. in akt. Modul-Konfigurat.
. fcab  aa        tax           Schaltet ursprüngliches
. fcac  9d dØ fd sta $fddØ,x    ... Modul wieder ein
. fcaf  6Ø        rts           ENDE

. fcbØ  6c fØ Ø5 jmp ($Ø5fØ)    Routinenausruf

. fcb3  :
. fcb3  *** PULS ***
. fcb3  :
. fcb3  48        pha           Rettet Akku,
. fcb4  8a        txa           ... X-Register
. fcb5  48        pha           ... und
. fcb6  98        tya           ... Y-Register
. fcb7  48        pha           ... auf Stack
. fcb8  8d dØ fd sta $fddØ      Schaltet Basic ein
. fcbb  4c ØØ ce jmp $ceØØ      Interrupt-Routine
. fcbe  a6 fb     ldx $fb       Holt ursprüngliches Modul
. fccØ  9d dØ fd sta $fddØ,x    Schaltet ursprüngliches Modul ein
. fcc3  68        pla           Holt Akku,
. fcc4  a8        tay           ... X-Register
. fcc5  68        pla           ... und
. fcc6  aa        tax           ... Y-Register
. fcc7  68        pla           ... vom Stack
. fcc8  4Ø        rti           ENDE

. fcc9  :
. fcc9  *** MODUL EINSCHALTEN UND AUFRUFEN ***
. fcc9  :
. fcc9  a6 fb     ldx $fb       Holt Modul-Nummer
. fccb  9d dØ fd sta $fddØ,x    Schaltet Modul ein
. fcce  6c fe Ø2 jmp ($Ø2fe)    Ruft Modul auf

      DER BEREICH $FCD1-$FCFØ ENTHÄLT $FF

. fcf1  :
. fcf1  *** SPRUNGTABELLE 1 ***
. fcf1  :
```

```
. fcf1  4c c9 fc jmp $fcc9    Modul einschalten und aufrufen

. fcf4  4c 59 fc jmp $fc59    Vorhandene Module initialisieren

. fcf7  4c 7f fc jmp $fc7f    LDA ($be),Y aus Modul (X-Reg.)

. fcfa  4c 89 fc jmp $fc89    Modul-Aufruf

. fcfd  4c b8 fc jmp $fcb8    Interrupt-Routine

. fdØØ-ff3f      EIN-/AUSGABE- & TED-REGISTER

. $fdØØ-$fdØ3                 ACIA-Register (6551)
. $fd1Ø                       6529-Register
. $fddØ-$fddf                 Modul-Select
. $ffØØ-$ff3f                 TED-Register (736Ø)

      DER BEREICH $FF4Ø-$FF48 ENTHÄLT $FF

. ff49  :
. ff49  *** SPRUNGTABELLE 2 ***
. ff49  :
. ff49  4c c2 b7 jmp $b7c2    Key-Definition

. ff4c  4c 49 dc jmp $dc49    Print

. ff4f  4c d8 fb jmp $fbd8    Meldung ausgeben

. ff52  4c 45 f4 jmp $f445

      DER BEREICH $FF55-$FF7E ENTHÄLT $FF


-------------------------------------------------------------------------
. ff7f  8a 83    .byte $8a $83       (Alte Version)
-------------------------------------------------------------------------

. ff7f  5e 85    .byte $5e $85       (Neue Version)

 . ff81  :
. ff81  *** KERNEL-SPRUNGTABELLE ***
. ff81  :
. ff81  4c 4e d8 jmp $d84e    (CINT)     Editor Reset

. ff84  4c Øb f3 jmp $f3Øb    (IOINIT)   Initialisiert Ein-/Ausgabe-Reg.

. ff87  4c 52 f3 jmp $f352    (RAMTAS)   RAM-Test

. ff8a  4c ce f2 jmp $f2ce    (RESTOR)   Initialisiert Vektortabelle

. ff8d  4c d3 f2 jmp $f2d3    (VECTOR)   Verwaltung der RAM-Vektoren
```

```
. ff9Ø  4c 1a f4 jmp $f41a    (SETMSG)  Systemmeldung ein-/ausschalten
. ff93  4c 4d ee jmp $ee4d    (SECOND)  Übergibt Sekundäradr. für LISTEN
. ff96  4c 1a ee jmp $ee1a    (TKSA)    Gibt Sekundäradresse aus
. ff99  4c 27 f4 jmp $f427    (MEMTOP)  Obere RAM-Grenze setzen/holen
. ff9c  4c 36 f4 jmp $f436    (MEMBOT)  Untere RAM-Grenze setzen/holen
. ff9f  4c 11 db jmp $db11    (SCNKEY)  Tastaturabfrage
. ffa2  4c 23 f4 jmp $f423    (SETTMO)  Setzt Timeout-Flag für IEEE-Bus
. ffa5  4c 8b ec jmp $ec8b    (ACPTR)   Daten vom seriellen Bus lesen
. ffa8  4c df ec jmp $ecdf    (CIOUT)   Ausgabe auf Bus
. ffab  4c 3b ef jmp $ef3b    (UNTLK)   UNTALK senden
. ffae  4c 23 ef jmp $ef23    (UNLSN)   UNLISTEN senden
. ffb1  4c 2c ee jmp $ee2c    (LISTEN)  LISTEN senden
. ffb4  4c fa ed jmp $edfa    (TALK)    TALK senden
. ffb7  4c 1c f4 jmp $f41c    (READST)  Ein-/Ausgabe-Status lesen
. ffba  4c 13 f4 jmp $f413    (SETLFS)  Logische Datei einrichten
. ffbd  4c Øc f4 jmp $f4Øc    (SETNAM)  Filename übergeben
. ffcØ  6c 18 Ø3 jmp ($Ø318)  (OPEN)    Logische Datei öffnen
. ffc3  6c 1a Ø3 jmp ($Ø31a)  (CLOSE)   Logische Datei schließen
. ffc6  6c 1c Ø3 jmp ($Ø31c)  (CHKIN)   Setzt Eingabekanal
. ffc9  6c 1e Ø3 jmp ($Ø31e)  (CHKOUT)  Setzt Ausgabekanal
. ffcc  6c 2Ø Ø3 jmp ($Ø32Ø)  (CLRCH)   Schließt alle Ein-/Ausgabe-Kanäle
. ffcf  6c 22 Ø3 jmp ($Ø322)  (BASIN)   Zeicheneingabe
. ffd2  6c 24 Ø3 jmp ($Ø324)  (BSOUT)   Zeichenausgabe
. ffd5  4c 43 fØ jmp $fØ43    (LOAD)    Load oder Verify
. ffd8  4c 94 f1 jmp $f194    (SAVE)
. ffdb  4c 2d cf jmp $cf2d    (SETTIM)  Uhr stellen
```

```
. ffde  4c 26 cf jmp $cf26    (RDTIM)    Uhrzeit lesen

. ffe1  6c 26 Ø3 jmp ($Ø326)  (STOP)     Abfrage der STOP-Taste

. ffe4  6c 28 Ø3 jmp ($Ø328)  (GETIN)    Ein-/Ausgabe-Status lesen

. ffe7  6c 2a Ø3 jmp ($Ø32a)  (CLALL)    Schließt alle Dateien und Kanäle

. ffea  4c fØ ce jmp $cefØ    (UDTIM)    Systemuhr aktualisieren

. ffed  4c 34 d8 jmp $d834    (SCREEN)   Ermittelt Bildschirmformat

. fffØ  4c 39 d8 jmp $d839    (PLOT)     Cursorposition setzen/holen

. fff3  4c 19 fc jmp $fc19    (IOBASE)   Holt Basisadr. der I/O-Register

. fff6  :
. fff6  *** SYSTEM-START ***
. fff6  :
. fff6  8d 3e ff sta $ff3e    Schaltet ROM ein
. fff9  4c a4 f2 jmp $f2a4    NMI

. fffc  f6 ff     .adr $fff6  START-Adresse
. fffe  b3 fc     .adr $fcb3  Puls-Adresse
```

Anhang A: Stichwortverzeichnis der ROM-Routinen

Anhang B: Wichtige Betriebssystem-Adressen

$8000	BASIC-Start
$800a	WARMSTART
$8019	ALTSTART
$818e	Tabelle der BASIC-Keywörter
$8471	Tabelle der Fehlermeldungen
$867e	Ausgabe Fehlermeldung
$86c5	Meldung ausgeben (allgemein)
$870f	READY. Indirekter Einsprung ($0302)
$88c0	Blocktransport
$8967	USER-Token generieren
$89d4	USER-Token einbauen
$8a79	NEW
$8a98	CLR
$8aff	LIST
$8b85	USER-Token erzeugen
$8bbc	RUN
$8bd3	Hauptschleife des BASIC-Interpreters
$8c9a	RESTORE
$8cc0	STOP (Tastatur-Abfrage)
$8cd7	STOP
$8cda	END
$8d03	CONT
$8d2c	GOSUB
$8d4d	GOTO
$8d83	RETURN
$8db0	DATA
$8de1	IF
$8e0b	ELSE

$8e0b REM
$8e1b ON
$8e7c LET
$8eb7 TI$ ZUWEISUNG
$8fe0 PRINT#
$8fe6 CMD
$9000 PRINT
$904f TAB (10er-Tabulator)
$905f SPC
$905f TAB
$9088 STRING-Ausgabe
$90a6 Zeichenausgabe
$90b8 GET
$90cb GET#
$90ee INPUT#
$9108 INPUT
$914f READ
$9294 NEXT
$9314 (FRMNUM) Numerischen Ausdruck auswerten
$9317 (CHKNUM) Prüft ob numerisch
$931a (CHKSTR) Prüft auf String
$932c (FRMEVL) Ausdruck auswerten
$9414 (EVAL) Term Auswerten Ind.-Adr.: ($030a)
$9465 NOT
$9491 (CHKCOM) Prüft ob Komma folgt
$95f8 OR
$95fb AND
$969b DIM
$9a62 FRE
$9a7d POS
$9a9d DEF
$9acb FN (DEF FN)
$9b66 STR$
$9c48 (FRESTR)
$9cbb CHR$
$9ccf LEFT$
$9d03 RIGHT$
$9d15 MID$
$9d61 LEN
$9d70 ASC
$9d81 (GETBYT) Byte übernehmen
$9d93 VAL
$9dde (GETADR) Adresse holen

$9dfa	PEEK
$9e12	POKE
$9e1b	DEC
$9e6a	WAIT
$9e87	FAC := ARG - FAC
$9e96	Exponente von FAC gleich Exponent von ARG
$9e9b	FAC := Variable + FAC
$9e9e	FAC := ARG + FAC
$a01e	LOG
$a062	FAC := FAC + 0.5
$a06c	FAC := Konstante - FAC
$a072	FAC := Konstante / FAC
$a078	FAC := Variable * FAC
$a07b	FAC := ARG * FAC
$a0ae	Multiplikation
$a0dc	ARG := Konstante aus ROM
$a107	ARG := Variable aus RAM
$a137	Exponenten addieren
$a162	FAC := FAC * 10
$a183	FAC := FAC / 10
$a194	FAC := ARG / FAC
$a21f	FAC := (Akku, Y-Reg)
$a24c	FAC in Speicher kopieren
$a281	FAC := ARG
$a291	ARG := FAC
$a2a0	FAC Runden
$a2b0	Vorzeichen ermitteln
$a2be	SGN
$a2dd	ABS
$a327	Gleitkomma-Integer-Wandlung
$a358	INT
$a37f	String-Gleitkomma-Wandlung
$a453	IN ausgeben
$a45f	Integer aus Akku, X-Reg. ausgeben
$a46f	Gleitkomma-String-Wandlung
$a5e4	SQR
$a5ee	FAC := ARG hoch FAC
$a627	Vorzeichenwechsel von FAC
$a660	EXP
$a6b3	Polynomauswertung
$a707	RND
$a7b5	SYS
$a7de	SAVE

$a7f0	VERIFY
$a7f3	LOAD
$a84d	OPEN
$a85a	CLOSE
$a8f8	DS$ löschen
$a954	GARBAGE COLLECT
$aa70	COS
$aa77	SIN
$aac0	TAN
$ab1a	ATN
$ab8f	RENUMBER
$adca	FOR
$aef7	USING
$b386	INSTR
$b42b	TRAP
$b440	RESUME
$b4be	ERR$
$b507	HEX$
$b544	PUDEF
$b557	DO
$b5ac	EXIT
$b603	LOOP
$b652	TRON
$b655	TROFF
$b6cd	AUTO
$b6e8	HELP
$b729	KEY-Anzeige
$b7a7	KEY-Definition
$b849	SOUND
$b8bd	VOL
$b8d1	PAINT
$b9d4	CHAR
$bae2	BOX
$bd35	GSHAPE
$be29	SSHAPE
$bf79	RGR
$bf85	RCLR
$bf87	RLUM
$bfc1	JOY
$bffd	RDOT
$c01e	CIRCLE
$c0da	Strecke zeichnen
$c1a5	Doppelpunkt setzen

$c1c3	Punkt setzen
$c1f3	Punkt testen
$c21a	Farbe und Helligkeit setzen
$c246	Zeiger in Bitspeicher setzen
$c2ad	Zeile und Spalte nach (X-Reg., Y-Reg.)
$c2d3	SCALE ausführen
$c2f0	(Akku, Y-Reg.) := (Akku, Y-Reg.) +/- KO(X-Reg.)
$c2f4	(Akku, Y-Reg.) := KO(Y-Reg.) +/- KO(X-Reg.)
$c305	(Akku, Y-Reg.) := KO(Y-Reg.) - KO(X-Reg.)
$c318	(Akku, Y-Reg.) := KO(Y-Reg.)
$c322	(Akku, Y-Reg.) := ABS(KO(Y-Reg.) - KO(X-Reg.))
$c337	(Akku, Y-Reg.) := (Akku,Y-Reg) * KO(X-Reg) / 65536
$c37b	KO(0) := KO(2) & KO(4) := KO(6)
$c3b3	Farbparameter übernehmen
$c3d9	Koordinaten übernehmen
$c40b	Polarkoordinaten
$c45a	Rechtwinklige Koordinaten
$c4d9	DRAW
$c50f	LOCATE
$c51a	COLOR
$c567	SCNCLR
$c5b8	SCALE
$c5c3	GRAPHIC
$c7c9	Grafik abschalten
$c7bf	Grafik prüfen
$c7f0	Speicheraufteilung für Grafik
$c877	R-Zeiger korrigieren
$c8bc	DIRECTORY
$c941	DSAVE
$c951	DSAVE
$c968	HEADER
$c99c	SCRATCH
$c9cc	COLLECT
$c9da	COPY
$c9f6	RENAME
$ca00	BACKUP
$ca16	Kommandostring senden
$ca3f	Kommandostring erzeugen
$cccf	DS$ übernehmen
$cd2b	ARE YOU SURE?-Meldung
$cd57	DS$ löschen
$cdab	Ausgabe der Autorennamen
$ce00	Interrupt

$cecd Tongeneratoren bearbeiten
$cef0 (UDTIM) Uhr Takt
$cf26 Uhr lesen
$cf2d (SETTIM) Uhr stellen
$cf36 Monitor-Meldung
$cf66 Monitor-Meldung ausgeben
$d000 Zeichensatz
$d834 (SCREEN) Ermittelt Bildschirmformat
$d839 (PLOT) Cursorposition setzen/holen
$d88b Clear Screen
$d89a Home
$d8ea Eingabe vom Bildschirm
$d965 Zeichen vom Bildschirm holen
$d9ba Anführungszeichen-Modus setzen/löschen
$d9c7 Abschluß von PRINT
$da21 Scroll aufwärts
$da3d Bildschirmzeile umkopieren
$da5e Leerzeile einfügen
$db11 (SCNKEY) Tastaturabfrage
$db70 Decoder-Abfrage
$db7a Keylog
$dc49 PRINT
$dc8c RETURN-Code
$dc9b ESC-O
$dca7 Steuercodes
$dcfa Cursor right
$dd00 Cursor down
$dd0d Cursor up
$dd1c Cursor left
$dd27 Text-Modus
$dd2e SHIHT - C= - Sperre
$dd35 SHIFT - C= - Freigabe
$dd3e Grafik-Modus
$dd47 Bearbeitung der SHIFT-Codes
$dd99 DELETE
$ddce INSERT
$de06 ESC-Sequenz
$de48 ESC-R Bildschirmfenster mit Rand
$de5e ESC-T Window-Obergrenze
$de60 ESC-B Window-Untergrenze
$de8b ESC-I Zeile einfügen
$dea0 ESC-D Zeile löschen
$decb ESC-Q Zeile rechts vom Cursor löschen

$dee1 ESC-P Zeile links vom Cursor löschen
$def6 ESC-V Scroll auf
$df04 ESC-W Scroll ab
$df1d ESC-L Scrolling freigeben
$df20 ESC-M Scrolling sperren
$df26 ESC-C Auto-Insert ein
$df29 ESC-A Auto-Insert ein
$df39 Zeilentabelle
$df82 ESC-J Setzt Cursor an Zeilenanfang
$df95 ESC-K Setzt Cursor ans Zeilenende
$e153 (TALK)
$e156 (LISTEN)
$e181 Byte ausgeben
$e1f7 (SECOND) Übergibt Sekundäradr. für LISTEN
$e203 (TKSA) Gibt Sekundäradresse aus
$e21d (CIOUT) Ausgabe auf Bus
$e22f (UNTALK)
$e23d (UNLISTEN)
$e2b8 Bus-Ausgabe
$e2d4 Bus-Empfang
$e2dc Zeitschleifen
$e319 PLAY-Taste prüfen
$e364 Kassettenport einschalten
$e378 Kassettenport ausschalten
$e38d Motor einschalten
$e3b0 Motor ausschalten
$e3b7 Kassettenpuffer löschen
$e3c3 Pufferzeiger initialisieren
$e3ce Abbruch durch STOP-Taste (Kassettenoperation)
$e3e4 Timer setzen
$e413 Puls schreiben
$e447 Zeiten setzen
$e468 Bit 0 senden
$e474 Bit 1 senden
$e480 Startbit senden
$e48c Byte schreiben
$e4ba String schreiben
$e535 Puffer schreiben
$e56c Headerblock schreiben
$e5b0 Speicherbereich schreiben
$e5f0 EOT-Block schreiben
$e5fd Puls lesen
$e591 Bit lesen

$e6d5 Startbit suchen
$e6ec Byte lesen
$e74b String lesen
$e8d3 Block vom Bandpufffer lesen
$e8f3 Programmfile lesen
$e91d Lesen vorbereiten
$e9cc Fileheader lesen
$ea21 Header identifizieren
$ea5b RS-232-Routinen
$eaf1 RS-232-Eingabe
$eb37 RS-232-Ausgabe
$eb46 RS-232-Arbeitsbereich initialisieren
$ebc6 Ein-/Ausgabemeldungen ausgeben
$ebe8 (BASIN) Zeicheneingabe
$ebd9 (GETIN) Ein-/Ausgabe-Status lesen
$ec24 Byte von Kassette holen
$ec4b BSOUT) Zeichen ausgeben
$ec8b (ACPTR) Daten vom seriellen Bus lesen $ffa5
$ecdf (CIOUT) Ausgabe auf Bus
$ed18 (CHKIN) Öffnet Eingabekanal
$ed60 (CHKOUT) Öffnet Ausgabekanal
$eda9 IEC-Bus prüfen
$edd4 IEC-Abschluß
$edea Teile von IOINIT
$edfa (TALK)
$ee13 TKATN Gibt ATN aus
$ee1a (TKSA) Gibt Sekundäradresse aus
$ee2c (LISTEN)
$ee45 LSTATN Gibt ATN am seriellen Bus aus
$ee4d (SECOND) Übergibt Sekundäradr. für LISTEN
$ee5d (CLOSE) Logische Datei schließen
$ef08 (CLALL) Schließt alle Dateien und Kanäle
$ef0c (CLRCH) Schließt Ein-/Ausgabe-Kanäle
$ef23 (UNLISTEN)
$ef3b (UNTALK)
$ef53 (OPEN) Logische Datei öffnen
$f043 (LOAD) Load oder Verify
$f194 (SAVE)
$f265 (STOP) Abfrage der STOP-Taste
$f2a4 NMI
$f2ce (RESTOR) Initialisiert Vektortabelle
$f2d3 (VECTOR) Verwaltung der RAM-Vektoren
$f30b (IOINIT) Initialisiert I/O-Register

$f351 (RAMTAS) RAM-Test
$f40c (SETNAM) Filename übergeben
$f413 (SETLFS) Logische Datei einrichten
$f41a (SETMSG) Systemmeldung schalten
$f41c (READST) Ein-/Ausgabe-Status lesen
$f423 (SETTMO) Setzt Timeout-Flag für IEC-Bus
$f427 (MEMTOP) Obere RAM-Grenze setzen/holen
$f436 (MEMBOT) Untere RAM-Grenze setzen/holen
$f445 Monitor-Aufruf
$f44c Monitor-Break
$f478 Register-Befehl (R)
$f492 Monitor-Eingabeschleife
$f4d7 Memory-Dump (M)
$f50a Register ändern (;)
$f529 Memory-Dump editieren (>)
$f54b GOTO (G)
$f5ce Compare (C)
$f5d1 Transfer (T)
$f66e Load (L), Save (S), Verify (V)
$f70a Fill (F)
$f724 Disassemble (D)
$f91f Assemble (A/.)
$fbd8 Meldung ausgeben
$fc19 (IOBASE) Holt Basisadresse der Ein-/Ausgabe-Register
$fc1e Modul-Reset
$fc59 Module initialisieren
$fc7f Modul-Zugriff
$fc89 Modul-Aufruf
$fcb3 Puls
$fcc9 Modul einschalten und aufrufen
$fcf1 Modul einschalten und aufrufen
$fcf4 Module initialisieren
$fcf7 Modul-Zugriff
$fcfa Modul-Aufruf
$fcfd Interrupt-Routine
$ff49 Key-Definition
$ff4c PRINT
$ff4f Meldung ausgeben
$ff52 Monitor-Aufruf
$ff81 (CINT) Editor Reset
$ff84 (IOINIT) Initialisiert I/O-Register
$ff87 (RAMTAS) RAM-Test
$ff8a (RESTOR) Initialisiert Vektortabelle

$ff8d (VECTOR) Verwaltung der RAM-Vektoren
$ff90 (SETMSG) Systemmeldung schalten
$ff93 (SECOND) Übergibt Sekundäradr. für LISTEN
$ff96 (TKSA) Gibt Sekundäradresse aus
$ff99 (MEMTOP) Obere RAM-Grenze setzen/holen
$ff9c (MEMBOT) Untere RAM-Grenze setzen/holen
$ff9f (SCNKEY) Tastaturabfrage
$ffa2 (SETTMO) Setzt Timeout-Flag für IEC-Bus
$ffa5 (ACPTR) Daten vom seriellen Bus lesen
$ffa8 (CIOUT) Ausgabe auf Bus
$ffab (UNTALK)
$ffae (UNLISTEN)
$ffb1 (LISTEN)
$ffb4 (TALK)
$ffb7 (READST) Ein-/Ausgabe-Status lesen
$ffba (SETLFS) Logische Datei einrichten
$ffbd (SETNAM) Filename übergeben
$ffc0 (OPEN) Logische Datei öffnen
$ffc3 (CLOSE) Logische Datei schließen
$ffc6 (CHKIN) Öffnet Eingabekanal
$ffc9 (CHKOUT) Öffnet Ausgabekanal
$ffcc (CLRCH) Schließt Ein-/Ausgabe-Kanäle
$ffcf (BASIN) Zeicheneingabe
$ffd2 (BSOUT) Zeichenausgabe
$ffd5 (LOAD) Load oder Verify
$ffd8 (SAVE)
$ffdb (SETTIM) Uhr stellen
$ffde (RDTIM) Uhrzeit nach Akku, X-Reg. und Y-Reg.
$ffe1 (STOP) Abfrage der STOP-Taste
$ffe4 (GETIN) Ein-/Ausgabe-Status lesen ($a7af)
$ffe7 (CLALL) Schließt alle Dateien und Kanäle
$ffea (UDTIM) Uhr Takt
$ffed (SCREEN) Ermittelt Bildschirmformat
$fff0 (PLOT) Cursorposition setzen/holen
$fff3 (IOBASE) Holt Basisadresse der Ein-/Ausgabe-Register
$fff6 SYSTEMSTART

Weitere Fachbücher aus unserem Verlagsprogramm

COMMODORE 16/116

W. Besenthal/J. Muus

Alles über den C16

Juli 1986, 292 Seiten

Dieses Buch ist ein Lern- und Nachschlagewerk für jeden Commodore-Anwender. Es ist übersichtlich gegliedert und enthält alle Informationen, die für die praktische Arbeit am Computer notwendig sind: BASIC-Kurs mit Beispielen, Strukturiertes Programmieren, Dateiverwaltung, Grafikprogrammierung, Tips & Tricks.

Best.-Nr. MT 90385, ISBN 3-89090-385-1
(sFr. 35,90/öS 304,20) **DM 39,-**

COMMODORE 64

F. Ende

Das große Spielebuch - Commodore 64

1984, 141 Seiten

46 Spielprogramme · Wissenswertes über Programmiertechnik · praxisnahe Hinweise zur Grafikherstellung · alles über Joystick- und Paddleansteuerung · das Spielebuch mit Lerneffekt.

Best.-Nr. MT 603, ISBN 3-922120-63-6
(sFr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 604 (Beispiele auf Diskette)
(sFr. 38,-/öS 342,-) **DM 38,-***

* inkl. MwSt. Unverbindliche Preisempfehlung.

S. Krute

Grafik & Musik auf dem Commodore 64

1984, 336 Seiten

68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.

Best.-Nr. MT 743, ISBN 3-89090-033-X
(sFr. 35,-/öS 296,40) **DM 38,-**

H. L. Schneider/W. Eberl

Das C64-Profihandbuch

1985, 413 Seiten

Ein Buch, das alle wichtigen Informationen für professionelle Anwendungen mit dem C64 enthält. Mit allgemeinen Algorithmen, die auch auf andere Rechner übertragbar sind, und vielen Utilities, getrennt nach BASIC- und Maschinenprogrammen. Besonders nützlich: erweiterte PEEK- und POKE-Funktionen.

Best.-Nr. MT 749, ISBN 3-89090-110-7
(sFr. 47,80/öS 405,60) **DM 52,-**

W.-J. Becker/M. Folprecht

Programmieren unter CP/M mit dem C64

1985, 290 Seiten

Wenn Sie wissen wollen, wie das Betriebssystem CP/M-2.2 auf dem C64 implementiert ist, außerdem einiges über Turbo-Pascal, Nevada-Fortran, MBASIC-80 erfahren wollen, dann ist dieses Buch genau richtig für Sie! Mit Schaltplänen zur eigenen Fertigung des CP/M-Moduls. Für eingefleischte C64-Profis.

Best.-Nr. MT 751, ISBN 3-89090-091-7
(sFr. 47,80/öS 405,60) **DM 52,-**

J. Mihalik

35 ausgesuchte Spiele für Ihren Commodore 64

1984, 141 Seiten

Programmieren Sie selbst 35 faszinierende Spiele · geschrieben in Commodore-64-BASIC · mit Farbe, Grafiken und Ton · Vorschläge zur Programmabwandlung · für kreative Computerfans, die ihre Programmierkenntnisse vertiefen wollen!

Best.-Nr. MT 774, ISBN 3-89090-064-X
(sFr. 23,-/öS 193,40) **DM 24,80**

W. Kassera/F. Kassera

C64 - Programmieren in Maschinensprache

1985, 327 Seiten inklusive Beispieldiskette

In diesem Buch finden Sie über 100 Beispiele zur Assembler-Programmierung mit viel Kommentar und Hintergrundinformationen: Das Schreiben von Maschinenprogrammen · Rechnen und Texten mit vorhandenen Routinen · Bedienung von Drucker und Floppy · Wie man BASIC- und Maschinenprogramme verknüpft · Erstellen von eigenen Befehlen in Modulform. Für Profis!

Best.-Nr. MT 830, ISBN 3-89090-168-9
(sFr. 47,80/öS 405,60) **DM 52,-**

P. W. Dennis/G. Minter

Spiele für den Commodore 64

1984, 196 Seiten

Bewährte alte und raffinierte neue Spiele für Ihren Commodore 64 · klar und übersichtlich gegliederte Programme im Commodore-BASIC · Sie lernen: wie man Unterprogramme einsetzt · eine Tabelle aufbauen und verarbeiten · Programme testen · mit vielen Programmiertricks · für Anfänger.

Best.-Nr. MT 90074, ISBN 3-89090-074-7
(sFr. 23,-/öS 193,40) **DM 24,80**
Best.-Nr. MT 795 (Beispiele auf Diskette)
(sFr. 38,-/öS 342,-) **DM 38,-***

* inkl. MwSt. Unverbindliche Preisempfehlung.

K. Schramm

Die Floppy 1541

1985, 434 Seiten

Für alle Programmierer, die mehr über ihre VC-1541-Floppystation erfahren wollen. Der Vorgang des Formatierens · das Schreiben von Files auf Diskette · die Funktionsweise von schnellen Kopier- und Ladeprogrammen · viele fertige Programme · Lesen und Beschreiben von defekten Disketten · Für Einsteiger und für fortgeschrittene Maschinensprache-Programmierer.

Best.-Nr. MT 90098, ISBN 3-89090-098-4
(sFr. 45,10/öS 382,20) **DM 49,-**
Best.-Nr. MT 710 (Beispiele auf Diskette)
(sFr. 29,90/öS 269,10) **DM 29,90***

* inkl. MwSt. Unverbindliche Preisempfehlung.

Die angegebenen Preise sind Ladenpreise

607326/1

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

S. Baloui

C 64-Fischertechnik: Messen, Steuern, Regeln

Februar 1986, 174 Seiten

Ziel dieses Buches ist es, jedem Besitzer eines Commodore 64/VC 20 eine neue Welt zu erschließen: Die Welt der Roboter, der computergesteuerten Fertigungsstraßen. Alles, was Sie benötigen, ist einer der beiden genannten Computer und der Fischertechnik-Computing-Baukasten mit dazugehörigem Interface.

Best.-Nr. MT 90194, ISBN 3-89090-194-8
(sFr. 27,60/öS 233,20) **DM 29,90**

F. Matthes

Pascal mit dem C 64

Juni 1986,
215 Seiten inklusive Diskette

Buch und Compiler ermöglichen jedem Besitzer eines C 64 den Einstieg in die moderne Programmiersprache Pascal. Dem Anfänger wird ein Einführungskurs in Pascal geboten, wobei viele überschaubare Beispiele aus der Praxis und Übungsaufgaben zum aktiven Lernen mit dem C 64 auffordern.
Für den Pascal-Profi gibt es neben nützlichen Beispielprogrammen ein spezielles Kapitel mit Tips und Tricks.
Der Compiler akzeptiert den gesamten Sprachumfang mit einigen Erweiterungen. Übersetzte Programme laufen ohne weitere Hilfsprogramme auf jedem C 64, nutzen den gesamten Programmspeicher des C 64 und sind 3–4mal schneller als vergleichbare Programme in BASIC.

- Dem Buch liegt ein leistungsfähiges PASCAL-SYSTEM mit einigen Pascal-Programmen auf Diskette bei.

Best.-Nr. MT 90222, ISBN 3-89090-222-7
(sFr. 47,80/öS 405,60) **DM 52,-**

M. Hegenbarth/R. Trierscheid

BASIC-Grundkurs mit dem C 64

1985, 377 Seiten

Der Computerneuling kann mit diesem Buch lernen, mit seinem C 64 in BASIC zu arbeiten, und wird auf die Besonderheiten seines Computers hingewiesen. Dabei müssen nicht unendlich viele und umfangreiche Beispielprogramme mühsam abgetippt werden; es ist sogar denkbar, die Kapitel erst durchzulesen und das Gelernte dann am Computer auszuprobieren. Erwähnenswert ist auch ein Kapitel, welches die Kommunikation zweier C 64 beschreibt, und der Anhang, in dem neben der Kurzbeschreibung der reservierten Worte des BASIC V2 (mit Beispielen) eine Liste nützlicher PEEKs, POKEs und SYS und noch vieles mehr enthalten ist.

Best.-Nr. MT 90361, ISBN 3-89090-361-4
(sFr. 40,50/öS 343,20) **DM 44,-**

H. Ponnath

C 64: Wunderland der Grafik

1985, 232 Seiten inklusive Beispieldiskette

Dieses Buch zeigt eine Vielzahl sehr interessanter Lösungen, um die grafischen Möglichkeiten des Commodore 64 optimal zu nutzen. Als Krönung enthält es ein zuschaltbares Assemblerprogramm, das umfangreiche grafische und einige neue BASIC-Befehle anbietet. Im zweiten Teil des Buches wird eine Möglichkeit gezeigt, wie man bis zu 70 verschiedene Farben erzeugen kann. Viele Beispielprogramme begleiten die Reise durch das Wunderland der Grafik.

Best.-Nr. MT 90363, ISBN 3-89090-363-0
(sFr. 45,10/öS 382,20) **DM 49,-**

Commodore Sachbuch

Alles über den C 64

Juli 1986, 514 Seiten

Das umfangreiche Grundlagenbuch für den Commodore 64, ein nützliches Werkzeug, damit das künftige Programmieren auch Spaß macht: BASIC-Lexikon mit allen Befehlen, Anweisungen und Funktionen in alphabetischer Reihenfolge – Programmierung in Maschinensprache und Einbindung von Maschinensprache-Routinen in BASIC-Programme – Bestandteil des Betriebssystems: Das Kernal – Ein- und Ausgabeprogrammierung von SPRITES und Sonderzeichen – Erzeugung von Laufbildern in hochauflösender Farbgrafik – Musiksynthese und Klangeffekte – Betriebssystem CP/M sowie weitere anspruchsvolle Sprachen – GEOS.

Best.-Nr. MT 90379, ISBN 3-89090-379-7
(sFr. 54,30/öS 460,20) **DM 59,-**

R. West

C-64/SX-64-Computer-Handbuch

1985, 688 Seiten

Das Buch reicht von den professionellen Aspekten der BASIC-Programmierung (Entwicklung klarer und strukturierter Problemlösungen und/oder effizienter Programme) über sehr systemnahe Informationen (Änderungen am eingebauten BASIC, am Betriebssystem etc.) bis hin zur Hardware (Schnittstellen, Kassettengeräte, Floppy) und allen Fragen, die damit zusammenhängen. Besonders wichtig bei dieser Fülle an Informationen: der klare Aufbau des Buches, der den schnellen Zugriff auf die benötigte Information garantiert und so das Buch zur idealen Arbeitsgrundlage macht.

- Eine Enzyklopädie der Profi-Programmierung auf dem C 64.

Best.-Nr. PW 80324, ISBN 3-921803-24-1
(sFr. 60,70/öS 514,80) **DM 66,-**

R. Valentine

C-64-Programmsammlung 50 Lehr-, Spiel- und Nutzprogramme

1985, 200 Seiten

Praxisorientierte Programme und interessante Tips für den 64-User, der schon Erfahrungen mit seinem Computer gesammelt hat und sein Wissen (und auch seine Programmsammlung) erweitern möchte. PEEK, POKE, Bit- und Bytemanipulationen werden an ebenso leicht verständlichen Beispielen erklärt wie die Verwendung der eingebauten Zeituhr und der Sound- und Grafikfeatures Ihres C 64. Abgerundet wird die ganze Sache durch ein kleines Datenverwaltungsprogramm, einen Pilot-Interpreter (!) und viele Spiele. Sämtliche Programme sind in BASIC geschrieben und gut erklärt – somit auch leicht eigenen Anforderungen anzupassen.

Best.-Nr. PW 80346, ISBN 3-921803-46-2
(sFr. 27,50/öS 232,40) **DM 29,80**

Die angegebenen Preise sind Ladenpreise

607326/2

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

Reparaturanleitung Computer: C64 - Technische Servicedaten für Ihren Computer

Einzigartige Serviceunterlagen für Reparaturen und Entwicklungsarbeiten am C64. Enthält Schaltpläne, Bauteile- und Vergleichstypenliste; Prüfpunkte mit Oszillogrammen der Signalformen, Logiktabellen, Spannungsangaben; schnelle Servicetests, Anleitung zur systematischen Fehlersuche.

Best.-Nr. PW 80355, ISBN 3-921803-55-1
(sFr. 27,50/öS 232,40) **DM 29,80**

COMMODORE 128/128 D

G. Jürgensmeier

WordStar für den Commodore 128 PC

1985, 435 Seiten

WordStar ist ein umfangreiches und leistungsfähiges Textverarbeitungsprogramm und damit sicherlich zu Recht das meistverkaufte Programm seiner Art. Doch bedeutet dies nicht unbedingt, daß es auch einfach zu bedienen ist. Hier setzt dieses Buch an: Es macht in vorbildlicher Weise mit allen Möglichkeiten von WordStar und MailMerge vertraut und ist damit eine ideale Ergänzung zum Handbuch. Es versammelt alle wichtigen Informationen für den effektiven Einsatz dieser Programme auf dem Commodore 128 PC.

Best.-Nr. MT 780, ISBN 3-89090-181-6
(sFr. 45,10/öS 382,20) **DM 49,-**

Dr. P. Albrecht

Multiplan für den Commodore 128 PC

1985, 226 Seiten

Multiplan wurde ursprünglich für das 16-Bit-Betriebssystem MS-DOS entwickelt. Inzwischen ist aber auch die in diesem Buch beschriebene CP/M-Version für den Commodore 128 PC auf dem Markt, die den vollen Leistungsumfang der 16-Bit-Version enthält.
Das vorliegende Buch soll eine praktische Einführung in den Umgang mit Multiplan auf dem Commodore 128 PC geben. Anhand von praxisnahen Beispielen werden alle Befehle und Funktionen in der Reihenfolge beschrieben, die der Arbeit in der Praxis entspricht. Bereits nach Abschluß des ersten Kapitels werden Sie in der Lage sein, eigene kleine Multiplan-Anwendungen zu realisieren.

Best.-Nr. MT 836, ISBN 3-89090-187-5
(sFr. 45,10/öS 382,20) **DM 49,-**

Dr. P. Albrecht

dBASE II für den Commodore 128 PC

1985, 280 Seiten

Das vorliegende Buch gibt nach einer kurzen Einführung in den Komplex »Datenbanken« eine Anleitung für den praktischen Umgang mit dBASE II. Schon nach Beherrschung weniger Befehle ist der Anwender in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integrierter Reportgenerator, der im Dialog mit dem Benutzer Berichte gestaltet und in Tabellenform ausdruckt.

Best.-Nr. MT 838, ISBN 3-89090-189-1
(sFr. 45,10/öS 382,20) **DM 49,-**

H. Haberl

Mini-CAD mit Hi-Eddi plus auf dem C64/C128

1985, 230 Seiten inklusive Diskette

Neben den »Standardbefehlen« zum Setzen und Löschen von Punkten, dem Zeichnen von Linien, Kreisen und Rechtecken sowie dem Ausfüllen unregelmäßiger Flächen und dem Verschieben und Duplizieren von Bildschirmbereichen bietet Hi-Eddi eine Reihe von Besonderheiten, die dieses Programm von anderen Grafikprogrammen abheben: Bis zu sieben Grafikbildschirme stehen gleichzeitig zur Verfügung; es besteht die Möglichkeit, Text in die Grafik einzufügen, die Bildschirme zu verknüpfen oder in schneller Folge durchzuschalten.

Best.-Nr. MT 90136, ISBN 3-89090-136-0
(sFr. 44,20/öS 374,40) **DM 48,-**

J. Hückstädt

BASIC 7.0 auf dem Commodore 128

1985, 239 Seiten

Ganz gleich, ob Sie bereits über Programmierkenntnisse verfügen oder nicht, dieses Buch wird Ihnen helfen, den größtmöglichen Nutzen aus dem leistungsstarken BASIC 7.0 des Commodore 128 PC zu ziehen. Sie eignen sich bei der Durcharbeitung dieses Buches alle notwendigen Kenntnisse an, um immer anspruchsvollere Aufgabenstellungen zu bewältigen: Listenverarbeitung, indexsequentielle Dateiverwaltung, Grafikdarstellungen und Sounderzeugung. Ein unentbehrliches Lehrbuch, das sich auch für den geübten Anwender als Nachschlagewerk eignet.

Best.-Nr. MT 90149, ISBN 3-89090-149-2
(sFr. 47,80/öS 405,60) **DM 52,-**

K. Schramm

Die Floppy 1570/1571

Juni 1986, 470 Seiten

Dieses Buch soll es sowohl dem Einsteiger als auch dem fortgeschrittenen Programmierer ermöglichen, die vielfältigen Möglichkeiten dieses neuen Gerätes voll auszuschöpfen. Sämtliche Betriebsarten und Diskettenformate werden ausführlich erläutert. Anhand vieler Beispiele werden Sie in die Dateiverwaltung mit dieser Floppy eingeführt. Der Benutzer lernt die zahlreichen Systembefehle kennen und erfährt zugleich wichtige Grundlagen für das Arbeiten mit dem Betriebssystem CP/M.

Best.-Nr. MT 90185, ISBN 3-89090-185-9
(sFr. 47,80/öS 405,60) **DM 52,-**

P. Rosenbeck

Das Commodore-128-Handbuch

1985, 383 Seiten

In diesem Buch finden Sie einen Querschnitt durch alle wichtigen Funktions- und Anwendungsbereiche des Commodore 128. Sie werden mit dem C64/C128-Modus und der Benutzung von CP/M-3.0 vertraut gemacht, erfahren alles über die Grafik- und Soundmöglichkeiten des C128, lernen die Techniken der Speicherverwaltung und das Banking kennen und werden in die Programmierung mit Assemblersprache sowie die Grafikprogrammierung des 80-Zeichen-Bildschirms eingeführt. Ein umfassendes Handbuch, das Sie immer griffbereit haben sollten!

Best.-Nr. MT 90195, ISBN 3-89090-195-6
(sFr. 47,80/öS 405,60) **DM 52,-**

Die angegebenen Preise sind Ladenpreise

607326/3

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

Prof. Dr. Wolf-Jürgen Becker

CP/M-3.0-Anwender-Handbuch C128

Mai 1986, 250 Seiten

Das Buch sagt Ihnen alles über den Aufbau einer Datenverarbeitungsanlage, Mikrocomputer, Programmiersprachen und Betriebssysteme im allgemeinen und über das Betriebssystem CP/M speziell auf dem C128 PC. Ausführliche Beschreibungen der CP/M-Befehle und ihrer Funktionen fehlen ebensowenig wie die umfassende Darstellung der Struktur von CP/M-3.0 auf dem C128. Im Kapitel über das Programmieren unter CP/M erfahren Sie dann, wie man das CP/M-Betriebssystem ändert, kommerzielle Software installiert und mit ihr arbeitet.

Best.-Nr. MT 90196, ISBN 3-89090-196-4
(sFr. 47,80/öS 405,60) **DM 52,-**

H. Ponnath

Grafik-Programmierung C128

März 1986, 196 Seiten

Eine Hilfe für den Einsteiger und eine Fundgrube von Anregungen für den Profi soll dieses Buch sein. Das Themenfeld ist weit gespannt und umfaßt unter anderem:
Mehrfarbengrafik im C128-Modus; die Programmierung von Sprites und Shapes; Assembler-Makros, die die Grafikprogrammierung unterstützen; die beiden Video-Chips des C128; Erzeugung selbstmodifizierender Programme; Generierung von Fractals, ein besonders spannendes Thema im Bereich mathematischer Grafikanwendungen.
Die ungewöhnlichen Grafikfähigkeiten des C128 werden voll ausgeschöpft.

Best.-Nr. MT 90202, ISBN 3-89090-202-2
(sFr. 47,80/öS 405,60) **DM 52,-**

G. Möllmann

C128-Programmieren in Maschinensprache

August 1986, ca. 250 Seiten

Dieses Buch ist für alle diejenigen geschrieben, die die Fähigkeiten ihres Commodore 128 voll ausschöpfen wollen, um selbst erfolgreich auf dem Commodore 128 programmieren zu können. Dazu gehört außer der Beschreibung der im 128er enthaltenen Bausteingruppen auch der Umgang mit den ROM-Routinen aus Basic und Betriebssystem.

- Eine Fundgrube für jeden ernsthaften C128-Programmierer!

Best.-Nr. MT 90213, ISBN 3-89090-213-8
(sFr. 47,80/öS 405,60) **DM 52,-**

R. Schineis/M. Braun

C128-ROM-Listing: BASIC-7.0-Betriebssystem

August 1986, ca. 300 Seiten

Nach einer Einführung in die Arbeitsweise des C128 werden der interne Aufbau und die Wirkungsweise des BASIC-Interpreters erläutert. Es wird hierbei unter anderem auf die Speicher- und Variablen-Organisation sowie auf die Struktur und Ablage von BASIC-Zeilen eingegangen. Vor dem Hauptteil, einem vollständig kommentierten Assemblerlisting des C-128-BASIC-Interpreters mit Cross-Referenzliste (Verweistabelle), werden Informationen über die Struktur und Interpretation des Listings und der Verweistabelle gegeben.

Best.-Nr. MT 90220, ISBN 3-89090-220-0
(sFr. 45,10/öS 382,20) **DM 49,-**

R. Schineis/M. Braun/N. Demgensky

C128-ROM-Listing: Operating System

März 1986, 450 Seiten

Dieses Buch ist für alle Programmierer und Anwender gedacht, die mehr über ihren Commodore 128 PC wissen wollen: Eine Einführung in die Organisation und Wirkungsweise eines Mikrocomputers sowie eine detaillierte Beschreibung der Mikroprozessorfamilie 65XX bzw. 8502, Aufbau und spezielle Hardwareeigenschaften des C128 mit Beispielprogrammen. Ein umfangreiches, vollständig kommentiertes Assemblerlisting mit Cross-Referenzliste (Verweistabelle) umfaßt das komplette Betriebssystem mit dem 40/80-Zeichen-Editor sowie allen Kernel-Routinen.

Best.-Nr. MT 90221, ISBN 3-89090-221-9
(sFr. 45,10/öS 382,20) **DM 49,-**

COMMODORE AMIGA

M. Breuer

Das AMIGA-Handbuch

März 1986, 461 Seiten

Das Buch liefert übersichtlich gegliedertes Grundwissen über die neue Commodore-Maschine. Aus dem Inhalt:
Vorhang auf: Der AMIGA! · Auf der Werkbank des AMIGA · Grundlage der Bedienung des AMIGA · Grafik mit Graficraft und Delux Paint. AMIGA für Fortgeschrittene: Das CLI · Automatisierung des AMIGA · Die Spezialchips des AMIGA · Grundlagen von Sound und Grafik.

- Mit vielen Abbildungen und Übersichtstafeln für den täglichen Einsatz.

Best.-Nr. MT 90228, ISBN 3-89090-228-6
(sFr. 45,10/öS 382,20) **DM 49,-**

COMMODORE PC 10/PC 20

D. A. Lien

BASIC-Programmierung PC10/PC 20

1985, 488 Seiten

Ein amerikanisch-lockerer BASIC-Kurs von dem kalifornischen Professor Lien. Durch seine Systematik ideal als Kursunterlage für PC 10/20 und Kompatible. Mit Einführung in das PC-10-System und Tastendarstellung im Text.

Best.-Nr. PW 80366, ISBN 3-921803-66-7
(sFr. 54,30/öS 460,20) **DM 59,-**

Die angegebenen Preise sind Ladenpreise

607326/4

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Christian Quirin Spitzner

C 16, C 116 und Plus/4 ROM-Listing

Der Autor:

CHRISTIAN QUIRIN SPITZNER, geboren 1965 in München, studiert zur Zeit Elektrotechnik. Mit Computern beschäftigt er sich seit 1980. Im Rahmen seiner Tätigkeit als freier Mitarbeiter beim Markt & Technik-Verlag ist sein Computerpark inzwischen auf sieben Stück angestiegen.
Mit Vorliebe programmiert er in Maschinensprache, wobei ihm meist ein ROM-Listing des betreffenden Computers Hilfestellung leistet. Da es aber bis jetzt für den C 16, C 116 und Plus/4 keine Literatur dieser Art gab, machte er sich selbst daran, ein ROM-Listing zu erstellen.

Ein ROM-Listing ist die kommentierte Wiedergabe der Systemsoftware eines Computers. Die drei Computer C 16, C 116 und Plus/4 besitzen bis auf geringfügige Unterschiede denselben ROM-Bereich, der insgesamt 32 Kbyte (Anwendersoftware des Plus/4 nicht mitgerechnet) umfaßt. Darin sind folgende Komponenten enthalten:
- Betriebssystem
- Monitor (TED-Mon)
- BASIC-Interpreter

Der Besitz eines ROM-Listings ist für all die von Interesse, ja geradezu eine Notwendigkeit, die ihren C 16, C 116 oder Plus/4 in Maschinensprache programmieren wollen. Ein dokumentiertes ROM-Listing gibt Aufschluß über die interne Arbeitsweise. Es dient außerdem als wichtiges Anschauungsmaterial für eigene Assemblerprogramme. Die wichtigsten Routinen, die sogenannten Kernel-Routinen, werden gesondert behandelt. Ausführlich erklärt und mit den richtigen Einsprungadressen, ermöglichen sie effektives Programmieren in Maschinensprache.
Darüber hinaus sind für alle Tüftler die Zeropage-Adressen von Bedeutung. Deshalb ist dieser Teil des ROM besonders ausführlich dokumentiert. Hier befinden sich alle Zeiger, Flags und wichtige Systemadressen. Mit ihrer Kenntnis eröffnen sich ungeahnte Wege in der Manipulation von internen Programmabläufen.
Zwei Stichwortregister im Anhang sind für die schnelle, sachbezogene Orientierung im ROM-Bereich und in der Zeropage gedacht.

Hardware-Anforderungen:
- Commodore-16- oder
- Commodore-116- oder
- Plus/4-Computersystem

ISB N 3-89090-425-4

DM 49,-
sFr 45,10
öS 382,20