Markt&Technik

R. Schineis, U. M. Braun, N. Demgensky

C128

ROM-Listing Operating System

Ausführlich dokumentiertes Listing
aller Betriebssystem-Routinen.
Mit umfangreicher Beschreibung der Prozessor
Input-/Output-Architektur und Befehlssatz
sowie Assemblerprogrammierung.

Mit Crossreferenz-listen

C128 ROM-Listing: Operating System

Die Autoren:

RUDOLF SCHINEIS, geboren 1947 in Bayreuth, aus dem Hause HARD + SOFT, seit über 10 Jahren in der Ausbildung tätig, befaßte sich mit Mikrocomputern und Maschinenprogrammierung schon zu Zeiten des Ur-PET ca. 1977. Autor der Betriebssystem-Listings für Commodore-Geräte CBM-8000, CBM-700, C64 und Floppy 8250.

MICHAEL BRAUN, geboren 1948 in Saarbrücken, unterrichtet seit 1976 Informatik in Berlin, auch ein Ur-PET-Maschinenprogrammierer. Autor der Betriebssystem-Listings für Commodore-Geräte CBM-8000, CBM-700, C64 und Floppy 8250.

NORBERT DEMGENSKY, geboren 1951 in Berlin, seit über 10 Jahren in der Groß-EDV als Programmierer tätig, beschäftigt sich seit 1977 professionell mit Mikrocomputern.

Dieses Buch enthält in konzentrierter Form umfassende Informationen über den Aufbau des Commodore-128-Computers. Es soll sich daher schnell zu einem wichtigen Arbeitsbuch für die Programmierung entwickeln.
Zum optimalen Verständnis der internen Struktur des C128 wird dem Leser eine detaillierte Beschreibung der Mikroprozessor-Familie »65xx« bzw. »8502« mit Einblick in die Organisation, den Befehlen und Adressierungsarten geboten. An diese allgemeinen Ausführungen schließt ein Kapitel mit den speziellen Hardware-Eigenschaften an. Am konkreten Beispiel sehen Sie Funktionsweise, Anwendung und Beziehung der leistungsfähigen C-128-Bausteine wie z.B. VIC.

Der Hauptteil des Buches umfaßt ein komplett kommentiertes Listing des Betriebssystems mit dem 40/80-Zeichen-Editor, dem Maschinensprache-Monitor sowie allen Kernal-Routinen, die u.a. der Steuerung der Peripherie-Bausteine dienen. Eine Crossreferenzliste (Verweistabelle) erleichtert das Auffinden gesuchter Routinen, einer Nutzung der C-128-ROM-Routinen für eigene Programme steht nun nichts im Wege.
Ein vollständig dokumentiertes ROM-Listing des BASIC-Interpreters des C128 erscheint als gesondertes Buch in gleicher Form und Aufbau unter Bestell-Nummer MT 90220 im Markt&Technik Buchverlag.

Hardwareanforderung:
Commodore 128 oder Commodore 128 D

In diesem Buch finden Sie:

- Organisation und Arbeitsweise des C128: Adreß-, Daten-, Steuerbus, ROM, RAM, I/O-Bausteine, grundsätzliche Arbeitsweise,
- die Hardware des C 128: Prozessor, Peripherie, VIC, SID, MMU, VDC, CIA, Bildschirmeditor, Kernal, umfangreiche Programmbeispiele,
- ein vollständig dokumentiertes strukturiertes ROM-Listing des C-128-Betriebssystems mit Crossreferenzliste inkl. Markierung aller Definitionen und sich ändernden Speicherstellen,
- umfangreiche Tabellen und Anhänge zum Nachschlagen.

ISB N 3-89090-221-9

4 001057 902213

Markt&Technik
Verlag Aktiengesellschaft

DM 49,-
sFr. 45,10
öS 382,20

R. Schineis
U. M. Braun
N. Demgensky

C128 ROM-Listing Operating System

Ausführlich dokumentiertes Listing aller Betriebssystem-Routinen. Mit umfangreicher Beschreibung der Prozessor Input-/Output-Architektur und Befehlssatz sowie Assemblerprogrammierung.

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Schineis, Rudolf:
C128 ROM-Listing — Operating System : ausführl. dokumentiertes Listing aller Betriebssystem-Routinen ; mit umfangreicher Beschreibung d. Prozessor-Input-/Output-Architektur u. Befehlssatz sowie Assemblerprogrammierung / R. Schineis ; U. M. Braun ; N. Demgensky. — Haar bei München : Markt-und-Technik-Verlag, 1986.
ISBN 3-89090-221-9
NE: Braun, U. M.:; Demgensky, Norbert:

Die Informationen im vorliegenden Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Buch gezeigten Modelle und Arbeiten ist nicht zulässig.

»Commodore 128 Personal Computer« ist eine Produktbezeichnung der Commodore Büromaschinen GmbH, Frankfurt, die ebenso wie der Name »Commodore« Schutzrecht genießt. Der Gebrauch bzw. die Verwendung bedarf der Erlaubnis der Schutzrechtsinhaberin.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86

ISBN 3-89090-221-9

© 1986 by Markt & Technik, 8013 Haar bei München
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Inhaltsverzeichnis

1	**Der Mikrocomputer C-128**	9
1.1	Einführung in die Hardware des C-128	9
1.1.1	Zentral-Prozessor	9
1.1.1.1	Interne Register der 8502-CPU	10
1.1.2	Peripherie-Bausteine	11
1.1.3	VIC - Videocontroller für den 40-Zeichen-Bildschirm	11
1.1.3.1	Interne Register des Videocontrollers VIC 8564	12
1.1.4	SID - Klangerzeugungs-Baustein	15
1.1.4.1	Interne Register des Sound Interface Device SID 6581	15
1.1.4.2	Funktionen der SID-Register	15
1.1.5	MMU - Speicherverwaltungs-Baustein	26
1.1.5.1	Interne Register der Memory Management Unit MMU 8722	26
1.1.5.2	Indirekte RAM-Routinen	30
1.1.6	VDC - Videocontroller für den 80-Zeichen-Bildschirm	33
1.1.6.1	Interne Register des Video Display Controllers VDC 8563	35
1.1.7	CIA - Ein-/Ausgabe-Baustein	38
1.1.7.1	Interne Register des Complex Interface Adapter 1 CIA 6526	43
1.1.7.2	Interne Register des Complex Interface Adapter 2 CIA 6526	46
1.2	Das Betriebssystem des C-128	47
1.2.1	Einführung	47
1.2.2	Der Monitor	48
1.2.3	Der Bildschirm-Editor	53
1.2.4	Das Kernal	65
1.3	Beispielprogramme	80
1.3.1	Unterprogramme zur VDC-Steuerung	80
1.3.2	Unterprogramme zum Datentransfer	88
1.3.3	Beispielprogramm zur CIA-Behandlung	92
2	**Grundsätzliches zum Assembler und den Assemblerlistings**	99
2.1	Einführung in die Assemblerprogrammierung	99
2.2	Erklärung des Assemblerlistings	102
2.2.1	Zeilennummern-Feld	102
2.2.2	Hex-Adressen-Feld	102
2.2.3	Objekt-Code-Feld	102
2.2.4	Quell-Code-Feld	102
2.2.4.1	Deklaration und Zuweisung von Label	102
2.2.4.2	Linksbündiges Label	103
2.2.4.3	Mnemonics	103
2.2.4.4	Assembler-Direktiven	103

2.2.4.5	Operand	104
2.2.5	Kommentar-Feld	104
2.3	Crossreferenzliste bzw. Verweistabelle	105
2.3.1	Handhabungsbeispiel	106
3	**Der C-128-Monitor**	107
3.1	Kommentiertes Assemblerlisting	107
3.2	Verweistabelle (Crossreference)	151
4	**Editor und Kernal des C-128**	157
4.1	Kommentiertes Assemblerlisting	157
4.1.1	Editor-/Operating-Sytem-Labels	157
4.1.2	CPU 8502 I/O-Port	166
4.1.3	VIC 8564 Video Controller	167
4.1.4	SID 6581 Sound Interface Device	170
4.1.5	MMU 8722 Memory Management Unit	171
4.1.6	VDC 8563 Video Display Controller	174
4.1.7	CIA 6526 Complex Interface Adapter	177
4.1.8	Bildschirm-Editor	184
4.1.9	Tastatur-Abfrage	205
4.1.10	Bildschirm-Editor (Fortsetzung)	237
4.1.11	Initialisierung	249
4.1.12	Floppy (Serial Device)	267
4.1.13	RS232-Schnittstelle	278
4.1.14	Kassettenrekorder	294
4.1.15	Kernal	316
4.1.16	DIN-Tastatur-Abfrage	367
4.1.17	Kernal (Fortsetzung)	374
4.2	Verweistabelle (Crossreference)	378

Anhänge

A	**Einführung in den Mikroprozessor 65xx bzw. 8502**	405
A.1	Allgemeines	405
A.2	Architektur und Organisation vom 65xx und 8502	405
A.2.1	PC - Programmzähler (Program Counter)	405
A.2.2	AC - Accumulator	406
A.2.3	XR / YR - Index-Register X und Y	406
A.2.4	SP - Stapel-Zeiger (Stack Pointer)	406
A.2.5	SR - Status-Register	406
A.2.6	ALU - Arithmetic Logic Unit	408
A.2.7	Interrupt-Logik	408
A.2.8	Timer	409

A.2.9 Speicheradressen mit besonderer Bedeutung 409
A.2.10 Besonderheiten des Prozessors 8502 410
A.3 Adressierungsarten 410
A.3.1 Immediate - direkte Adressierung 410
A.3.2 Zero-Page-Adressierung 411
A.3.3 Zero-Page indizierte Adressierung - Indexregister 411
A.3.4 Absolute Adressierung 411
A.3.5 Absolut indizierte Adressierung - Indexregister X und Y 412
A.3.6 Indiziert-indirekte Adressierung - Indexregister X (Zero-Page) 412
A.3.7 Indirekt-indizierte Adressierung - Indexregister Y (Zero-Page) 413
A.3.8 Indirekt absolute Adressierung 413
A.3.9 Relative Adressierung 413
A.3.10 Implizite Adressierung - Akkumulator Adressierung 414

B Prozessor-Befehle 415

C Umrechnungstabelle 441

Stichwortverzeichnis 447

Übersicht weiterer Markt&Technik-Bücher 451

1 Der Mikrocomputer C-128

Der Commodore C-128 ist als Nachfolger des C-64, des verbreitetsten Mikrocomputers überhaupt, zu sehen, und ist als erstes Gerät der Firma Commodore kompatibel zu seinem Vorgänger. Kompatibel heißt in diesem Fall, daß im C-128 eigentlich ein kompletter C-64 eingebaut ist, mit allen seinen Stärken und Schwächen.

1.1 Einführung in die Hardware des C-128

Die 128-er Betriebsart bietet sowohl mehr Speicherplatz als beim C-64, als auch ein erheblich verbessertes BASIC. Hier können wahlweise zwei Bildschirmformate, 40- und 80-Zeichen pro Zeile verwendet werden, wobei beide Bildschirmformate über getrennte Video-Ausgänge herausgeführt werden. Es eröffnet sich hier also ein weites Feld für anspruchsvolle Software, z.B. mit Grafikausgabe auf einem Bildschirm und erklärenden Texten auf dem anderen. Dies setzt natürlich das Vorhandensein zweier Monitore voraus, will man beide Bilder gleichzeitig sehen.

Ein weiterer Pluspunkt des C-128 ist ein eingebauter zweiter Mikroprozessor vom Typ Z80. Hiermit eröffnet sich dem Anwender die weite Welt des CP/M-Betriebssystems, das wohl zu den bekanntesten Systemen überhaupt gehört. Mit Hilfe einer mitgelieferten Systemdiskette wird der C-128 somit zu einem vollwertigen CP/M-Rechner, auf dem eigentlich alle verfügbare CP/M-Software laufen sollte. Wir wollen hier allerdings nicht verschweigen, daß das CP/M auf dem C-128 nicht gerade eines der schnellsten ist, aber der C-128 ist ja als Homecomputer (Hobby-Computer) zu sehen, und Hobbyisten haben doch immer viel Zeit!?

Beginnen wir unseren Spaziergang durch das Innenleben des C-128 mit einem Überblick über die Hardware des Rechners.

1.1.1 Zentral-Prozessor

Der im C-128 verwendete Mikroprozessor ist ein 8502 und ist in die weitverbreitete Familie der 65xx-Prozessoren einzureihen. Er hat den gleichen Befehlssatz, gleiche Adressierungsarten und unterscheidet sich im wesentlichen, z.B. vom 6502 in den CBM-80xx Rechnern, nur durch eine unterschiedliche Anschlußbelegung des Bausteins und den zusätzlichen Ein- bzw. Ausgabe-Port auf den Speicherstellen $00 und $01. Diesen Port besitzt auch

bereits der Prozessor 6510, der im C-64, dem wohl bekanntesten Mikrocomputer überhaupt, verwendet wird.

Der einzige Unterschied zwischen dem 6510 und dem 8502 ist, von der Anschlußbelegung einmal abgesehen, ein zusätzliches Bit im Ein-/Ausgabe-Port, der damit beim 8502 sieben Bits umfaßt.

1.1.1.1 Interne Register der 8502-CPU

Adresse $0000: 8502 Port Datenrichtungs Register

```
Bits        Funktion

x0101111    0 = Input, 1 = Output (s. Daten Reg.)
```

Adresse $0001: 8502 Port Daten Register

```
Bits        Funktion

7.......    unbenutzt

.6......    Status CAPS LOCK/DIN Taste
        0   Taste gedrückt
        1   Taste nicht gedrückt
..5.....    Steuerung Kassettenmotor
        0   Motor ein
        1   Motor aus

...4....    Eingang Taste Kassettenrek.
        0   Taste an Rekorder gedrückt
        1   Taste an Rekorder nicht gedrückt

....3...    Kassette Schreibleitung

        0   serielle Schreibleitung 0
        1   serielle Schreibleitung 1

.....2..    Auswahl I/O / ROM $D000-$DFFF (C-64)

        0   Character Generator ROM ein $D000-$DFFF
        1   I/O-Chips C-64 ein $D000-$DFFF

......1.    Auswahl KERNAL ROM / RAM $E000-$FFFF (C-64)

        0   8 KByte RAM ein $E000-$FFFF
        1   KERNAL ROM C-64 ein $E000-$FFFF

.......0    Auswahl BASIC ROM / RAM $A000-$BFFF (C-64)
            (C-128: Multicolorfarbe II in zweitem Farb-RAM
            für Splitscreens)

        0   8 KByte RAM ein $A000-$BFFF
        1   BASIC ROM C-64 ein $A000-$BFFF
```

1.1.2 Peripherie-Bausteine

Mit einem Mikroprozessor alleine ist ein Computer natürlich noch nicht funktionsfähig; er benötigt noch diverse Hilfsbausteine zur Kommunikation mit der Außenwelt und zum Speichern von Programmen und Daten, und - natürlich - ein Betriebsystem, das für ein sinnvolles Zusammenspiel aller dieser Komponenten sorgt.

An Peripherie-Bausteinen finden wir im C-128 zunächst einmal alles das wieder, was auch schon im C-64 vorhanden ist, sogar die Speicheradressen der Bausteine sind die gleichen geblieben; sonst wäre die 64-Kompatibilität auch nicht zu erreichen gewesen.

Im einzelnen sind folgende Peripherie-Bausteine vorhanden:

Baustein	Basic-Adresse	Funktion
VIC	$D000	Video-Kontroller für 40-Zeichen-Bildschirm und C-64-Modus
SID	$D400	Klangerzeugung
MMU	$D500	Verwaltung der RAM-Banks und allgemeine Speicherverwaltung
VDC	$D600	Video-Kontroller für 80-Zeichen-Bildschirm
CIA #1	$DC00	Bedienung von Tastatur, Joystick und Kassettenport
CIA #2	$DD00	Userport, RS-232 Schnittstelle

1.1.3 VIC - Videocontroller für den 40-Zeichen-Bildschirm

Im 40-Zeichenmodus bzw. im C-64-Modus wird die gesamte Bildschirmsteuerung durch den VIC übernommen. Der VIC hat zunächst die Aufgabe, Zeichen auf dem Bildschirm darzustellen und zwar in 16 verschiedenen Farben. Hierzu dienen zwei Speicherbereiche, die jeweils ein Abbild des Schirminhaltes darstellen. Der eine Bereich von $0400 bis 07E7 enthält die darzustellenden Zeichen, und der zweite Bereich von $D800 bis $DBE7 die jeweils notwendige Farbinformation.

Die beiden Speicherbereiche können jeweils mit einem POKE-Befehl vom BASIC her beschrieben werden. Von einem Assemblerprogramm aus ist der Bildschirm wie ein normaler Speicherbereich zu adressieren, d.h. ein STA-Befehl in der Adresse $0400 schreibt das Zeichen aus dem Akkumulator in die linke obere Ecke des Schirms. Gleiches gilt auch für den Farbspeicher.

Ein Farbwechsel eines Zeichens ist durch einfaches Ändern der zugehörigen Farbadresse jederzeit möglich.

Zusätzlich zur normalen Zeichendarstellung ist der VIC in der Lage, Grafik zu erzeugen und zwar in Form einer sogenannten Bitmap-Grafik. Ein bestimmter Speicherbereich wird hierbei als Grafikbereich angenommen und für jedes dort gesetzte Bit erscheint ein Punkt auf dem Bildschirm; jedes nicht gesetzte Bit wird ignoriert. Es kann hier mit einer Auflösung von 320 * 200 Punkten gearbeitet werden, was einem Speicherbedarf von 8 KByte entspricht. Alternativ hierzu kann eine verminderte Auflösung gewählt werden (160 * 200) Punkte, wobei dann jeweils 2 Bit im Speicher einem Punkt auf dem Bildschirm entsprechen. Hierdurch sind insgesamt vier verschiedene Farben pro Bildschirmpunkt darstellbar.

Alle diese Funktionen werden beim C-128 durch das exzellente BASIC unterstützt; der Anwender ist also nicht mehr gezwungen, mittels umständlicher POKE-Befehle Grafik oder auch Sprites zu erzeugen. Die einzelnen Befehle hierzu finden Sie im Handbuch des C-128.

Zwei Adressen, die bei der C-64-Version des VIC nicht vorhanden und auch im Handbuch nicht beschrieben sind, bedürfen einer näheren Erklärung. Es handelt sich hier um die Register 47 ($2F) und 48 ($30). Das Register 47, im Assemblerlisting mit "vicpen" bezeichnet, ist ein zusätzlicher Port, der im 128-er Modus die zusätzlichen Tasten wie Zehnertastatur, Help-Taste usw. freigibt. Im C-64-Modus sind diese Tasten nicht ansprechbar, da das VIC-Register nicht angesprochen wird.

Im Register 47 befinden sich die Tastaturmatrix-Leitungen der zusätzlichen Tasten, die durch die entsprechende Software auch im C-64-Modus abgefragt werden können.

Das Register 48, im Assemblerlisting als "vicspd" bezeichnet, dient zur Umschaltung der Prozessorgeschwindigkeit. Eine Null im Bit 0 stellt den Prozessor auf 1-MHz-Betrieb, eine Eins auf 2-MHz-Betrieb. Zusätzlich sollte allerdings die Bildschirmausgabe des VIC abgeschaltet werden, da ansonsten ein recht wirres Bildschirmmuster dargestellt wird.

1.1.3.1 Interne Register des Video Controllers VIC 8564

Unbenutzte Bits erscheinen beim Lesen immer als "1".

Adresse	Bits	Funktion
$D000	76543210	Sprite0, lo-Byte x-Wert (Abszisse)
$D001	76543210	Sprite0, lo-Byte y-Wert (Ordinate)

```
$D002       76543210  Sprite1, lo-Byte x-Wert (Abszisse)
$D003       76543210  Sprite1, lo-Byte y-Wert (Ordinate)

$D004       76543210  Sprite2, lo-Byte x-Wert (Abszisse)
$D005       76543210  Sprite2, lo-Byte y-Wert (Ordinate)

$D006       76543210  Sprite3, lo-Byte x-Wert (Abszisse)
$D007       76543210  Sprite3, lo-Byte y-Wert (Ordinate)

$D008       76543210  Sprite4, lo-Byte x-Wert (Abszisse)
$D009       76543210  Sprite4, lo-Byte y-Wert (Ordinate)

$D00A       76543210  Sprite5, lo-Byte x-Wert (Abszisse)
$D00B       76543210  Sprite5, lo-Byte y-Wert (Ordinate)

$D00C       76543210  Sprite6, lo-Byte x-Wert (Abszisse)
$D00D       76543210  Sprite6, lo-Byte y-Wert (Ordinate)

$D00E       76543210  Sprite7, lo-Byte x-Wert (Abszisse)
$D00F       76543210  Sprite7, lo-Byte y-Wert (Ordinate)

$D010       76543210  MSB (=Bit8) Abszisse Sprite7-0

$D011       7.......  MSB (=Bit8) Raster-Wert Reg. 18
            .6......  Extended Color Text (0 = aus, 1 = ein)
            ..5.....  Bit Map Mode (0 = aus, 1 = ein)
            ...4....  Blank Screen to Color Border (0 = Blank)
            ....3...  24/25 Zeilen BS (0 = 24, 1 = 25)
            .....210  Soft-Scroll vertikal, Dot-Bits 0-7

$D012       76543210  lo-Byte Vorgabe für Raster IRQ (WRITE)
                      lo-Byte akt. Raster-Wert (READ)
                      MSB (=Bit8) in Reg. 17, Bit7

$D013       76543210  Light-Pen, x-Wert (Abszisse)
$D014       76543210  Light-Pen, y-Wert (Ordinate)

$D015       76543210  Sprite7-0 (1 = ein)

$D016       76......  unbenutzt
            ..5.....  immer 0 setzen (reserviert)
            ...4....  Priorität (0 = aus, 1 = ein)
            ....3...  38/40 Spalten/Zeile (0 = 38, 1 = 40)
            .....210  Soft-Scroll horizontal, Dot-Bits 0-7

$D017       76543210  Sprite7-0 vert. verdoppeln (1 = ein)

$D018       7654....  Adreßbits 13-10 Basisadresse BS RAM
            ....321.  Adreßbits 13-11 Basisadresse Chargen. ROM
            .......0  unbenutzt

$D019       7.......  Interrupt-Flag (1 = VIC IRQ)
            .654....  unbenutzt
            ....3...  Light-Pen IRQ (1 = IRQ)
            .....2..  Sprite/Sprite Kollision (1 = IRQ)
            ......1.  Sprite/Hintergrund Kollision (1 = IRQ)
            .......0  Raster-Vergleich IRQ (1 = IRQ)
```

```
$D01A       7654....  unbenutzt
            ....3...  Light-Pen IRQ (1 = IRQ enabled)
            .....2..  Sprite/Sprite Kollision (1 = IRQ enabled)
            ......1.  Sprite/Hintergrund Koll.(1 = IRQ enabled)
            .......0  Raster-Vergleich IRQ (1 = IRQ enabled)

$D01B       76543210  Priorität für Sprite7-0 (1 = ein)

$D01C       76543210  Multi-Color-Modus für Sprite7-0 (1 = ein)

$D01D       76543210  Sprite7-0 hor. verdoppeln (1 = ein)

$D01E       76543210  Sprite-Kollision mit Sprite7-0 (1 = ein)

$D01F       76543210  Hintergrund-Koll.mit Sprite7-0 (1 = ein)

$D020       7654....  unbenutzt
            ....3210  Rahmenfarbe (0-15)

$D021       7654....  unbenutzt
            ....3210  Hintergrundfarbe #0 (0-15)

$D022       7654....  unbenutzt
            ....3210  Hintergrundfarbe #1 (0-15)

$D023       7654....  unbenutzt
            ....3210  Hintergrundfarbe #2 (0-15)

$D024       7654....  unbenutzt
            ....3210  Hintergrundfarbe #3 (0-15)
$D025       7654....  unbenutzt
            ....3210  Sprite Multicolor #0 (0-15)

$D026       7654....  unbenutzt
            ....3210  Sprite Multicolor #1 (0-15)

$D027       7654....  unbenutzt
            ....3210  Farbe Sprite0 (0-15)

$D028       7654....  unbenutzt
            ....3210  Farbe Sprite1 (0-15)

$D029       7654....  unbenutzt
            ....3210  Farbe Sprite2 (0-15)

$D02A       7654....  unbenutzt
            ....3210  Farbe Sprite3 (0-15)

$D02B       7654....  unbenutzt
            ....3210  Farbe Sprite4 (0-15)

$D02C       7654....  unbenutzt
            ....3210  Farbe Sprite5 (0-15)

$D02D       7654....  unbenutzt
            ....3210  Farbe Sprite6 (0-15)
```

```
$D02E       7654....  unbenutzt
            ....3210  Farbe Sprite7 (0-15)

$D02F       7654....  unbenutzt
            ....3210  VIC Tastatur-Port (K3-K0)

$D030       7654321.  unbenutzt
            .......0  Flag CPU-Takt (0 = 1 MHz, 1 = 2 MHz)
```

1.1.4 SID - Klangerzeugungs-Baustein

Der SID ist völlig identisch mit dem des C-64. Er bietet drei Stimmen, die mit unterschiedlichen Schwingungsformen und Hüllkurven (Lautstärke-Verlauf) versehen werden können, und ein programmierbares Filter. Alle wichtigen Parameter zur Klangerzeugung lassen sich im übrigen auch mit BASIC-Befehlen setzen.

1.1.4.1 Interne Register des Sound Interface Device SID 6581

```
Adresse     Bits      Funktion

$D400       76543210  lo-Byte Frequenz Stimme 1

$D401       76543210  hi-Byte Frequenz Stimme 1

$D402       76543210  lo-Byte Pulsbreite Stimme 1

$D403       7654....  unbenutzt
            ....3210  hi-Byte Nibble Pulsbreite Stimme 1

$D404                 Kontrollregister Stimme 1

            7.......  1 = Zufallsgenerator ein
            .6......  1 = Rechteckschwingung ein
            ..5.....  1 = Sägezahnschwingung ein
            ...4....  1 = Dreieckschwingung ein
            ....3...  1 = Oszillator 1 aus
            .....2..  1 = Stimme 1 mit 3 ring-modulieren
            ......1.  1 = Stimme 1 mit 3 synchronisieren
            .......0  1 = Start Att/Dec/Sus, 0 = Start Rel.

$D405       7654....  Attack - Zeit (0-15) Stimme 1
            ....3210  Decay  - Zeit (0-15) Stimme 1

$D406       7654....  Sustain - Zeit (0-15) Stimme 1
            ....3210  Release - Zeit (0-15) Stimme 1

$D407       76543210  lo-Byte Frequenz Stimme 2

$D408       76543210  hi-Byte Frequenz Stimme 2

$D409       76543210  lo-Byte Pulsbreite Stimme 2
```

$D40A	7654....	unbenutzt
	3210	hi-Byte Nibble Pulsbreite Stimme 2
$D40B		Kontrollregister Stimme 2
	7.......	1 = Zufallsgenerator ein
	.6......	1 = Rechteckschwingung ein
	..5.....	1 = Sägezahnschwingung ein
	...4....	1 = Dreieckschwingung ein
	....3...	1 = Oszillator 1 aus
	.....2..	1 = Stimme 2 mit 1 ring-modulieren
	......1.	1 = Stimme 2 mit 1 synchronisieren
	0	1 = Start Att/Dec/Sus, 0 = Start Rel.
$D40C	7654....	Attack - Zeit (0-15) Stimme 2
	3210	Decay - Zeit (0-15) Stimme 2
$D40D	7654....	Sustain - Zeit (0-15) Stimme 2
	3210	Release - Zeit (0-15) Stimme 2
$D40E	76543210	lo-Byte Frequenz Stimme 3
$D40F	76543210	hi-Byte Frequenz Stimme 3
$D410	76543210	lo-Byte Pulsbreite Stimme 3
$D411	7654....	unbenutzt
	3210	hi-Byte Nybble Pulsbreite Stimme 3
$D412		Kontrollregister Stimme 3
	7.......	1 = Zufallsgenerator ein
	.6......	1 = Rechteckschwingung ein
	..5.....	1 = Sägezahnschwingung ein
	...4....	1 = Dreieckschwingung ein
	....3...	1 = Oszillator 1 aus
	.....2..	1 = Stimme 3 mit 2 ring-modulieren
	......1.	1 = Stimme 3 mit 2 synchronisieren
	0	1 = Start Att/Dec/Sus, 0 = Start Rel.
$D413	7654....	Attack - Zeit (0-15) Stimme 3
	3210	Decay - Zeit (0-15) Stimme 3
$D414	7654....	Sustain - Zeit (0-15) Stimme 3
	3210	Release - Zeit (0-15) Stimme 3
$D415	3210	Filter-Frequenz Low-Byte
$D416	76543210	Filter-Frequenz High-Byte
$D417	7654....	Filter-Resonanz
	....3...	Filter Externer Eingang, 1 = Ein
	.....2..	Filter Stimme 3, 1 = Ein
	......1.	Filter Stimme 2, 1 = Ein
	0	Filter Stimme 1, 1 = Ein

```
$D018       7.......  1 = Stimme 3 abgeschaltet
            .6......  1 = Filter Hochpaß ein
            ..5.....  1 = Filter Bandpaß ein
            ...4....  1 = Filter Tiefpaß ein
            ....3210  Lautstärke (0-15)
$D019       76543210  AD-Wandler 1

$D01A       76543210  AD-Wandler 2

$D01B       76543210  Rauschgenerator Stimme 3, aktueller Wert

$D01C       76543210  Lautstärke Stimme 3, aktueller Wert
```

1.1.4.2 Funktionen der SID-Register

$D400 / $D401 Frequenz Stimme 1

Die gewünschte Frequenz (Tonhöhe) des Tongenerators 1 wird hier als 16-Bit-Wert angegeben, wobei in $D400 das Low-Byte und in $D401 das High-Byte angegeben wird. Die Frequenz kann mit Hilfe einer Formel errechnet werden. Sie wird zum einen vom genannten 16-Bit-Wert, zum anderen vom Systemtakt bestimmt . Die Formel lautet:

$$\text{Frequenz} = \frac{\text{Wert} * \text{Taktfrequenz}}{16777216} \text{ [Hz]} \qquad (16777216 = 2^{24})$$

Der Systemtakt für den SID, wie auch für die anderen Peripherie-Bausteine, beträgt immer 1 MHz, unabhängig vom Fast- oder Slow-Modus. Somit kann die Formel umgeformt werden zu:

```
Frequenz = Wert * 0,0587221 [Hz]
```

Wer jetzt diese Formel einmal nachrechnet wird feststellen, daß die Taktfrequenz weniger als 1 MHz beträgt. Genau beträgt sie 985193 Hz, bedingt durch unsere PAL-Fernsehnorm, die eine andere Farbfrequenz benötigt als die amerikanische Norm NTSC *(Never Try Same Color, oder so ähnlich).*

Der Kammerton A (A-4) hat eine Frequenz von 440 Hz. Die Erhöhung des Tons um eine Oktave entspricht der Verdopplung der Frequenz (A-5 = 880 Hz) und die Verminderung des Tons um eine Oktave halbiert die Frequenz (A-3 = 220 Hz). Die einzelnen Töne einer Oktave stehen im Verhältnis "1 : (12te Wurzel aus 2)" zueinander. Das "Ais", manchmal auch "B" genannt, ist also in der Frequenz um diesen Faktor (1,059463094) höher, bzw. das Gis um diesen Faktor niedriger. Ausgehend von diesen Werten kann man jede gewünschte Frequenz errechnen. Es folgt eine Tabelle für die Noten der sogenannten wohltemperierten Stimmung mit den notwendigen Werten für die SID-Register, wahlweise in Dezimal- oder Hexadezimal-Darstellung.

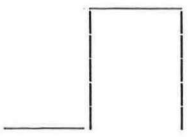
Musiknoten-Tabelle

Note	Frequenz	Wert	Low	High	Low	High
			Dezimal		Hexadezimal	
C -0	16	278	22	1	16	01
C#-0	17	295	39	1	27	01
D -0	18	312	56	1	38	01
D#-0	19	331	75	1	4B	01
E -0	21	350	94	1	5E	01
F -0	22	371	115	1	73	01
F#-0	23	393	137	1	89	01
G -0	24	417	161	1	A1	01
G#-0	26	442	186	1	BA	01
A -0	27	468	212	1	D4	01
A#-0	29	496	240	1	F0	01
B -0	31	525	13	2	0D	02
C -1	33	556	44	2	2C	02
C#-1	35	590	78	2	4E	02
D -1	37	625	113	2	71	02
D#-1	39	662	150	2	96	02
E -1	41	701	189	2	BD	02
F -1	44	743	231	2	E7	02
F#-1	46	787	19	3	13	03
G -1	49	834	66	3	42	03
G#-1	52	884	116	3	74	03
A -1	55	936	168	3	A8	03
A#-1	58	992	224	3	E0	03
B -1	62	1051	27	4	1B	04
C -2	65	1113	89	4	59	04
C#-2	69	1180	156	4	9C	04
D -2	73	1250	226	4	E2	04
D#-2	78	1324	44	5	2C	05
E -2	82	1403	123	5	7B	05
F -2	87	1486	206	5	CE	05
F#-2	92	1575	39	6	27	06
G -2	98	1668	132	6	84	06
G#-2	104	1768	232	6	E8	06
A -2	110	1873	81	7	51	07
A#-2	117	1984	192	7	C0	07
B -2	123	2102	54	8	36	08
C -3	131	2227	179	8	B3	08
C#-3	139	2360	56	9	38	09
D -3	147	2500	196	9	C4	09
D#-3	156	2649	89	10	59	0A
E -3	165	2806	246	10	F6	0A
F -3	175	2973	157	11	9D	0B
F#-3	185	3150	78	12	4E	0C
G -3	196	3337	9	13	09	0D
G#-3	208	3536	208	13	D0	0D
A -3	220	3746	162	14	A2	0E
A#-3	233	3969	129	15	81	0F
B -3	247	4205	109	16	6D	10
C -4	262	4455	103	17	67	11
C#-4	277	4720	112	18	70	12

Note	Frequenz	Wert	Low	High	Low	High
			Dezimal		Hexadezimal	
D -4	294	5001	137	19	89	13
D#-4	311	5298	178	20	B2	14
E -4	330	5613	237	21	ED	15
F -4	349	5947	59	23	3B	17
F#-4	370	6300	156	24	9C	18
G -4	392	6675	19	26	13	1A
G#-4	415	7072	160	27	A0	1B
A -4	440	7493	69	29	45	1D
A#-4	466	7938	2	31	02	1F
B -4	494	8410	218	32	DA	20
C -5	523	8910	206	34	CE	22
C#-5	554	9440	224	36	E0	24
D -5	587	10002	18	39	12	27
D#-5	622	10596	100	41	64	29
E -5	659	11226	218	43	DA	2B
F -5	698	11894	118	46	76	2E
F#-5	740	12601	57	49	39	31
G -5	784	13351	39	52	27	34
G#-5	831	14145	65	55	41	37
A -5	880	14986	138	58	8A	3A
A#-5	932	15877	5	62	05	3E
B -5	988	16821	181	65	B5	41
C -6	1046	17821	157	69	9D	45
C#-6	1109	18881	193	73	C1	49
D -6	1175	20004	36	78	24	4E
D#-6	1244	21193	201	82	C9	52
E -6	1318	22453	181	87	B5	57
F -6	1397	23789	237	92	ED	5C
F#-6	1480	25203	115	98	73	62
G -6	1568	26702	78	104	4E	68
G#-6	1661	28290	130	110	82	6E
A -6	1760	29972	20	117	14	75
A#-6	1865	31754	10	124	0A	7C
B -6	1976	33642	106	131	6A	83
C -7	2093	35643	59	139	3B	8B
C#-7	2217	37762	130	147	82	93
D -7	2349	40008	72	156	48	9C
D#-7	2489	42387	147	165	93	A5
E -7	2637	44907	107	175	6B	AF
F -7	2794	47578	218	185	DA	B9
F#-7	2960	50407	231	196	E7	C4
G -7	3136	53404	156	208	9C	D0
G#-7	3322	56580	4	221	04	Dd
A -7	3520	59944	40	234	28	EA
A#-7	3729	63508	20	248	14	F8

$D402 / $D403 Pulsbreite Stimme 1

Eine Rechteckspannung bzw. -Kurve besteht aus zwei Teilen: einem Teil des Zyklus, in dem die Spannung 0 V beträgt und einem Teil, in dem die Spannung auf dem Maximalwert liegt. Ein kleines Beispiel soll dies verdeutlichen:



Teil1 Teil2

Die Pulsbreite ist die Breite des Teils 2 im Verhältnis zum Teil 1 und wird in Prozenten angegeben. Eine Pulsbreite von 50 % bedeutet also, daß beide Teile des Zyklus gleich breit, bzw. zeitlich gleich lang sind.

Eine Rechteckschwingung setzt sich aus vielen Sinusschwingungen zusammen: zum einen dem Grundton, der die eigentliche Frequenz bestimmt und zum anderen aus vielen Obertönen, also Sinusschwingungen mit höherer Frequenz. Durch Verändern der Pulsbreite einer Rechteckwelle verändert sich auch die Zusammensetzung dieser Sinusschwingungen, womit akustisch eine Klangänderung wahrgenommen wird.

Die Pulsbreite kann beim SID in den oben genannten Registern angegeben werden, wobei sich die Pulsbreite nach folgender Formel errechnet:

$$PW = \frac{Wert}{40{,}95} \quad (\%)$$

Angegeben werden können hier Werte zwischen 0 und 4095, wobei der Wert 2048 eine Pulsbreite von etwa 50 % ergibt.

In den Registern wird in $D402 das Low-Byte des Wertes und in $D403 das High-Byte des Wertes angegeben, wobei das High-Byte nur Werte zwischen 0 und 15 annehmen kann.

Die Pulsbreite hat nur Einfluß auf die Rechteckschwingung, die anderen Formen der SID-Generatoren werden hierdurch nicht verändert.

$D404 Kontroll-Register Stimme 1

Das Kontrollregister steuert die verschiedenen Funktionen des ersten Tongenerators im SID. Die einzelnen Bits haben folgende Bedeutung:

Bit 0 Gate-Bit

Wenn dieses Bit auf "1" gesetzt wird, beginnt der Hüllkurvenzyklus. Die in den entsprechenden Registern angegebene Hüllkurve wird von ATTACK über DECAY durchlaufen und schließlich der durch SUSTAIN angegebene Lautstärkepegel erreicht.

Wird dieses Bit auf "0" gesetzt, wird der RELEASE-Zyklus durchgeführt. Die Lautstärke des Tons geht also in der definierten Zeit auf Null zurück.

Bit 1 SYNC

Wenn dieses Bit auf "1" gesetzt wird, werden die Generatoren 1 und 3 miteinander synchronisiert. Der Generator 1 läuft zunächst mit der definierten Frequenz weiter, ebenfalls der Generator 3. Wann immer nun die vom Generator 3 erzeugte Schwingung einen Nulldurchgang hat, wird Generator 1 gezwungen, seine gerade erzeugte Schwingung ebenfalls neu zu starten. Generator 1 läuft also in Abhängigkeit von Generator 3. Klanglich entstehen hierdurch Töne mit teilweise sehr unharmonischen, metallischen Klängen.

Bit 2 Ringmodulator

Auch wenn dieses Bit auf "1" gesetzt wird, werden die beiden Generatoren 1 und 3 miteinander verknüpft. Das Ergebnis ist hier eine Ring-Modulation - auch Kreuzmodulation genannt - der beiden durch die Generatoren erzeugten Frequenzen. Klanglich entstehen hierbei Töne mit nicht harmonischen Obertönen, wie sie z.B. zur Simulation von Glocken, Gongs und anderem verwendet werden können.

Bit 3 Testbit

Das Setzen dieses Bits auf "1" hält den Generator 1 an und setzt die Generatoren zurück.

Bit 4 Dreieckschwingung

Wird dieses Bit auf "1" gesetzt, erzeugt der Generator eine Dreieckschwingung. Klanglich entspricht sie in etwa einer Sinusschwingung, also einem reinen Ton mit wenig Obertönen.

Bit 5 Sägezahnschwingung

Dieses Bit selektiert die Sägezahnschwingung. Sie enthält wesentlich mehr Obertöne als die Dreickschwingung und kann als Grundklang für viele Instrumente dienen, wenn sie durch Filterung entsprechend bearbeitet wird.

Bit 6 Rechteckschwingung

Dieses Bit selektiert eine Rechteckschwingung mit der bereits oben beschriebenen Pulsbreite. Eine reine Rechteckschwingung ähnelt klanglich etwa dem Ton einer Oboe und kann durch Filterung vielseitig verändert werden.

Bit 7 Rauschen

Das letzte Bit dieses Registers selektiert den Rauschgenerator, der ein sogenanntes "Weißes Rauschen" erzeugt. Es handelt sich hierbei um ein Gemisch unterschiedlichster Frequenzen ohne definierte Tonhöhe.

Ein Hinweis noch: Die einzelnen Kurvenformen können nicht gleichzeitig erzeugt werden, d.h. es kann nur jeweils eine der genannten Schwingungen erzeugt werden. Werden mehrere der Bits auf "1" gesetzt, ist das Ergebnis eine UND-Verknüpfung der jeweiligen erzeugten Schwingungen. Hierdurch können natürlich durchaus interessante Schwingungsformen und damit Klänge entstehen, es kann allerdings vorkommen, daß der Rauschgenerator in eine Endlosschleife kommt. In diesem Fall müssen Sie mit dem Testbit die Generatoren zurücksetzen.

$D405 / $D406 Attack/Decay, Sustain/Release

Die Register Attack/Decay bzw. Sustain Release bestimmen die Hüllkurve eines Klanges. Wer sich schon mit elektronischer Klangerzeugung beschäftigt hat, kennt dieses auch von diversen Musik-Synthesizern, wo entsprechende ADSR-Generatoren für den zeitlichen Ablauf eines Klangereignisses sorgen.

Beim Setzen des Gate-Bits im jeweiligen Kontrollregister wird der entsprechende Generator eingeschaltet. ATTACK bestimmt hier die Zeit, in der die Lautstärke des Tons auf den Maximalwert ansteigt. Anschließend folgt mit dem DECAY wiederum ein Zeitwert der angibt, in welcher Zeit die Lautstärke abnehmen soll und auf den durch SUSTAIN bestimmten Lautstärkewert sinkt. Nach Abschalten des Gate-Bits bestimmt RELEASE, in welcher Zeit die Lautstärke des Tons auf Null sinken soll.

ATTACK, DECAY und RELEASE sind also Zeitwerte. SUSTAIN ist ein Lautstärkewert zwischen 0 und 15. Die jeweiligen Zeit-Werte können Sie den folgenden Tabellen entnehmen.

ATTACK		DECAY / RELEASE	
Wert	Zeit	Wert	Zeit
0	2 ms	0	6 ms
1	8 ms	1	24 ms
2	16 ms	2	48 ms
3	24 ms	3	72 ms
4	38 ms	4	114 ms
5	56 ms	5	168 ms
6	68 ms	6	204 ms
7	80 ms	7	240 ms
8	100 ms	8	300 ms
9	250 ms	9	750 ms
10	500 ms	10	1,5 s
11	800 ms	11	2,4 s
12	1 s	12	3 s
13	3 s	13	9 s
14	5 s	14	15 s
15	8 s	15	24 s

$D407 ... $D40D Generator 2

Die einzelnen Register arbeiten wie bei Stimme 1, mit zwei Ausnahmen im Kontrollregister:

SYNC: Synchronisiert Stimme 2 mit Stimme 1

RING: Ringmoduliert wird hier Stimme 2 mit Stimme 1

$D40E ... $D414 Generator 3

Die einzelnen Register arbeiten wie bei Stimme 1 mit zwei Ausnahmen im Kontrollregister:

SYNC: Synchronisiert Stimme 3 mit Stimme 2

RING: Ringmoduliert wird hier Stimme 3 mit Stimme 2

$D415 / $D416 Filter-Freuquenz

Der SID enthält ein programmierbares Filter, dessen Mittenfrequenz in diesen beiden Registern angegeben wird. Der Wert kann zwischen 0 und 2047 liegen, was einer Mittenfrequenz zwischen 30 Hz und 12 KHz entspricht.

Die Frequenz kann nach folgender Formel in etwa errechnet werden:

$$\text{Frequenz} = \frac{12000 * \text{Wert}}{2048} + 30 \quad [\text{Hz}]$$

oder umgeformt:

$$\text{Frequenz} = \text{Wert} * 5{,}86 + 30 \quad [\text{Hz}]$$

$D417 Filter-Kontrollregister 1

Die oberen 4 Bits (4-7) bestimmen die Resonanz des Filters. Unter Resonanz versteht man ein gewisses Mitschwingen des Filters mit der aktuellen Mittenfrequenz. Ein bekannter Effekt der dieses verdeutlicht ist der sogenannte Wah-Wah-Effekt, bei dem das Filter teilweise mitschwingt und die Mittenfrequenz zyklisch steigt und sinkt.

Angegeben werden kann hier ein Wert zwischen 0 und 15, wobei 0 keine und 15 maximale Resonanz darstellt.

Die restlichen Bits schalten die drei Stimmen des SID und die externe Klangquelle wahlweise auf das Filter (Bit ist "1") oder direkt auf den Ausgang (Bit ist "0"). Die einzelnen Bits selektieren folgende Stimmen:

Bit 0 Stimme 1
Bit 1 Stimme 2
Bit 2 Stimme 3
Bit 3 Externe Klangquelle

$D418 Filter Kontrollregister 2

Bit 0-3 Lautstärke

Hier wird die Gesamtlautstärke des SID angegeben, wobei der Wert 0 keinen Ton produziert und bei einem Wert von 15 maximale Lautstärke erreicht wird. Die anderen Werte selektieren Lautstärkestufen dazwischen und können z.B. für Tremolo-Effekte verwendet werden.

Bit 4-6 Filterart

Hier können die einzelnen Filterarten definiert werden:

Bit 4 Tiefpaß-Filter.
Alle Frequenzen oberhalb der Mittenfrequenz werden mit 12 db/Okt. beschnitten.

Bit 5 Bandpaß-Filter.
Alle Frequenzen oberhalb und unterhalb der Mittenfrequenz werden mit 6 db/Okt. beschnitten.

Bit 6 Hochpaß-Filter.
Alle Frequenzen unterhalb der Mittenfrequenz werden mit 12 db/Okt. beschnitten. Die einzelnen Filtertypen können kombiniert werden. So ergibt die Kombination aus Hoch- und Tiefpaß-Filter z.B. ein Notch-Filter, d.h. die Mittenfrequenz wird gesperrt.

Bit 7 Stummschalten Stimme 3

Stimme 3 wird stumm geschaltet, falls diese zur Erzeugung von Modulationen verwendet wird.

$D419 / $D41A AD-Wandler 1 und 2

Hier werden die an den C-128 anschließbaren Paddles ausgelesen, die im wesentlichen nur aus einem Potentiomenter, d.h. einem veränderlichen Widerstand, aufgebaut sind. Entsprechend dem momentanen Widerstandswert wird ein Bitwert zwischen 0 und 255 errechnet und kann aus diesen Registern ausgelesen werden.

Durch Verwendung geeigneter Schaltungen (z.B. mit einem FET-Transistor) könnten hier auch Spannungen gemessen und somit z.B. auch Sprache digitalisiert und im Rechner verarbeitet werden. Die AD-Wandler im SID brauchen für einen Wandlungszyklus allerdings 512 Mikrosekunden, womit nicht mehr als ca. 1950 Wandlungen pro Sekunde möglich sind. Damit wäre der sinnvoll wandelbare Frequenzbereich auf maximal 975 Hz beschränkt, was selbst für Sprache ein bißchen zu wenig ist.

$D41B Random-Register Stimme 3

Dieses Register kann jederzeit ausgelesen werden und enthält jeweils eine Zahl zwischen 0 und 255, die der Lautstärke der gerade produzierten Schwingung entspricht. Eine Sägezahnschwingung würde, abhängig von der Frequenz, eine mehr oder weniger schnelle Folge von Zahlen zwischen 0 und 255 liefern, die sich ständig wiederholt. Hiermit können nun andere Stimmen moduliert werden, wobei die gelesenen Werte dort z.B. die Frequenz, die Pulsbreite oder ähnliches verändern.

$D41C **Lautstärke-Register Stimme 3**

Hier erscheint der gerade aktuelle Wert des Hüllkurven-Generators von Stimme 3. Diese Werte können ebenfals zur Modulation der anderen Stimmen verwendet werden.

1.1.5 MMU - Speicherverwaltungs-Baustein

Die MMU ist der Baustein, dem im C-128, neben dem Prozessor, die Hauptarbeit zukommt. Die MMU sorgt für die jeweils richtige Adressierung der Speicher- und Peripherie-Bausteine in den verschiedenen Betriebsarten.

Hierzu enthält die MMU zahlreiche Register, in denen die gewünschten Parameter eingestellt werden können. Hier zunächst eine Erläuterung der internen Register:

1.1.5.1 Interne Register der Memory Management Unit MMU 8722

```
Adresse   Bits          Funktion

$D500/$FF00             Configuration Register

          76......      Auswahl der 64 K RAM-Bank

                  00    RAM Bank 0
                  01    RAM Bank 1
                  10    RAM Bank 2 (wenn nicht vorhanden: Bank 0)
                  11    RAM Bank 3 (wenn nicht vorhanden: Bank 1)

          ..54....      Auswahl RAM/ROM Adreßbereich $C000 - $FFFF

                  00    Kernal ROM
                  01    Internal Function ROM
                  10    External Function ROM
                  11    RAM Bank, gewählt mit Bit7/6

          ....32..      Auswahl RAM/ROM Adreßbereich $8000 - $BFFF

                  00    BASIC ROM High
                  01    Internal Function ROM
                  10    External Function ROM
                  11    RAM Bank, gewählt mit Bit7/6

          ......1.      Auswahl RAM/ROM Adreßbereich $4000 - $7FFF

                   0    BASIC ROM  Low
                   1    RAM Bank, gewählt mit Bit7/6

          .......0      Auswahl RAM/ROM-I/O Adreßbereich $D000 - $DFFF
```

```
                0  I/O-Chips
                1  RAM/ROM/Char.gen., gewählt mit Bit 5/4

$D501                Preconfiguration Register A
$FF01                Load Configuration Register A

$D502                Preconfiguration Register B
$FF02                Load Configuration Register B

$D503                Preconfiguration Register C
$FF03                Load Configuration Register C

$D504                Preconfiguration Register D
$FF04                Load Configuration Register D
```

In die Prekonfigurations-Register A bis D wird ein Bitmuster geschrieben, das durch Schreiben eines beliebigen Wertes in das entsprechende Load Configuration Register ins Konfigurations-Register übernommen wird

```
$D505                Mode Configuration Register

        7.......     Flag für 40/80-Zeichen-BS - Taste:

                0    80-Zeichen BS - Taste nicht gedrückt
                1    40-Zeichen BS - Taste gedrückt

        .6......     Flag für C 128 / C-64 - Modus:

                0    C 128 - Modus
                1    C-64 - Modus

        ..5.....     Flag für Steckmodul 'GAME' (C-64):

                0    'GAME' belegt
                1    'GAME' nicht belegt

        ...4....     Flag für Steckmodul'EXROMIN' (C-64):

                0    'EXROMIN' belegt
                1    'EXROMIN' nicht belegt

        ....3...     FSDIR-Control Bit (FAST SERIAL DEVICE)
                0    FSD Input
                1    FSD Output

        .....21.     noch nicht verwendet

        .......0     Auswahl des aktiven Prozessors:

                0    Z 80
                1    8502
```

```
$D506                   RAM Configuration Register

          76......      Auswahl RAM-Bank für VIC-Chip
                  00    RAM Bank 0
                  01    RAM Bank 1
                  10    RAM Bank 2 (wenn nicht vorhanden: Bank 0)
                  11    RAM Bank 3 (wenn nicht vorhanden: Bank 1)

          ..54....      Auswahl 256K-Block (spätere 1MB-Version):

                  00    256K-Block 0
                  01    256K-Block 1
                  10    256K-Block 2
                  11    256K-Block 3

          ....32..      Definition COMMON RAM AREA:

                  00    keine  COMMON AREA, unabhängig von Bits1/0
                  01    untere COMMON AREA
                  10    obere  COMMON AREA
                  11    untere und obere COMMON AREA

          ......10      Definition Größe der COMMON RAM AREA:

                  00     1 K
                  01     4 K
                  10     8 K
                  11    16 K


$D507                   Zero-Page Offset  Low
$D508                   Zero-Page Offset High

$D509                   Stapel-Page Offset Low
$D50A                   Stapel-Page Offset High

$D50B                   Versions-Register

          7654....      Bank-Version

                0010 2 64K Banks (kommentierter Rechner)
                bis
                1111 15 64K Banks (zukünftige Versionen)

          ....3210      MMU - Version

                0000  der kommentierte Rechner besitzt MMU V 0
```

Hier ein kleiner Überblick über die Speicher-Aufteilung im Normalzustand, der für den BASIC-Betrieb und natürlich auch beim Aufruf eines Assembler-Programms mit SYS gilt, wobei unwichtige Speicherbereiche weggelassen wurden:

Bank	Adressen	Funktion
0	0000-03FF	Betriebssystem / BASIC-Variablen
0	0400-07FF	40-Zeichen-Bildschirm
0	0800-0AFF	BASIC-Stack für gosub, return etc. / BASIC-Variablen
0	0B00-0DFF	Kassetten- und RS232-Puffer
0	0E00-0FFF	Sprite-Definitionen
0	1000-10FF	Funktionstasten-Definitionen
0	1100-12FF	BASIC-Variablen für Grafik usw.
0	1300-1BFF	Normalerweise unbenutzt
0	1C00-FEFF	Bereich für BASIC-Programm
0	FF00-FF04	Kopie der MMU-Register
0	FF05-FFFF	Kopie der IRQ,NMI und RESET-Routinen
1	0000-03FF	identisch mit Bank 0
1	0400-FFEF	Bereich für BASIC Variablen
1	FF00-FFFF	in Bank 0
2	0000-FFFF	beim normalen C-128 wie Bank 0, durch Speicherausbau erreichbar
3	0000-FFFF	beim normalen C-128 wie Bank 1, durch Speicherausbau erreichbar
4-7		unbenutzt, eventuell durch Speicherausbau nutzbar
14	D000-DFFF	Zeichengenerator
15	4000-AFFF	BASIC-Interpreter
	B000-BFFF	Maschinensprache Monitor
	C000-CFFF	Editor
	D000-DFFF	Ein-/Ausgabe-Bausteine
	E000-FF00	Kernal

Für alle Banks gilt, daß die Adressen $FF00 bis $FF04 immer "sichtbar" sind, also Vorrang vor allem anderen haben.

Durch Ändern der MMU-Register kann nun diese Aufteilung in gewissen Grenzen geändert werden; es treten aber gerade für den Assembler-Programmierer einige Schwierigkeiten auf. Nach dem Aufruf des Programms mit SYS wird normalerweise ein Programm aufgerufen, das sich in der Bank 0 im normalen BASIC-Speicher befindet, also ab Adresse $1C00 oder, bei kleineren Unterprogrammen, im reservierten Bereich von $1300 bis $1BFF. Beim Start des Assembler-Programms ist die gesamte Bank 0, mit Ausnahme der MMU-Register bei $FF00, mit RAM belegt, was ja auch durchaus sinnvoll ist. Wenn im Programm jetzt z.B. ein Zeichen auf dem Bildschirm ausgegeben werden soll, tritt das Problem auf, daß die Kernal-Routinen, bzw. auch die Editor-Routinen, momentan überhaupt nicht im Adressbereich vorhanden sind. Vor dem Aufruf der Kernal-Routine muß also die MMU umgeschaltet werden, damit im anzusprechenden Bereich das ROM aktiviert wird. Beispiele hierfür finden Sie im Beispielteil weiter hinten; hier geht es zunächst einmal um das grundsätzliche Verständnis.

Nach Abschluß der Kernal-Funktion muß die MMU wieder auf den ursprünglichen Zustand zurückgeschaltet werden. Es ist hierbei zu beachten, daß nach dem Umschalten der MMU in dem entsprechenden Bereich kein RAM mehr vorhanden ist. Das hat zur Folge, daß unter Umständen ein dauerndes Umschalten der MMU notwendig ist, falls sich die auszugebenden Daten ausgerechnet im Bereich "unter" den System-ROMS befinden.

Unbehelligt von diesen ganzen Umschaltvorgängen bleibt lediglich der Bereich von $0000 bis $3FFF, sofern die Umschaltung sich auf die System-ROMS bezieht. Besonders problematisch wird die Angelegenheit, wenn Sie auf Daten aus der anderen Speicherbank zugreifen wollen. Der Bereich von $0000 bis $3FFF, die Common Area, ist für beide RAM-Banks gleich; alles darüber ist pro Bank einzeln vorhanden. Wenn also Ihr Programm z.B. auf der Adresse $2000 plötzlich Daten aus der anderen Speicherbank benötigt, müßten Sie jetzt die MMU auf die andere Bank umschalten, ein Datenbyte lesen und wieder zurückschalten. Dummerweise schalten Sie hierbei aber auch Ihr Programm mit ab; der Systemabsturz ist geradezu vorprogrammiert.

1.1.5.2 Indirekte RAM-Routinen

Abhilfe schaffen hier nur "indirekte RAM-Routinen", die im *gemeinsamen* Bereich liegen, und von dort aus die Umschaltung vornehmen, ein Datenbyte lesen, zurückschalten und zurück zum Hauptprogramm springen. Der ganze Vorgang nimmt natürlich etliches an Zeit in Anspruch, womit der große Vorteil der doppelten Geschwindigkeit im 2-MHz-Betrieb wieder aufgezehrt wird.

Das Betriebssystem stellt einige dieser RAM-Routinen bereits zur Verfügung. Für besondere Funktionen ist aber im gemeinsamen Bereich kein weiterer Platz vorhanden. Hier hilft nur noch, den gemeinsamen Bereich durch Änderung des entsprechenden MMU-Registers zu vergrößern.

Die im Betriebssystem verfügbaren indirekten RAM-Routinen sind:

Adresse	Name	Funktion
$02A2	FETCH	Laden eines Bytes aus einer beliebigen Bank
$02AF	STASH	Speichern eines Bytes in eine beliebige Bank
$02BE	CMPARE	Vergleichen eines Bytes mit einer beliebigen Bank
$02CD	JSRFAR	Unterprogramm in beliebiger Bank ausführen
$02E3	JMPFAR	Programm in beliebiger Bank anspringen

FETCH

Diese Routine dient zum Lesen eines Bytes aus einer beliebigen Bank. Hierbei wird das Byte indirekt gelesen, wobei in der Adresse $02AA die hierzu verwendete Adresse aus der Zero-Page stehen muß. Das X-Register muß die Belegung für das MMU-Konfigurations-Register enthalten, mit der die gewünschte Bank addressiert wird. Das Y-Register enthält einen eventuellen Offset zur Basis-Adresse in der Zero-Page. Ein kleines Beispielprogramm:

```
*** Laden eines Bytes aus Bank 1, Adresse $1234

LDA  #$34        Low-Byte Adresse
STA  $FE         Zero-Page Adresse Low
LDA  #$12        High-Byte Adresse
STA  $FF         Zero-Page Adresse High
LDA  #$FE        Zero-Page Adresse
STA  $02AA       In der Routine speichern
LDY  #$00        Offset zur Adresse = 0
LDX  #$7F        MMU-CR Belegung für Bank 1
JSR  $02A2       Fetch aufrufen
.
.
.                Accu enthält das gelesene Byte
```

STASH

Mit Hilfe dieser Routine kann ein Byte an eine Adresse in einer beliebigen Bank geschrieben werden. Wie bei FETCH wird indirekt über eine Zero-Page-Adresse geschrieben, deren Adresse in $02B9 angegeben werden muß. Das X-Register enthält wieder den MMU-Wert und das Y-Register einen Offset zur Basisadresse. Auch hierzu wieder ein kleines Beispielprogramm:

```
*** Schreiben eines Bytes nach Bank 1, Adresse $1234

LDA  #$34        Low-Byte Adresse
STA  $FE         Zero-Page Adresse Low
LDA  #$12        High-Byte Adresse
STA  $FF         Zero-Page Adresse High
LDA  #$FE        Zero-Page Adresse
STA  $02B9       In der Routine speichern
LDY  #$00        Offset zur Adresse = 0
LDX  #$7F        MMU-CR Belegung für Bank 1
LDA  #$xx        Gewünschtes Byte
JSR  $02AF       Stash aufrufen
.
.
```

CMPARE

Mit Hilfe dieser Routine kann ein Byte an einer Adresse in einer beliebigen Bank mit dem Akku-Inhalt verglichen werden. Wie bei FETCH und STASH wird indirekt über eine Zero-Page-Adresse verglichen, deren Adresse in $02C8 angegeben werden muß. Das X-Register enthält wieder den MMU-Wert und das Y-Register einen Offset zur Basis-Adresse. Auch hierzu wieder ein kleines Beispielprogramm:

```
*** Vergleichen eines Bytes mit Bank 1, Adresse $1234

LDA  #$34        Low-Byte Adresse
STA  $FE         Zero-Page Adresse Low
LDA  #$12        High-Byte Adresse
STA  $FF         Zero-Page Adresse High
LDA  #$FE        Zero-Page Adresse
STA  $02C8       In der Routine speichern
LDY  #$00        Offset zur Adresse = 0
LDX  #$7F        MMU-CR Belegung für Bank 1
LDA  #$xx        Vergleichsbyte
JSR  $02BE       Cmpare aufrufen
.
.
```

JSRFAR

Unterprogramme in anderen RAM-Banks können nicht mit einem normalen JSR-Befehl aufgerufen werden, wenn die Zieladresse außerhalb des gerade adressierten Bereichs liegt. Mit Hilfe der Routine JSRFAR kann ein Programm in jeder Bank an beliebiger Adresse aufgerufen werden. Hierzu sind einige Vorbereitungen nötig, wenn der aufgerufenen Routine bestimmte Inhalte in den Prozessor-Registern mitgegeben werden sollen. In der Zero-Page befinden sich an den Adressen $0002 bis $0009 die Speicherzellen für die Prozessor-Register, die beim Aufruf an die Routine übergeben werden sollen.

```
$0002  =  Bank
$0003  =  Programmzähler Highbyte
$0004  =  Programmzähler Lowbyte
$0005  =  Status-Register
$0006  =  Akkumulator
$0007  =  X-Register
$0008  =  Y-Register
$0009  =  Stapelzeiger
```

Die Adresse der gewünschten Routine ist in den Adressen $0003 und $0004, die gewünschte Bank in der Adresse $0002 anzugeben. Die anderen Adressen müssen mit den gewünschten Inhalten der Prozessor-Register geladen werden.

Anschließend wird die Routine JSRFAR aufgerufen. Nach der Rückkehr enthalten die oben genannten Adressen die Prozessor-Inhalte, die vom Unterprogramm übergeben wurden.

JMPFAR

Diese Routine dient zum Anspringen von Programmen, vergleichbar einem JMP-Befehl. Das aufgerufene Programm muß also selbst für einen Rücksprung in die Originalbank sorgen. Für die Übergabe der Prozessor-Register bzw. der Programm-Startadresse gilt das unter JSRFAR gesagte.

Weitere Beispiele für solche indirekten RAM-Routinen finden Sie im Beispielteil weiter hinten in diesem Buch.

1.1.6 VDC - Video Controller für den 80-Zeichen-Bildschirm

Die Ansteuerung des 40-Zeichen-Bildschirms ist, wie oben angesprochen, für den Assembler-Programmierer relativ einfach, da ein Abbild des Bildschirm-Inhaltes stets im RAM zu finden ist, und die Änderung der entsprechenden Adressen auch eine Änderung des Bildschirminhaltes zur Folge hat.

Der 80-Zeichen-Bildschirm ist leider nicht so einfach zu bedienen, da das Bildschirm-RAM bei 80-Zeichen *nicht* im Adressbereich des Prozessors liegt. Der verwendete VDC hat, unabhängig von den 128 KByte des eigentlichen Rechners, eigene 16 KByte als Arbeitsspeicher zur Verfügung, die er völlig selbstständig verwaltet.

Die 16 KByte verteilen sich wie folgt:

Adresse	Funktion
0000-07FF	Bildschirmspeicher, 2 KByte
0800-0FFF	Attributspeicher für Farbe usw., 2 KByte
1000-1FFF	unbenutzt, 4 KByte
2000-2FFF	Zeichensatz für Grafik-/Großbuchstaben (4 KByte)
3000-3FFF	Zeichensatz für Klein-/Großbuchstaben (4 KByte)

Die Zeichensätze, und damit die Form der Bildschirmzeichen entsprechen denen des 40-Zeichen-Bildschirms. Sie werden beim Systemstart aus dem Zeichensatz-ROM in der Bank 14 in das RAM des VDC kopiert. Bei jedem Betätigen der CAPS-LOCK- bzw. ASCII/DIN-Taste wird der jeweils aktuelle Zeichensatz neu kopiert, was durch ein kurzes Stoppen der restlichen Rechneraktivität erkennbar ist. Diese Funktion wird im Interrupt durchgeführt und kann softwäremäßig auch verhindert werden (siehe Assemblerlisting).

Wenn jetzt ein Zeichen auf den Bildschirm gebracht werden soll, muß es also in das RAM des Video-Controllers übertragen werden. Solange Sie mit PRINT-Befehlen oder der entsprechenden Kernal-Routine ($FFD2) arbeiten, wird Ihnen die Arbeit vom Editor abgenommen; wenn Sie mit einem, dem POKE entsprechenden, Befehl arbeiten wollen, wird es etwas komplizierter. Zunächst sei erwähnt, daß Sie vom BASIC her keine Möglichkeit hierzu haben, es sei denn, sie verwenden hierfür eine Assembler-Unterroutine.

Jedes Ansprechen des VDC-RAMs muß immer über den VDC selbst erfolgen. Hierzu dienen zwei Speicherstellen im Adressbereich des Prozessors, die zwei Adressen im VDC darstellen. In die erste Adresse wird die Nummer des VDC-Registers geschrieben, auf das sich die Operation beziehen soll. Das zweite Register wird dann entweder mit dem eigentlichen Datenbyte beschrieben oder, je nach Funktion, ein Datenbyte von dort gelesen.

Das Steuerregister befindet sich auf Adresse $D600, das Datenregister auf Adresse $D601. Die eigentlichen Register des VDC liegen im Bereich von 0 bis 36, wobei für die normale Datenübergabe die Register 18, 19 und 31 interessant sind.

Der VDC ist übrigens auch in der Lage, Grafik zu erzeugen, und zwar ebenfalls Bitmap-Grafik mit einer Auflösung von 640 * 200 Punkten in einer Farbe. Hierzu werden die gesamten 16 KByte des VDC-RAMs verwendet, die nach der entsprechenden Umschaltung im Register 25, Bit 7 des VDC als Punktdaten behandelt werden. Die Darstellung von Grafik und Bauchstaben ist allerdings nicht gleichzeitig möglich, es sei denn, die Buchstaben werden aus Punkten selbst zusammengesetzt. Nachteilig ist auch, daß diese Grafik erstens vom BASIC nicht unterstützt wird, sondern wie beim C-64 vom Anwender selbst, und zwar ausschließlich, in Assembler programmiert werden muß, und zweitens zum Zurückschalten in den normalen Betrieb erst wieder die normalen Zeichensätze geladen werden müssen.

Beispielprogramme für die Datenübergabe an den VDC finden Sie im Beispielteil. Die dort angegebenen Unterroutinen können Sie in eigenen Programmen verwenden.

Der Video-Controller VDC 8563 belegt nur die Adressen $D600 und $D601, verwaltet jedoch intern 36 Register.

Auf die Register wird zum Schreiben/Lesen zugegriffen, indem die gewünschte Registernummer ins Adreßregister geschrieben und das Datenregister geschrieben/gelesen wird.

1.1.6.1 Interne Register des Video Display Controllers VDC 8563

Adresse	Bits	Funktion
$D600		Adressregister
	..543210	WRITE: Nummer gewähltes Register (0-36)
	7.......	READ: Statusbit des VDC
		0 letzter VDC-Job noch in Arbeit 1 VDC bereit für nächsten Job

Da die Firma Commodore mit Informationen über die von ihr entwickelte Chips äußerst sparsam umgeht, sind über die weiteren Statusbits nur Spekulationen möglich (Vergleich mit 6845):

.6.....	Status Light-Pen Register:
	0 nach Lesen Reg. 16/17 (LPEN) 1 Light-Pen Strobe festgestellt
..5.....	Vertical Blanking: Kathodenstrahl
	0 nicht im Rücklauf 1 im Rücklauf

Die Bits 4 bis 0 werden im 6845 nicht benutzt. Im C 128 werden Bits 0 bis 3 des 8563 abgefragt ($E17E). Die Bedeutung ist jedoch unklar.

$D601 VDC 8563 Datenregister

Reg.	Bits	Funktion
0	76543210	gesamte Anzahl der Zeichen zwischen aufeinanderfolgenden horizontalen Synchronimpulsen-1
1	76543210	Anzahl Zeichen pro Bildschirmzeile
2	76543210	Position des horizontalen Synchronimpulses
3	7654.... 3210	Breite des vertikalen Synchronimpulses Breite des horizontalen Synchronimpulses (Zeichengenerator-Taktzyklen+1)
4	76543210	Gesamtzahl BS-Zeilen-1
5	...43210	Feinabgleich (Anzahl der Bildzeilen)
6	76543210	Anzahl der dargestellten Zeilen
7	76543210	Position des vertikalen Synchronimpulses (Anzahl Zeichenzeilen)

8	10		Anzeigeform:
		xO	kein Zeilensprung
		01	Zeilensprung
		11	Zeilensprung doppelte Dichte
9	...43210		Anzahl der Bildzeilen-1 pro BS-Zeile
10	.65.....		Cursor-Modus:
		00	Cursor blinkt nicht
		01	Cursor ausgeschaltet
		10	Cursor blinkt schnell (1/16 Bildwiederholfrequenz)
		11	Cursor blinkt langsam (1/32 Bildwiederholfrequenz)
	...43210		Cursor-Anfangszeile im Zeichen (0-7)
11	...43210		Cursor-Endzeile im Zeichen (0-7)
12	76543210		addrh Startadresse BS-Speicher (Video-RAM)
13	76543210		addrl Startadresse BS-Speicher (Video-RAM)
14	76543210		addrh akt. Cursorposition im BS-Speicher
15	76543210		addrl akt. Cursorposition im BS-Speicher
16	76543210		vertikale Lichtgriffel-Positioen (in Bildzeilen+1)
17	76543210		horizontale Lichtgriffel-Positioen (in Bildspalten+1)
18	76543210		addrh aktuelle Schreib-/Leseposition im BS-Speicher
19	76543210		addrl aktuelle Schreib-/Leseposition im BS-Speicher
20	76543210		addrh Startadresse Attribut-Speicher
21	76543210		addrl Startadresse Attribut-Speicher
22	7654....		Zeichenbreite in Pixel-1 (einschließlich Zwischenraum)
	3210		Zeichenbreite in Pixel-1 (ohne Zwischenraum)
23	...43210		Anzahl Bildzeilen/BS-Zeile (ohne Zwischenraum)
24	7.......		Block Copy
		0	Block wird geschrieben, danach Schreiben in Register 30

		1	Block wird kopiert, danach Schreiben in Register 30
	.6.....		
		0	Normaldarstellung
		1	REVERS-Modus
	..5.....		Blinkfrequenz der Bildschirmzeichen
		0	schnell (1/16 Bildwiederholfrequenz)
		1	langsam (1/32 Bildwiederholfrequenz)
	...43210		Anzahl der BS-Zeilen, die BS nach oben kopiert wird
25	7.....		Betriebsart
		0	Text-Modus
		1	Bit-Map-Modus
	.6.....		Attribute
		0	kein Attribut, Vordergrundfarbe laut Register 26
		1	Vordergrundfarbe und Attribut laut Register 26
	..5.....		Semigraphik-Modus
		0	horizontaler Zwischenraum zweier Zeichen in Hintergrundfarbe
		1	horizontaler Zwischenraum wie letztes normal dargestelltes Zeichen
	...4....		Pixelgröße
		0	1 Dot-Takt
		1	2 Dot-Takte
	3210		Anzahl Pixel, um die Bildschirm nach links geschoben wird
26	7654....		Vordergrundfarbe, wenn Bit 6 von Register 25 = 0
	3210		Hintergrundfarbe für alle Zeichen und Rahmenfarbe
27	76543210		Anzahl Zeichen/BS-Zeile
28	765.....		Adreßbits 15-13 des Zeichengenerators
	...4....		DRAM-Type
		0	4416
		1	4164
29	...43210		BS-Zeilen Zähler für Unterstreichen

30	76543210	Anzahl Block-Write- bzw. Block-Copy-Zyklen
31	76543210	Daten für Block-Write bzw. normales Write
32	76543210	addrh Quelladresse Block-Copy
33	76543210	addrl Quelladresse Block-Copy
34	76543210	Zeichen vom Anfang der aktuellen BS-Zeile bis zur positiven Flanke des Display-Enable-Pins
35	76543210	Zeichen vom Anfang der aktuellen BS-Zeile bis zur negativen Flanke des Display-Enable-Pins
36	3210	Anzahl der Refresh-Zyklen pro BS-Zeile

1.1.7 CIA - Ein-/Ausgabe-Baustein

Die im C-128 verwendeten CIA-Bausteine dienen zur Abfrage der Tastatur, zur Bedienung des Kassettenrekorders, des User-Ports und der RS232-Schnittstelle. Es existieren zwei CIAs, die auf den Adressen $DC00 bis $DC0F bzw. $DD00 bis $DD0F liegen. Beide CIAs sind identisch, haben aber unterschiedliche Funktionen im System. Im folgenden werden die einzelnen CIA-Register am Beispiel des CIAs mit der Basis-Adresse $CC00 erläutert:

Adresse	Name	Bedeutung
CC00	PRA	Peripheres Datenregister A
CC01	PRB	Peripheres Datenregister B
CC02	DDRA	Datenrichtungsregister A
CC03	DDRB	Datenrichtungsregister B
CC04	TA Lo	Zeitgeber A Lowbyte
CC05	TA Hi	Zeitgeber A Highbyte
CC06	TB Lo	Zeitgeber B Lowbyte
CC07	TB Hi	Zeitgeber B Highbyte
CC08	TOD 10tel	Echtzeituhr 10tel Sekunden
CC09	TOD Sec	Echtzeituhr Sekunden
CC0A	TOD Min	Echtzeituhr Minuten
CC0B	TOD Hr	Echtzeituhr Stunden
CC0C	SDR	Serielles Datenregister
CC0D	ICR	Interrupt Kontroll-Register
CC0E	CRA	Kontroll-Register A
CC0F	CRB	Kontroll-Register B

PRA, PRB, DDRA, DDRB

Diese vier Register stellen zwei Ein-/Ausgabe-Ports dar, die vom Prozessor beliebig verwendet werden können. Die Register DDRA bzw. DDRB sind die Datenrichtungsregister, die bitweise die Datenrichtung der einzelnen

Portleitungen bestimmen. Jedes im Datenrichtungsregister gesetzte Bit stellt die entsprechende Datenleitung auf Ausgang, jedes nicht gesetzte Bit auf Eingang.

Die Register PRA und PRB sind die eigentlichen Ein-/Ausgabe-Register. Ein Schreibzugriff setzt die als Ausgang geschalteten Leitungen auf logisch High, sofern das entsprechende Bit im eingeschriebenen Byte gesetzt ist. Ein Zugriff liest die momentan anliegende Information der Eingangsleitungen.

TA, TB

Die CIA enthält zwei Timer, die zur Zeitmessung oder zur Erzeugung von periodischen Impulsen oder Programm-Unterbrechungen (Interrupts) verwendet werden können. Die Timer im CIA#1 ($CC00) werden vom System selbst verwendet und sollten daher nicht benutzt werden. Die Timer im CIA#2 ($DD00) werden nur während des RS232-Betriebs benutzt und können ansonsten frei verwendet werden.

Der Timer kann, in Abhängigkeit vom jeweiligen Steuerregister CRA bzw. CRB, entweder mit dem Systemtakt von 1 MHz oder einem Takt an einem zusätzlichen CIA-Anschluß getaktet werden. Für normale Anwendungen sollte der Systemtakt als Grundlage dienen.

In die beiden Timer-Register Lo und Hi kann eine 16-Bit-Zahl eingeschrieben werden, die dann nach dem Start des Timers mit dem Takt auf Null runtergezählt wird. Beim Erreichen von Null wird das entsprechende Bit im Interupt-Register gesetzt und, sofern freigegeben, ein Interrupt ausgelöst.

CRA

Das Kontrollregister CRA steuert im wesentlichen die Bedienung des Timers A. Die acht Bits des Register haben folgende Bedeutung:

Bit	Beschreibung
0	Timer wird gestoppt, wenn das Bit "0" ist. Timer wird gestartet, wenn das Bit "1" ist.
1	Wenn das Bit "1" ist, wird bei jedem Erreichen von Null bei Timer A ein Impuls auf Leitung 6 des Ports B ausgegeben (Takterzeugung). Wenn das Bit "0" ist, bleibt Leitung 6 unverändert.

2	Wenn das Bit "0" ist, wird (bei gesetztem Bit 1) an Leitung 6 von Port B jeweils in Impuls ausgegeben. Wenn das Bit "1" ist, wird der Zustand der Leitung jeweils umgeschaltet.
3	Wenn das Bit "0" ist, läuft der Timer A nach dem Start kontinuierlich weiter. Wenn das Bit "1" ist, stoppt der Timer automatisch beim Erreichen von Null.
4	Wenn das Bit "1" ist, werden die zuvor in die Timer-Register geschriebenen Werte in das eigentliche Zählwerk des Timers übernommen.
5	Wenn das Bit "0" ist, zählt der Timer die Systemtakt-Impulse. Wenn das Bit "1" ist, werden die Impulse an der CIA-Leitung "CNT" gezählt.
6	Wenn das Bit "0" ist, dient der serielle Port (SDR) als Eingang. Wenn das Bit "1" ist, dient der SDR als Ausgang.
7	Wenn das Bit "0" ist, werden für die Echtzeituhr (TOD) 60 Hz am Zähleingang angenommen. Wenn das Bit "1" ist, werden 50 Hz angenommen.

CRB

Das Kontrollregister CRB steuert im wesentlichen die Bedienung des Timer B. Die acht Bit des Registers haben folgende Bedeutung:

Bit	Beschreibung
0	Timer wird gestoppt, wenn das Bit "0" ist. Timer wird gestartet, wenn das Bit "1" ist.
1	Wenn das Bit "1" ist, wird bei jedem Erreichen von Null bei Timer B ein Impuls auf Leitung 7 des Ports B ausgegeben (Takterzeugung, Tonfrequenzen). Wenn das Bit "0" ist, bleibt Leitung 7 unverändert.
2	Wenn das Bit "0" ist, wird (bei gesetztem Bit 1) an Leitung 7 von Port B jeweils in Impuls ausgegeben. Wenn das Bit "1" ist, wird der Zustand der Leitung jeweils umgeschaltet.
3	Wenn das Bit "0" ist, läuft der Timer B nach dem Start kontinuierlich weiter. Wenn das Bit "1" ist, stoppt der Timer automatisch beim Erreichen von Null.
4	Wenn das Bit "1" ist, werden die zuvor in die Timer-Register geschriebenen Werte in das eigentliche Zählwerk des Timers übernommen.

5+6	5=0 6=0: Timer B zählt Systemtakt. 5=1 6=0: Timer B zählt CNT-Impulse. 5=0 6=1: Timer B zählt Timer A Null-Durchgänge. 5=1 6=1: Timer B zählt Timer A Null-Durchgänge, solange die Leitung "CNT" auf High liegt.
7	Wenn das Bit "0" ist, arbeiten Schreib/Lesezugriffe auf die Echtzeitregister mit der Echtzeituhr. Die Uhrzeit wird gestellt oder gelesen. Wenn das Bit "1" ist, werden die Register für die programmierbare Alarmzeit angesprochen.

TOD

TOD (Time of Day) stellt eine programmierbare Echtzeituhr dar, die mit der Genauigkeit der Netzfrequenz läuft. Je nach Netzfrequenz muß das Bit 7 im CRA entsprechend gesetzt sein, ansonsten geht die Uhr falsch. Die Uhr enthält insgesamt vier Register für Stunden, Minuten, Sekunden und 1/10 Sekunden. Alle Daten sind BCD-kodiert, d.h. die oberen und unteren 4 Bit stellen jeweils eine Dezimalzahl im Bereich von 0 bis 9 dar. Die Zehner der Stunden zählen nur von 0 bis 11, da der Baustein im amerikanischen Sprachraum entwickelt wurde und dort mittels "am" (ante meridiem = vormittags) und "pm" (post meridiem = nachmittags) die Unterscheidung zwischen z.B. 1 Uhr und 13 Uhr vorgenommen wird. Entsprechend wird im Stunden-Register das oberste Bit (Bit 7) auf 0 (AM) oder 1 (PM) gesetzt und muß zur korrekten Uhrzeitanzeige beachtet werden.

Die Echtzeituhren in beiden CIAs werden vom Betriebssystem im übrigen nicht benutzt, können also frei für eigene Zwecke verwendet werden. Die interne Uhr des C-128 wird durch einfaches Zählen von Interrupts erzeugt, wie schon immer bei den anderen Commodore-Rechnern.

Beim Einschreiben oder Auslesen der Uhrzeit muß jeweils eine bestimmte Reihenfolge beachtet werden. Das Schreiben der Uhrzeit muß immer in der Reihenfolge von Stunden nach 1/10 Sekunden erfolgen. Beim Schreiben in das Stundenregister wird die Uhr angehalten und startet erst wieder beim Schreiben der 1/10 Sekunden, um Überläufe während des Setzvorganges zu vermeiden.

Beim Lesen der Uhr muß die gleiche Reihenfolge eingehalten werden, um Überläufe während des Lesevorganges zu vermeiden. Die Uhr wird während des Lesens jedoch nicht angehalten.

Zusätzlich zur Uhrzeit kann eine beliebige Alarmzeit gesetzt werden, bei deren Erreichen das entsprechende Bit im Interrupt-Kontrollregister (ICR, siehe dort) gesetzt und, je nach Steuerung, dort ein Interrupt ausgelöst

wird. Das Bit 7 im CRB bestimmt, ob Schreib- oder Lesevorgänge sich auf die Uhrzeit (Bit = 0) oder auf die Alarmzeit (Bit = 1) beziehen.

ICR

Das ICR (Interrupt-Kontrollregister) bestimmt, welche Ereignisse innerhalb der CIA (Timer-Unterlauf, Alarmzeit usw.) einen Interrupt auslösen sollen.

Die einzelnen Bits haben folgende Bedeutung:

Bit	Funktion
0	Das Bit wird gesetzt, wenn ein Unterlauf (Null erreicht) bei Timer A auftritt.
1	Das Bit wird gesetzt, wenn ein Unterlauf (Null erreicht) bei Timer B auftritt.
2	Das Bit wird gesetzt, wenn die programmierte Alarmzeit erreicht ist.
3	Das Bit wird gesetzt, wenn das serielle Datenregister (SDR), je nach Funktion, voll oder leer ist.
4	Das Bit wird gesetzt, wenn ein negativer Übergang am FLAG-Pin der CIA auftritt.
5+6	Unbenutzt, Wert beim Lesen immer Null.
7	Steuerbit

Um einzelne Bits zur Veranlassung eines Interrupts zu setzen, muß eine entsprechende Bitkombination in das ICR eingeschrieben werden, wobei das Steuerbit bestimmt, welche Bits zu setzen sind. Wenn das Steuerbit "0" ist, werden alle in der Bitkombination gesetzten Bits im ICR gelöscht, alle nicht gesetzten Bits bleiben unverändert. Ist das Steuerbit "1", werden alle in der Bitkombination gesetzten Bits im IFR gesetzt, alle nicht gesetzten Bits lassen die entsprechenden Interruptbits unverändert.

Beim Lesen des ICR zeigt das Steuerbit an, ob die CIA einen Interrupt ausgelöst hat (Bit 7 = "1") oder nicht (Bit 7 = "0"). Ist das Bit "1", können die restlichen Bits ausgewertet werden, um festzustellen, welches Ereignis eingetreten ist.

Anzumerken ist noch, daß die CIA #1 immer einen IRQ auslöst und die CIA #2 immer einen NMI. Vom Betriebssystem wird der NMI zur Steuerung der RS232-Schnittstelle verwendet. Außerdem wird bei jedem Drükken der RESTORE-Taste ebenfalls ein NMI ausgelöst.

SDR

Die CIA enthält ein serielles Datenregister, das wahlweise als Ausgang oder als Eingang arbeiten kann. Die Steuerung hierüber wird mit dem Bit 6 des CRA vorgenommen. Im Eingangsmodus werden jeweils acht aufeinander folgende Datenbits eingelesen und anschließend das entsprechende Bit im ICR gesetzt und wahlweise ein Interrupt ausgelöst. Die Taktrate für das Einlesen wird über den Pin CNT der CIA bestimmt. Bei jedem positiven Übergang an diesem Pin wird das nächste Bit eingelesen.

Im Ausgangsmodus bestimmt der Timer A die Taktfrequenz. Ein in das SDR eingeschriebenes Byte wird bitweise mit der halben Frequenz des Timers ausgegeben und nach der Ausgabe aller 8 Bits wieder das entsprechende Bit im ICR gesetzt und wahlweise ein Interrupt ausgelöst.

Mit Hilfe des SDR ist eine serielle Schnittstelle beliebiger Baudrate programmierbar, die z.B. zur Kombination mit anderen Rechnersystemen eingesetzt werden kann. Bei der Übertragung der einzelnen Bits ist zu beachten, daß immer das Bit 7 zuerst übertragen oder eingelesen wird.

Die beiden CIAs werden vom System für die verschiedensten Steuerzwecke verwendet. Eine genaue Dokumentation der verwendeten Register und Bits können Sie der folgenden Tabelle entnehmen. Dort sind die einzelnen Bits mit ihrer Bedeutung im C-128-System erläutert.

Im Beispielteil finden Sie zu diesem Thema ein Beispiel einer Parallel-Schnittstelle, realisiert über den User-Port des C-128.

1.1.7.1 Interne Register des Complex Interface Adapter 1 CIA 6526

Adresse	Bits	Funktion
$DC00	76543210	Spalten Tastaturmatrix (OUTPUTS)
	außerdem:	
	76.....	Auswahl Paddle-Set A/B
	...43210	Joystick 0
$DC01	76543210	Reihen Tastaturmatrix
	außerdem:	
	...43210	Joystick 1
$DC02	76543210	Datenrichtungsregister A Alle Bits als Ausgänge geschaltet zum Scannen der Tastatur

```
$DC03     76542310     Datenrichtungsregister B
                       Alle Bits als Eingänge geschaltet
                       zum Scannen der Tastatur

$DC04     76543210     Timer A Low-Register
$DC05     76543210     Timer A High-Register

                       benutzt für Serial Device
                       und Kassettenrekorder

$DC06     76543210     Timer A Low-Register
$DC07     76543210     Timer A High-Register
$DC08     7654....     unbenutzt
          ....3210     Zehntelsekunden (BCD)

$DC09     7654....     Sekunden Zehner (BCD)
          ....3210     Sekunden Einer (BCD)

$DC0A     7654....     Minuten Zehner (BCD)
          ....3210     Minuten Einer (BCD)


$DC0B     7.......        PM / AM - Flag
                       0  AM (Vormittags)
                       1  PM (Nachmittags)

          .65.....     unbenutzt
          ...4....     Stunden Zehner (BCD)
          ....3210     Stunden Einer (BCD)


$DC0C     76543210     Serial Daten-Register


$DC0D                  IRQ-Flagregister
          READ:

          7.......     IR IRQ-Flag (1 = IRQ)
          .65.....     unbenutzt (immer 0)

          ...43210     IRQ-Flags:  1 = IRQ
                                   0 = NO IRQ
          ...4....     FLAG-IRQ (Flag-Input =1)
          ....3...     SERIAL PORT (full/empty)
          .....2..     ALARM (Alarmzeit erreicht)
          ......1.     TIMER B (underflow)
          .......0     TIMER A (underflow)

          WRITE:

          7.......        SET/CLEAR Bit
                       0  CLEAR alle Maskenbits = 1
                       1  SET   alle Maskenbits = 1

          .65.....     unbenutzt
```

```
          ...43210      Maskenbits: 1 = SET/CLEAR
                                    0 = UNAFFECTED
          ...4....      Maskenbit FLAG IRQ
          ....3...      Maskenbit SERIAL PORT
          .....2..      Maskenbit ALARM
          ......1.      Maskenbit TIMER B
          .......0      Maskenbit TIMER A


$DC0E     7.......        TODIN
                       0  60 Hz an TOD-Pin
                       1  50 Hz an TOD-Pin

          .6......        SPMODE
                       0  SERIAL PORT INPUT (ext. Shift-Takt erforderl.)
                       1  SERIAL PORT OUTPUT (CNT als Shift-Takt)

          ..5.....        INMODE
                       0  TIMER A zählt PHI2 Pulse
                       1  TIMER A zählt positive CNT-Übergänge

          ...4....        LOAD
                       0  wirkungslos
                       1  FORCE LOAD

          ....3...        RUNMODE
                       0  CONTINOUS
                       1  ONE-SHOT

          .....2..        OUTMODE
                       0  PULSE
                       1  TOGGLE

          ......1.        PBON Steuerung PB6
                       0  PB6 (Bit6 Port B) normal Operation
                       1  PB6 ist TIMER A output

          .......0        START TIMER A
                       0  Stop TIMER A
                       1  Start TIMER A

$DC0F     7.......        ALARM
                       0  Write TOD-Register setzt Uhr
                       1  Write TOD-Register setzt Alarm

          .65.....        INMODE
                      00  TIMER B zählt PHI2-Pulse
                      01  TIMER B zählt positive CNT-Übergänge
                      10  TIMER B zählt TIMER A underflow-Pulse
                      11  TIMER B zählt TIMER A underflow-Pulse wenn CNT hi

          ...4....        LOAD
                       0  wirkungslos
                       1  FORCE LOAD

          ....3...        RUNMODE
                       0  CONTINOUS
                       1  ONE-SHOT
```

```
.....2..      OUTMODE
           0  PULSE
           1  TOGGLE

......1.      PBON Steuerung PB7
           0  PB7 (Bit7 Port B) normal Operation
           1  PB7 ist TIMER B output

.......0      START TIMER B
           0  Stop TIMER B
           1  Start TIMER B
```

1.1.7.2 Interne Register des Complex Interface Adapter 2 CIA 6526

Adresse	Bits	Funktion
$DD00	7.......	DATA Input Serial Device
	.6......	CLOCK Input Serial Device
	..5.....	DATA Output Serial Device
	...4....	CLOCK Output Serial Device
	....3...	ATN Output Serial Device
	.....2..	TXD Transmit Data
	10	Adressbits 15/14 BS-RAM
$DD01	7.......	DSR Data Set Ready
	.6......	CTS Clear to Send
	..5.....	DCD Data Carrier Detect
	...4....	
	....3...	RI Ring Indicator
	.....2..	DTR Data Terminal Ready
	......1.	RTS Request to Send
	0	RXD Received Data
$DD02	76543210	Datenrichtungsregister A
	0.......	INPUT DATA Serial Device
	.0......	INPUT CLOCK Serial Device
	..1.....	OUTPUT DATA Serial Device
	...1....	OUTPUT CLOCK Serial Device
	....1...	OUTPUT ATN Serial Device
	.....1..	OUTPUT TXD Transmit Data
	11	OUTPUT Adressbits 15/14 BS-RAM
$DD03	76543210	Datenrichtungsregister B
	0.......	INPUT DSR Data Set Ready
	.0......	INPUT CTS Clear to Send
	..0.....	INPUT DCD Data Carrier Detect
	...0....	INPUT unbenutzt
	....0...	INPUT RI Ring Indicator
	.....1..	OUTPUT DTR Data Terminal Ready
	......1.	OUTPUT RTS Request to Send
	0	INPUT RXD Received Data

Restliche Register wie CIA 1

1.2 Das Betriebssystem des C-128

1.2.1 Einführung

Jeder Computer, also auch der C-128, braucht ein Betriebssystem; ein Programm, das z.B. den Datenverkehr mit der Umwelt regelt, und ganz allgemein für ein Zusammenspiel der einzelnen Komponenten sorgt.

Das Betriebssystem des C-128 gliedert sich in mehrere Teile, die in einer gewissen Hierarchie zu sehen sind. Ganz oben in der Priorität steht das "Kernal", das eigentliche Betriebssystem der Rechners. Hier werden die Peripherie-Bausteine verwaltet und damit die Eingabe von Zeichen über die Tastatur, die Ausgabe von Zeichen auf dem Bildschirm und der Datenverkehr z.B. mit der Floppy oder einem Drucker gehandhabt.

Die nächste Stufe in der Hierarchie ist der Bildschirm-Editor. Er sorgt dafür, daß man nicht nur einzelne Zeichen auf der Tastatur eingeben kann, sondern auch mit den Steuertasten (z.B. den Cursortasten) frei auf dem Schirm arbeiten, Zeichen löschen, einfügen und ändern kann.

Die oberste Stufe der Hierarchie ist dann der BASIC-Interpreter. Er sorgt dafür, daß der Benutzer Programme schreiben und ausführen, sie mit bestimmten Befehlen auf der Diskette oder dem Kassettenrekorder speichern und von dort wieder laden kann. Der BASIC-Interpreter greift auch wieder auf Kernal-Routinen zu und ruft, zum Editieren von Programmen oder zur Dateneingabe, auch den Editor auf.

In der Regel sind BASIC-Interpreter so geschrieben, daß sie nicht direkt auf Peripherie-Bausteine zugreifen, sondern den Datenaustausch über festgelegte Routinen des Kernals durchführen. Beim C-128 haben die Programmierer des Interpreters allerdings etwas "geschlampt". Viele Funktionen, z.B. die Klangerzeugung beim PLAY-Befehl, laufen völlig innerhalb des BASIC-Interpreters ab.

In diesem Buch befassen wir uns ausschließlich mit dem eigentlichen Betriebssystem des Rechners, also dem Kernal, dem Monitor und dem 40/80-Zeichen-Bildschirmeditor. Beim Durcharbeiten des Assemblerlistings werden Sie bemerken, daß das Betriebssystem mit zwei Ausnahmen völlig in sich geschlossen ist, also seinerseits keine Routinen des BASIC-Interpreters aufruft. Die einzigen Ausnahmen sind ein eventueller Aufruf einer BASIC-Interrupt-Routine, z.B. zur Bearbeitung der PLAY-Funktion oder einer Sprite-Kollision, und der Aufruf des Maschinensprache-Monitors.

1.2.2 Der Monitor

Das eingebaute Monitorprogramm ermöglicht das einfache Anzeigen und Ändern von Bytes im Speicher des C-128, das Anzeigen der internen Prozessorregister, und ermöglicht durch einen eingebauten Assembler auch das Eingeben kleinerer Maschinenprogramme.

Der Monitor wird vom BASIC her mit dem Befehl "MONITOR" aufgerufen. Außerdem wird in den Monitor verzweigt, wenn in eigenen Maschinenprogrammen ein BRK-Befehl gefunden wird oder, was ja auch von Zeit zu Zeit vorkommen kann, der Rechner ein wenig aus dem Takt kommt.

Im Monitor stehen Ihnen folgende Befehle zur Verfügung:

Befehl	Funktion
A xxxx aaaa	Assemblierung einer Zeile im mnemonischen Befehlscode. xxxx ist die Adresse, an die assembliert werden soll, aaaa ist eine gültige Zeile in 6502-Syntax, z.B. lda #$05.
C xxxx yyyy zzzz	Vergleichen zweier Speicherbereiche. xxxx ist die Startadresse, yyyy die Endadresse des einen Bereichs; zzzz ist die Startadresse des anderen. Alle Unter- schiede in den Speicherbereichen werden durch Angabe der jeweiligen Adresse im ersten Bereich angegeben.
D xxxx (yyyy)	Ein Speicherbereich wird disassembliert, d.h. in 6502-Syntax mnemonisch ausgegeben. xxxx ist die Startadresse des Bereiches, yyyy ist wahlweise die Endadresse. Ohne Endadresse wird eine halbe Bildschirmseite disassembliert.
F xxxx yyyy zz	Füllen eines Speicherbereiches von xxxx bis yyyy mit einem angegebenen Byte.
G xxxx	Starten eines Maschinenprogrammes an der angegebenen Adresse. Zuvor können die Prozessorregister mit dem R-Befehl angesehen und geändert werden. Durch ein BRK gelangt man wieder in den Monitor.

J xxxx	Starten eines Maschinenprogrammes an der angegebenen Adresse. Zuvor können die Prozessorregister mit dem R-Befehl angesehen und geändert werden. Durch ein RTS gelangt man wieder in den Monitor.
H xxxx yyyy zz ..	Durchsuchen des Speicherbereiches von xxxx bis yyyy nach dem angegebenen Bytes. Statt Hexbytes kann hier auch eine Zeichenfolge im ASCII-Code angegeben werden, wenn dieser ein Apostroph vorangestellt wird ('abc).
J xxxx	Aufrufen eines Unterprogrammes an der angegebenen Adresse. Zuvor können die Prozessorregister mit dem R-Befehl angesehen und geändert werden. Anschließend wird wieder in den Monitor gesprungen.
L "name",xx,yyyy	Laden eines Programms von Gerät xx an die Adresse yyyy. Fehlt die Ladeadresse, wird das Programm an die Originaladresse des Programms geladen.
M xxxx yyyy	Anzeigen des Speicherinhaltes von xxxx bis yyyy in Hexa-Darstellung und ASCII-Code. Fehlt die zweite Adress- angabe, werden 12 Zeilen dargestellt, also 96 Bytes bei 40-Zeichen, 192 Bytes bei 80-Zeichen-Darstellung.
R	Anzeigen der Prozessor-Register
S "Name",xx,yy,zz	Sichern eines Speicherbereiches von yyyy bis zzzz auf Gerät xx.
T xxxx yyyy zzzz	Kopieren des Speicherbereiches von xxxx bis yyyy an die Adresse zzzz. Es findet hier keine Adressumrechnung für Sprungadressen statt. Ein entsprechender Befehl "N" wie in anderen Monitorprogrammen zum Umrechnen von absoluten Sprungadressen fehlt.

V "name",xx,yyyy	Vergleichen eines Programms von Gerät xx mit dem Speicher an Adresse yyyy. Fehlt die Adressangabe, wird das Programm mit der Originaladresse des Programms verglichen.
X	Monitor verlassen
> xxxx zz ...	Unterbefehl von "M". Bis zu 8 (16) Bytes können an die angegebene Adresse übernommen werden.
; xx yy ...	Unterbefehl von "R". Hiermit können die Prozessorregister geändert werden.
.	Gleiche Funktion wie "A".
‘	Übergabe von Floppy-Befehlen. Ohne weitere Angaben wird die letzte Fehlermeldung der Floppy angezeigt. Ansonsten können beliebige Befehle an die Floppy übermittelt werden.
$ xxxx	Umrechnen eines Hexadezimalen Wertes. Angezeigt wird der Wert jeweils in hexadezimal, dezimal, oktal und binär.
+	Umrechnen eines Dezimalwertes.
&	Umrechnen eines Oktalwertes.
%	Umrechnen eines Binärwertes.

Alle Adressangaben können bis zu fünfstellig sein: vier Stellen für die Adresse plus eine Stelle für die Bank.

Innerhalb des Monitors sind einige Routinen vorhanden, die durchaus sinnvoll von Assemblerprogrammen verwendet werden können. Hier eine kurze Zusammenstellung der wichtigsten Routinen:

Adresse	Name	Funktion
$B89F	putwrd	Adresse als Hexa-ASCII ausgeben
$B8A4	puthxs	Byte in Akku als Hexa-ASCII + Space ausgeben
$B8C2	puthex	Byte in Akku als Hexa-ASCII ausgeben

$B8D2	makhex	Byte in Akku nach Hexa-ASCII wandeln
$BA0A	bindec	Wandlung Hexa-Dezimal nach BCD
$BA5D	unpack	Wert in beliebiger Basis ausgeben

putwrd **$B89F** **Adresse als Hexa-ASCII ausgeben**

Eine Zwei-Byte Adresse die sich im Akku (Low-Byte) und X-Register (High-Byte) befindet wird nach Hexa-ASCII gewandelt und auf dem aktiven Kanal ausgegeben. Anschließend wird noch ein Leerzeichen ausgegeben, wie im folgenden Beispiel:

```
LDA  $FC     ;Low-Byte des Wertes
LDX  $FD     ;High-Byte der Adresse
JSR  $B89F   ;wandeln und ausgeben
```

```
$FC = $12
$FD = $34
```

Ausgabe auf Schirm: 1234b (b = Leerzeichen)

puthxs **$B8A5** **Byte in Akku als Hexa-ASCII und Space ausgeben**

Das Byte im Akku wird als zwei Hexa-ASCII-Zeichen auf dem aktiven Kanal ausgegeben. Anschließend wird noch ein Leerzeichen ausgegeben.

puthex **$B8C2** **Byte in Akku als Hexa-ASCII ausgeben**

Das Byte im Akku wird als zwei Hexa-ASCII-Zeichen auf dem aktiven Kanal ausgegeben. Ein folgendes Leerzeichen wird nicht ausgegeben.

makhex **$B8D2** **Byte in Akku nach Hexa-ASCII wandeln**

Das Byte im Akku wird in zwei Hexa-ASCII-Zeichen umgewandelt, wobei nach der Rückkehr aus der Routine das obere Nibble im Akku und das untere Nibble im X-Register steht. Eine Ausgabe auf dem aktiven Kanal findet nicht statt.

bindec **$BA0A** **Wandlung Hexadezimal nach BCD**

Diese Routine wandelt einen maximal 6-stelligen Hexadezimal-Wert in das BCD-System um. BCD (Binary Coded Decimal) bedeutet, daß die jeweils

zwei Dezimalziffern in einem Byte kodiert werden, wobei in jedem Nybble eine Ziffer von 0 bis 9 steht.

Der zu wandelnde Wert muß sich in der Zero-Page in den Adressen $66 bis $68 befinden, wobei in $68 das höchstwertige Byte und in $66 das niedrigstwertige Byte stehen muß. Nach der Rückkehr aus der Routine steht das Ergebnis in den Speicherstellen $0AA0 bis $0AA3, wobei in $0AA0 das höchstwertige Byte und in $0AA3 das niedrigstwertige Byte steht.

In den Speicherstellen $66 bis $68 steht z.B. der hexadezimale Wert $3039 in der Reihenfolge:

```
$68 = 00
$67 = 30
$66 = 39
```

Nach der Rückkehr aus der Routine steht das Ergebnis in den Speicherstellen $0AA0 - $0AA3 in der Form:

```
$0AA0 = 00
$0AA1 = 01
$0AA2 = 23
$0AA3 = 45
```

also - hintereinander gelesen - 00012345, jeweils zwei Dezimalziffern in einem Byte.

unpack $BA5D Wert in beliebiger Basis ausgeben

Mit Hilfe dieser Routine können beliebige Werte auf dem aktuellen Ausgabekanal ausgegeben werden, die sich in den Speicherstellen $0AA0 bis $0AA3 befinden, wobei in $0AA0 das höchstwertige Byte und in $0AA3 das niedrigstwertige Byte stehen muß.

Der Wert kann in einem der drei Zahlensysteme binär, oktal oder dezimal ausgegeben werden, wobei bei dezimaler Ausgabe zuvor mit Hilfe der oben beschriebenen Routine "bindec" der Hexadezimal-Wert ins Dezimale gewandelt werden muß.

Als zusätzliche Parameter müssen angegeben werden:

```
Akku        = Flag Nullenunterdrückung
              0 = Führende Nullen unterdrücken
              1 = Führende Nullen nicht unterdrücken

X-Register = Anzahl der auszugebenden Ziffern


Y-Register = Anzahl Bits pro Ziffer minus 1, also
```

```
0 bei Binär,
2 bei Oktal,
3 bei Dezimal.
```

Die Werte werden auf dem gerade aktuellen Kanal ausgegeben, also im Normalfall auf dem Bildschirm.

Eine Routine zum Wandeln von beliebigen Zahlensystemen in das Hexadezimalsystem ist zwar im Monitor vorhanden, kann von externen Programmen aber nur unter größeren Umständen aufgerufen werden. Daher wurde hier auf eine Beschreibung dieser Routine namens "eval" verzichtet. Näheres ist dem Assemblerlisting des Monitors zu entnehmen.

1.2.3 Der Bildschirm-Editor

Um Programme oder Daten komfortabel eingeben zu können, enthält das Betriebsystem des C-128 einen Bildschirm-Editor, der - im Gegensatz zu vielen anderen Mikrocomputern - ein sogenannter "Fullscreen-Editor" ist. Dieser Typ eines Bildschirm-Editors erlaubt es dem Anwender, sich mit dem Cursor völlig frei auf dem Bildschirm zu bewegen, hier und dort Änderungen am Programm vorzunehmen, die beim Drücken der RETURN- bzw. ENTER-Taste sofort ausgeführt werden.

Das Gegenstück hierzu, der sogenannte "Line-Editor", hingegen erlaubt Cursorbewegungen und damit auch Änderungen nur innerhalb der aktuellen Bildschirmzeile. BASIC-Programmzeilen, die geändert werden sollen, müssen hier z.B. mit einer Art EDIT-Befehl extra aufgerufen werden, um dann einzeln bearbeitet werden zu können.

Es ist natürlich klar, daß ein Fullscreen-Editor wesentlich komplizierter zu programmieren ist, als ein simpler Line-Editor. Hier hat Commodore, übrigens schon von den Zeiten des legendären PET an, in Bezug auf den Komfort des Editors die Nase vorn.

Vom BASIC-Interpreter oder vom Betriebssystem her beschränken sich die Funktionen des Editors im wesentlichen auf zwei Funktionen: das Einlesen von Zeichen über die Tastatur und das Ausgeben von Zeichen auf dem Bildschirm. Hinzu kommen beim C-128 noch einige Funktionen, die z.B. Bildschirm-Fenster setzen und ähnliches.

Der Aufruf der Editor-Funktionen erfolgt über eine Sprungleiste. Unter einer Sprungleiste ist eine Reihe von JUMP-Befehlen zu verstehen, die ihrerseits zu den eigentlichen Arbeitsroutinen springen. Hier werden Sie sich fragen, wozu solch eine Sprungleiste nützlich ist. Auf den ersten Blick stellt sie doch lediglich eine gewisse Platzverschwendung und, gerade bei zeitkritischen Programmen, eine Verlängerung der Befehls-Ausführungszeit dar.

Der Grund ist eigentlich ganz einfach. Auch Betriebssysteme sind Programme, und für Programme gilt der schöne Satz: Ein Programm ist nie fehlerfrei; es sei denn, es ist veraltet.

Während der Programmierung eines doch recht umfangreichen Betriebssystems treten immer wieder Fehler zu Tage, die eine Änderung am Programm erfordern. Hiermit geht aber leider auch eine gewisse Verschiebung der Programm-Routinen einher, womit ebenfalls alle Programme zu ändern wären, die auf das Betriebssystem oder, in diesem Fall den Editor, zugreifen.

Bei Verwendung einer Sprungleiste ist man dieser Arbeiten enthoben, da die Adresse der Funktion innerhalb der Sprungleiste immer gleich bleibt. Nur die JUMP-Befehle zeigen jetzt unter Umständen auf andere Adressen.

An dieser Stelle erlauben Sie uns einen kleinen Hinweis: In diesem Buch finden Sie ein komplettes Assembler-Listing des Betriebsystems mit eingehenden Kommentaren. Es ist daher müßig, in diesem doch mehr oder weniger allgemeinen Teil alle Routinen noch einmal aufzuführen. Vielmehr sollen nur die wichtigsten Routinen kurz erläutert werden, um aufzuzeigen, wie der Assembler-Programmierer diese Routinen sinnvoll verwenden kann.

Weiterhin wollen wir ein wenig unsere "Trickkiste" öffnen und - gerade am noch recht unbekannten C-128 - ein paar Tricks und Kniffe zeigen. Hierzu gehört unserer Meinung nach aber auch dann und wann ein Hinweis auf das, was man einen guten Programmierstil nennt, z.B. auch die Verwendung der oben erwähnten Sprungleisten, gerade auch in Ihren eigenen Programmen. Umfangreiche Programme haben die Eigenart, nie ganz fertig zu sein und gerade die Verwendung solcher, doch eigentlich ganz simpler, Hilfsmittel kann eine Menge Arbeit ersparen. Vor allem dann, wenn man nicht so komfortable Hilfsmittel wie einen Assembler zur Verfügung hat, der ein Programm, von sagen wir acht Kbyte, vollständig neu assembliert; natürlich unter Umrechnung aller absoluten Sprünge.

Die erwähnte Sprungleiste umfaßt 16 Funktionen, die teilweise vom Anwender verwendet werden können und die auch das restliche Betriebssystem verwendet.

Zunächst einmal eine tabellarische Auflistung der möglichen Funktionen:

Name	Adresse	Funktion
jpcint	$C000	Video-Chip/Editor initialisieren
jdsply	$C003	Ausgabe AC auf Schirm, Attribut in XR
jlp2	$C006	Zeichen aus Tastatur-Puffer nach AC
jloop5	$C009	Zeichen vom Schirm nach AC
jprint	$C00C	Schirm-Ausgabe AC

```
jscorg   $C00F      Schirm-Organisation lesen
jkey     $C012      Tastatur-Abfrage
jrep     $C015      Code aus Tast.-Tab. holen, REPEAT-Test
jplot    $C018      Cursor-Position lesen/schreiben
jcurs    $C01B      Cursorposition im 80-Zeichen BS setzen
jesc     $C01E      Bearbeitung der ESC-Funktionen
jkyset   $C021      Funktionstaste belegen
jrsirq   $C024      Raster-Interrupt bearbeiten
jint80   $C027      80-Zeichen-Schirm initialisieren
jswap    $C02A      Umschalten 40/80-Zeichen-Schirm
jwind    $C02D      Schirm-Fenster definieren
```

Die einzelnen Routinen werden im folgenden jeweils kurz besprochen, da sie einige der wichtigsten Routinen für die Kommunikation mit dem Rechner von eigenen Programmen her enthalten.

jpcint ($C000) Video-Chip/Editor initialisieren

Diese Routine setzt nach dem Einschalten des Rechners bzw. einem Drükken des RESET-Tasters die Grundeinstellung der Video-Kontroller und der für das Editorprogramm wichtigen Speicheradressen.

jdsply ($C003) Ausgabe AC auf Schirm, Attribut in XR

Hiermit kann ein beliebiges Zeichen direkt auf den Bildschirm gebracht werden, wobei das Zeichen im Akku stehen und bereits im Bildschirm-Code vorliegen muß. Das X-Register muß das hierfür gültige Attributbyte enthalten, also die Farbinformation für den 40-Zeichen-Bildschirm und für den 80-Zeichen-Bildschirm zusätzlich die Attribute für Unterstreichen, Blinken usw.

Das Attributbyte hat folgenden Aufbau:

Bit	40-Zeichen-Schirm	80-Zeichen-Schirm
0 - 3	Farbinformation	Farbinformation
	0000 = Schwarz	0000 = Schwarz
	0001 = Weiß	0001 = Grau
	0010 = Rot	0010 = Blau
	0011 = Grün	0011 = Hellblau
	0100 = Violett	0100 = Dunkelgrün
	0101 = Dunkelgrün	0101 = Hellgrün
	0110 = Blau	0110 = Dunkelgrau
	0111 = Gelb	0111 = Grün
	1000 = Hellbraun	1000 = Rot
	1001 = Braun	1001 = Rosa
	1010 = Rosa	1010 = Hellbraun
	1011 = Dunkelgrau	1011 = Violett
	1100 = Grau	1100 = Braun

	1101 = Hellgrün 1110 = Hellblau 1111 = Hellgrau	1101 = Gelb 1110 = Hellgrau 1111 = Weiß
4	- - -	0 = Blinken aus 1 = Blinken an
5	- - -	0 = Unterstreichen aus 1 = Unterstreichen an
6	- - -	0 = Normale Darstellung 1 = Inverse Darstellung
7	- - -	0 = Grafik/Großbuchstaben 1 = Klein-/Großbuchstaben

Im 40-Zeichen-Modus werden die Farben, und damit das Attributbyte, wie beim C-64 dargestellt. Es werden also lediglich die untersten 4 Bits verwendet.

Im 80-Zeichen-Modus wird lediglich das Bit 6 ($40) nicht verwendet. Wie auch beim 40-Zeichen-Bildschirm werden die inversen Zeichen durch ein gesetztes oberstes Bit des Schirmcodes dargestellt ($00 bis $7F = normales Zeichen, $80 bis $FF = inverses Zeichen). Das Bit 7 (Grafikdarstellung) ist im Zusammenhang mit der Shift-Commodore-Taste zu sehen, mit der bekanntlich von Großbuchstaben/Grafikzeichen auf Klein-/Großschreibung umgeschaltet werden kann. Im Gegensatz zum 40-Zeichen-Modus bzw. zum C-64 ist es hier aber möglich, Zeichen aus beiden Zeichensätzen *gleichzeitig* auf dem Schirm zu sehen. Während im 40-Zeichen-Modus der Zeichensatz umgeschaltet wird, bestimmt hier lediglich das oberste Bit des Attributbyts, welcher Zeichensatz für das einzelne Zeichen herangezogen werden soll. Hierdurch sind insgesamt 512 verschiedene Schriftzeichen gleichzeitig auf dem Bildschirm darstellbar, und zwar 128 Zeichen, diese in inverser Darstellung (=256) und das Ganze noch in zwei Zeichensätzen (=512).

Der Zeichensatz des VDC befindet sich im eigenen 16-KByte-RAM-Bereich des VDC und kann mit eigenen Routinen verändert werden. Hierdurch ist es z.B. möglich, die dort enthaltenen inversen Zeichen durch eigene Zeichen zu ersetzen, und die Revers-Darstellung durch das Setzen des entsprechenden Attribut-Bits zu erreichen. Hierdurch läßt sich die Anzahl der darstellbaren Zeichen noch einmal verdoppeln, so daß insgesamt 1024 (!) verschiedene Zeichen gleichzeitig auf dem Schirm sichtbar sein können.

Die Routine "jdsply" schreibt das Zeichen an die aktuell gesetzte Schreibposition, die durch die Speicherstellen "pnt" ($E0/$E1) als Basis-Adresse und "pntr" ($EC) als Offset markiert wird. Die zugehörige Adresse im At-

tributspeicher wird durch die Routine selbstständig errechnet und befindet sich nach der Rückkehr in der Adresse "user" ($E2/$E3).

jlp2 **$C006** **Zeichen aus Tastatur-Puffer nach AC**

Während des Interrupts werden Tastendrücke zunächst nur registriert und in den dafür vorgesehenen Tastatur-Puffer von zehn Zeichen eingelesen. Mit der Routine "jlp2" können Sie nun aus eigenen Programmen Zeichen von der Tastatur holen, wobei die Zeichen dann aus dem Tastatur-Puffer geholt werden, sofern dort Zeichen vorhanden sind.

Nach dem Aufruf der Funktion enthält der Akkumulator das gelesene Zeichen oder, wenn kein Tastendruck vorliegt, eine Null. Der Inhalt des Tastatur-Puffers wird durch die Routine um dieses eine Byte vermindert. Die Speicherstelle "ndx" ($D0) gibt dabei an, wieviel Zeichen sich gerade im Tastatur-Puffer befinden.

Die Funktionstasten F1 bis F8 sowie auch die HELP-Taste und die Shift-RUN/STOP-Taste werden dabei etwas anders behandelt. Auf die Funktionstasten F1 bis F8 können mit dem KEY-Befehl Texte oder auch Befehle gelegt werden, die dann beim Drücken der entsprechenden Taste ausgeführt werden. Hierzu dient ein weiterer Zähler "kyndx" ($D1) der angibt, wieviel Zeichen auszugeben sind und das Indexfeld "keyidx" ($D2), das auf das aktuelle Zeichen aus der Key-Definition zeigt. Die Funktionstasten haben dabei Vorrang vor dem normalen Tastatur-Puffer. Wenn z.B. die Tasten "a", "F1" und "b" betätigt werden, werden bei mehrfachem Aufruf von "jlp2" zunächst die einzelnen Zeichen des Tastaturstrings der Funktionstaste "F1", sofern vorhanden, zugewiesen und anschließend daran das "a" und das "b" gelesen.

jloop5 **$C009** **Zeichen vom Schirm nach AC**

Die Routine "jloop5" liest ein Zeichen aus dem Bildschirmspeicher im ASCII-Format. Gelesen wird wieder aus der Adresse, die durch die aktuelle Bildschirmposition "pnt" ($E0/$E1) und den Offset "pntr" ($EC) angegeben wird. Nach der Rückkehr enthält der Akkumulator das gelesene Zeichen und die Variable "tcolor" ($F2) das dazugehörende Attributbyte.

jprint **$C00C** **Schirm-Ausgabe AC**

Mit Hilfe dieser Routine ist es möglich, ein Zeichen auf den Bildschirm zu bringen, wobei hier das Zeichen, im Gegensatz zur Routine "jdsply", im

ASCII-Format übergeben werden muß. Es können hier alle Zeichen, auch Steuerzeichen wie Cursorbewegungen, Farbumschaltungen usw. verwendet werden. Diese Routine wird z.B. auch vom BASIC innerhalb des PRINT-Befehls aufgerufen.

jscorg **$C00F** **Schirm-Organisation lesen**

Beim C-128 ist es, im Gegensatz zum C-64, wie bei den großen CBM-Rechnern möglich, ein Bildschirmfenster zu setzen. Das bedeutet, daß die normalerweise eingestellte Bildschirmgröße von 25 Zeilen zu 40 bzw. 80 Zeichen jederzeit vom Anwender geändert werden kann, und irgendwelche Schirmausgaben, zumindestens mit PRINT-Befehlen, sich nur noch auf dieses Schirmfenster beziehen. Mit der Routine "jscorg" können Sie sich jederzeit einen Überblick verschaffen, wie die momentane Aufteilung des Bildschirms ist. Nach dem Aufruf des Routine enthält das X-Register die momentane Anzahl zulässiger Spalten (Zeichen pro Zeile), das Y-Register die maximal zulässige Anzahl Zeilen und der Akku die Zeilenbreite insgesamt, also entweder 40 oder 80.

jkey **$C012** **Tastatur-Abfrage**

Diese Routine übernimmt die eigentliche Tastatur-Abfrage und das Bereitstellen der eingelesenen Zeichen im Tastatur-Puffer. Für den Anwender ist sie unerheblich, da diese Abfrage im Interrupt selbsttätig vorgenommen wird.

jrep **$C015** **Code aus Tastaturtabelle holen, REPEAT-Test**

Auch diese Routine ist für den Anwender uninteressant, da sie im Interrupt selbsttätig ausgeführt wird. Aufgrund der, in der Tastaturabfrage ermittelten, Tastennummer wird aus der zuständigen Tastaturtabelle der zugehörige ASCII-Wert ermittelt und geprüft, ob für diese Taste eine Wiederholung (Repeat) zulässig ist, sofern die Taste noch seit dem letzten Aufruf gedrückt ist.

jplot **$C018** **Cursorposition lesen/schreiben**

Hiermit ist es jederzeit möglich festzustellen, an welcher Position auf dem Bildschirm sich momentan der Cursor befindet, unabhängig davon, ob er gerade sichtbar ist oder nicht. Die Cursorposition ist im übrigen die Posi-

tion, an der das nächste mit PRINT bzw. der Routine "jprint" auszugebende Zeichen erscheint.

Um die Cursorposition zu lesen, muß zuvor das Carry-Flag mit dem Befehl "SEC" gesetzt werden. Nach dem Aufruf der Routine enthält das X-Register die aktuelle Zeilennummer der Cursorposition und das Y-Register die aktuelle Position innerhalb der Zeile. Beide Werte sind immer relativ zur linken oberen Ecke des aktuellen Bildschirmfensters zu sehen.

Zum Schreiben der Cursorposition muß vor dem Aufruf zunächst das Carry-Flag mit dem Befehl "CLC" gelöscht werden. Außerdem muß das X-Register die gewünschte Zeilennummer und das Y-Register die gewünschte Spalte enthalten; beide Werte wieder relativ zur linken oberen Ecke des Schirmfensters. Die erfolgreiche Ausführung des Befehls wird durch ein gelöschtes Carry-Flag angezeigt. Ist das Carry-Flag gesetzt, ist ein Fehler aufgetreten, also z.B. eine Zeilennummer verwendet worden, die nicht in die aktuelle Fenstergröße passt.

Die aktuelle Zeichenposition "pnt" ($E0/$E1), der Spaltenoffset "pntr" ($EC) sowie der Zeiger auf den Attributspeicher "user" ($E2/$E3) werden entsprechend neu errechnet.

jcurs $C01B Cursorposition im 80-Zeichen-BS setzen

Im 80-Zeichen-Bildschirm wird der Cursor hardwaremäßig vom VDC erzeugt, also nicht wie beim 40-Zeichen-Bildschirm durch einfaches invertieren des aktuellen Zeichens. Mit der Routine "jcurs" wird die aktuelle Cursorposition an den VDC übergeben und dort auf die entsprechende Stelle im Bildschirm-Speicher gesetzt. Die Adresse wird wieder durch die aktuelle Zeichenposition "pnt" ($E0/$E1) und den Spaltenoffset "pntr" ($EC) markiert, die selbstverständlich vorher auf die gewünschte Adresse zu setzen sind.

Das An- bzw. Abschalten des Cursors kann leider nicht über eine Routine der Editor-Sprungleiste erreicht werden. Die entsprechenden Routinen heißen "crsron" ($CD6F) zum Einschalten und "crsrof" ($CD9F) zum Abschalten des Cursors. Beide Routinen wirken sich immer auf den gerade aktuellen Schirm, also wahlweise den 40-Zeichen- oder 80-Zeichen-Bildschirm aus.

Der Cursormodus für den 40-Zeichen-Bildschirm wird durch die Adresse "blnon" ($0A26) festgelegt. Ist dort das Bit 6 ($40) gesetzt, wird ein fester Cursor erzeugt, ist das Bit nicht gesetzt, wird ein blinkender Cursor erzeugt.

Im 80-Zeichen-Modus sind noch ein paar Steuerungen mehr möglich. Hierzu dient die Adresse "curmod" ($0A2B), die bitweise die gewünschte Cursorform angibt. Die einzelnen Bits haben folgende Bedeutung, wobei nicht alle Möglichkeiten im C-128 genutzt, aber durchaus verwendet werden können:

Bit	Bedeutung
7	Nicht benutzt
6	Wenn das Bit "1" ist, blinkt der Cursor; ist das Bit "0", steht der Cursor fest
5	Dieses Bit arbeitet in Zusammenhang mit Bit 6: Ist Bit 6 = "0" und Bit 5 = "1", dann ist der Cursor abgeschaltet. Ist Bit 6 = "1", bestimmt dieses Bit die Blinkfrequenz: "0" = schnell "1" = langsam (Normalzustand)
4	Immer "0".
3	Sollte immer "0" sein; das Bit wird als höchstwertiges Bit zu den folgenden Bits 0 bis 2 betrachtet.
0-2	Diese Bits geben an, an welcher Punktposition innerhalb des Zeichens der Cursor beginnen soll.

Beim C-128 wird ein Zeichen aus acht Punktreihen zu jeweils acht Punkten, also insgesamt 64 Punkten, zusammengesetzt. Der Cursor wird durch invertieren der entsprechenden vertikalen Reihen dargestellt, wobei durch die Bits 0 bis 2 gewählt wird, an welcher Position er beginnen soll. Sind alle Bits "0" beginnt der Cursor an der ersten Position; es wird also ein Blockcursor dargestellt. Mit dem binären Wert 7 (alle drei Bits = "1") beginnt der Cursor in der letzten Punktposition; es wird also ein Unterstrich-Cursor dargestellt. Es sind hier natürlich auch Zwischenwerte möglich, die allerdings recht seltsam aussehen können (halber Cursor und ähnliches).

Das Bit 3 stellt das oberste Bit dieser Punktkombination dar. Der Grund liegt darin, daß der VDC durch andere Programmierung Zeichen auch mit 16 Punktzeilen darstellen könnte (z.B. beim CBM-700 verwendet). Die Auflösung ist dabei natürlich erheblich besser und die Zeichen sind in sich viel geschlossener und schärfer. Leider sind Farbmonitore erschwinglicher Preisklasse hiermit überfordert, daher wurde darauf verzichtet. Diese Mög-

lichkeit merkt man übrigens auch bei einer Analyse der im VDC-RAM abgelegten Zeichensätze. Im Zeichensatz-ROM belegen die beiden Zeichensätze zusammen nur 4 Kbyte; im VDC-RAM jedoch 8 KByte. Hinter jeder Zeichendefinition, die aus 8 Byte besteht, befindet sich hier jeweils eine weitere Gruppe von 8 Null-Bytes, um die 16 möglichen Punktreihen zu belegen.

Hier noch einmal die sinnvollen Cursorformen in tabellarischer Übersicht:

Wert	Cursorform
$00	Blockcursor, fest
$07	Unterstrich-Cursor, fest
$20	Cursor ausgeschaltet
$40	Blockcursor, schnell blinkend
$47	Unterstrich-Cursor, schnell blinkend
$60	Blockcursor, langsam blinkend
$67	Unterstrich-Cursor, langsam blinkend

jesc **$C01E** **Bearbeitung der ESC-Funktionen**

Der C-128 besitzt diverse Editor-Funktionen, die über eine ESCAPE-Sequenz aufgerufen werden. Hierzu wird zunächst ESC gedrückt und anschließend ein Buchstabe von "a" bis "z" oder das Paragraphen-Zeichen, bzw. der Klammeraffe bei der ASCII-Tastatur. Ein nochmaliges ESCAPE direkt hinter dem ersten schaltet die Funktion sofort wieder aus, falls ESCAPE versehentlich gedrückt wurde. Diese Funktionen können auch über PRINT-Befehle bzw. die oben erwähnte Routine "jprint" ausgeführt werden.

Die Routine "jesc" ruft die entsprechenden Editor-Funktionen direkt, ohne vorheriges ESCAPE, auf. Hierzu ist vor dem Aufruf im Akkumulator der gewünschte Befehlsbuchstabe zu setzen.

Die möglichen Befehle in tabellarischer Auflistung:

Akkuinhalt	Funktion
§	Bildschirm ab Cursorposition löschen
a	Autoinsert-Modus einschalten. Vor der Ausgabe eines Zeichens wird vorher mit "Insert" automatisch eine freie Stelle eingerichtet
b	Rechte untere Fensterecke an aktueller Cursorposition setzen

c	Autoinsert-Modus ausschalten
d	Aktuelle Zeile löschen. Der Rest des Bildschirm wird nach oben gerollt
e	Festen Cursor setzen
f	Blinkenden Cursor setzen
g	Glocke zulassen. Beim Betätigen von Control-g ertönt das Glockensignal, desgleichen bei der Ausgabe von CHR$(7) auf dem Bildschirm.
h	Glocke abschalten. Ein Aufruf der Glocke wird ignoriert
i	Zeile an der Cursorposition einfügen. Die folgenden Zeilen verschieben sich nach unten
j	Cursor an den Anfang der Zeile setzen
k	Cursor auf Zeilenende setzen
l	Bildschirm-Rollen einschalten. Wird bei PRINT-Befehlen der untere Rand erreicht, werden die Zeilen nach oben gerollt und eine Leerzeile am Schirmende eingefügt.
m	Bildschirm-Rollen ausschalten. Wird bei PRINT-Befehlen der untere Rand erreicht, wird die Ausgabe wieder am oberen Schirmrand fortgesetzt.
n	Negativ-Darstellung ausschalten (siehe "r")
o	Einfüge-, Anführungszeichen- und Reverse-Modus (z.B. nach Anführungszeichen) ausschalten
p	Zeile `bis` Cursorposition löschen
q	Zeile `ab` Cursorposition löschen
r	Negativ-Darstellung einschalten. Der gesamte Bildschirminhalt wird invertiert, allerdings nicht der Bildschirm-Rahmen.

s	Block-Cursor einschalten
t	Obere linke Fensterecke an aktueller Cursorposition setzen
u	Unterstrich-Cursor einschalten
v	Schirm um eine Zeile aufwärts rollen
w	Schirm um eine Zeile abwärts rollen
x	Umschalten von 40- auf 80-Zeichenmodus und zurück
y	Standard-Tabulatoren setzen
z	Alle Tabulatoren löschen

jkyset $C021 Funktionstaste belegen

Hiermit können die Funktionstasten von Assemblerprogrammen aus mit Werten belegt werden. Hierzu sind zunächst einige Vorbereitungen zu treffen. In der Zero-Page ist an drei aufeinander folgenden Adressen ein Zeiger auf den neuen, der Funktionstaste zuzuordnenden Text einzurichten. Anschließend ist in den Akkumulator die Adresse des ersten Bytes dieses Zeigers zu laden, in das X-Register die Nummer der Funktionstaste laut unten stehender Tabelle und in das Y-Register die Länge des Textes.

Die Funktionstaste "F1" soll z.B. mit dem Text "Testbelegung" definiert werden, der in der Bank 1 an der Adresse $3000 steht. Hierzu wird zunächst einmal ein Zeiger in der Zero-Page eingerichtet, sinnvollerweise an einer sonst nicht benötigten Stelle, z.B. den Adressen $FA - $FF.

Belegt wird also

```
$FA  mit $00, dem Lowbyte der Adresse $3000
$FB  mit $30, dem Highbyte der Adresse $3000
$FC  mit $01, der Banknummer
```

Anschließend werden die Prozessor-Register geladen, und zwar:

```
AC  mit $FA, der Zeigeradresse
XR  mit $01, der Nummer der Funktionstaste
YR  mit $0C, der Länge des Textes
```

Nun kann die Routine aufgerufen und die Funktionstaste belegt werden. Mögliche Funktionstasten sind die Tasten F1 bis F8, die HELP-Taste und

die Shift-RUN/STOP-Taste. Die anzugebende Nummer finden Sie in der folgenden Tabelle:

```
Nummer      Taste      Standardwert

  1           F1       GRAPHIC
  2           F2       DLOAD"
  3           F3       DIRECTORY     + CHR$(13)
  4           F4       SCNCLR        + CHR$(13)
  5           F5       DSAVE"
  6           F6       RUN           + CHR$(13)
  7           F7       LIST          + CHR$(13)
  8           F8       MONITOR       + CHR$(13)
  9   Sh.-RUN/STOP     DL"*" + CHR$(13) + RUN + CHR$(13)
 10         HELP       HELP  + CHR$(13)
```

Beachten Sie bitte beim Belegen der Funktionstasten, daß alle zehn Definitionen zusammen nicht mehr als 246 Zeichen umfassen dürfen.

jrsirq **$C024** **Raster-Interrupt bearbeiten**

Diese Routine ist für den Anwender nicht von Belang, da sie im Interrupt automatisch aufgerufen wird. Sie dient zur Umschaltung von Grafik- und Textbildschirm, wenn im BASIC ein Grafikmodus mit geteiltem Schirm eingestellt wurde. Beim Aufbau wird das Bild jeweils zeilenweise aufgebaut, wobei eine Zeile immer einer vertikalen Punktbreite entspricht. Der gesamte Bildschirm umfaßt theoretisch etwa 250 Zeilen inklusive des oberen und unteren Rahmens, wobei, je nach Monitor, einige Zeilen nicht sichtbar sind. Der VIC kann nun veranlaßt werden, bei Erreichen einer bestimmten Rasterzeile einen Interrupt auszulösen. Solche Interrupts werden von dieser Routine bearbeitet, die dann jeweils vom Grafikschirm auf den Textschirm umschalten und somit einen geteilten Bildschirm simulieren.

jint80 **$C027** **80-Zeichen-Schirm initialisieren**

Mit dieser Routine wird der Zeichengenerator in das RAM des VDC kopiert. Wie schon erwähnt, wird beim Betätigen der ASCII/DIN-Taste der jeweils aktuelle Zeichensatz übertragen. Diese Funktion übernimmt die vorliegende Routine, die den Zeichengenerator von der Adresse $D000 in Bank 14 in das RAM des VDC kopiert.

jswap **$C02A** **Umschalten 40/80-Zeichen-Schirm**

Diese Funktion schaltet auf den jeweils anderen Bildschirm, also von 40-Zeichen auf 80-Zeichen oder umgekehrt, um. Sie entspricht vollständig der Funktion "ESC x" und wird von dieser auch aufgerufen.

jwind **$C02D** **Bildschirmfenster definieren**

Mit dieser Funktion können Bildschirmfenster gesetzt werden, wie auch schon mit den Funktionen "ESC b" bzw. "ESC t". Vor dem Aufruf ist im Akkumulator die gewünschte Zeilennummer absolut, d.h. also von 0 bis 24 und im X-Register die gewünschte Spalte, ebenfalls absolut, also von 0 bis 39 bzw. 0 bis 79, anzugeben. Bei gesetztem Carry-Flag wird die rechte untere Ecke des Fensters gesetzt, bei gelöschtem Carry-Flag die linke obere Ecke.

1.2.4 Das Kernal

Das Kernal umfaßt alle Funktionen, die zum ordnungsgemäßen Betrieb des Rechners notwendig sind, und ruft seinerseits auch den Editor auf. Im Kernal sind alle Routinen enthalten, um den Datenverkehr auf dem seriellen Bus zur Floppy oder einem Drucker zu regeln, sowie die gesamte Bedienung der RS232-Schnittstelle und des Kassettenrekorders. Weiterhin übernimmt es die Bearbeitung von Interrupts, also IRQ und NMI sowie die erstmalige Einstellung aller Systemparameter nach dem Einschalten oder dem Drücken des Reset-Tasters.

Auch das Kernal verfügt über eine Sprungleiste, die hier allerdings am Ende des Programms liegt und über die alle notwendigen Funktionen aufgerufen werden können. Der grundlegende Teil der Sprungleiste ist übrigens mit der des C-64 identisch; sie ist nur um einige Funktionen erweitert worden, die beim C-128 hinzugekommen sind. Diese Sprungleiste ist bei allen CBM-Rechnern zu finden, zumindestens allen, die mit den Prozessoren der Serie "65xx" arbeiten.

Wenn Programme entwickelt werden, empfiehlt es sich, benötigte Routinen immer über diese Sprungleiste und nicht direkt anzusprechen. Bei einer Umstellung des Programms auf einen anderen CBM-Rechner ist man dann von der Neuanpassung aller verwendeten Systemroutinen entbunden, da alle Einsprungadressen der Sprungleiste auf allen Rechnern an der gleichen Stelle liegen, vom legendären PET-2001, über den CBM-8032 und CBM-700, bis hin zum C-64 und dem C-128. Alle diese Rechner haben die gleichen Grundfunktionen an gleicher Stelle ansprechbar gespeichert, so daß

die notwendige Anpassungsarbeit auf eventuell verwendete Speicheradressen und eventuell andere Adressen für z.B. Bildschirmspeicher reduziert wird.

Wie auch schon beim Editor, hier zunächst einmal eine tabellarische Auflistung der einzelnen Funktionen und anschließend eine eingehende Besprechung der wichtigsten Routinen. Alle Routinen unterhalb der punktierten Linie sind übrigens identisch mit denen des C-64.

Name	Adresse	Funktion
kspio	$FF47	Auf I/O schnelle Floppy umschalten
kclsal	$FF4A	Alle Files von Gerät in AC schließen
kc64md	$FF4D	Auf C64-Modus umschalten
kdmacl	$FF50	DMA-Anforderung ext. RAM zulassen
kbotcl	$FF53	Boot-Programm von Diskette laden
kphenx	$FF56	Aufruf Kaltstartroutinen Funktionskarte
klkupl	$FF59	Geräte- & Sek.-Adr. über la (ac) suchen
klkups	$FF5C	Geräte-Adr. über Sek.Adr. (yr) suchen
kswapr	$FF5F	Umschalten 40/80-Zeichen-BS
kdlchr	$FF62	80-Zeichen-BS initialisieren
kpfkey	$FF65	Funktionstaste belegen
kstbnk	$FF68	Bank für load/sav/verify setzen
kgtcfg	$FF6B	MMU-Wert zu Bank in xr nach ac
kjsrfr	$FF6E	JSR, Bank xr
kjmpfr	$FF71	JMP, Bank xr
kifetv	$FF74	LDA (FETVEC),y von Bank in xr
kistav	$FF77	STA (STAVEC),y in Bank in xr
kicmpv	$FF7A	CMP (CMPVEC),y mit Bank in xr
kprimm	$FF7D	Folgenden Text ausgeben und überspringen
.....		
kpcint	$FF81	Video-Chips/Editor initialisieren
kioini	$FF84	I/O-Bausteine initialisieren
kramts	$FF87	RAM, Kass.-Puffer u. Vectoren einrichten
kresto	$FF8A	Standard I/O-Vectoren einrichten
kvecto	$FF8D	User I/O-Vectoren einrichten
kstmsg	$FF90	Flag setzen Unterdrückung Meldungen
ksecnd	$FF93	Ausgabe Sek.Adr. nach LISTEN auf IEC-Bus
ktksa	$FF96	Ausgabe Sek.Adr. nach TALK auf IEC-Bus
kmemtp	$FF99	höchste Speichergrenze lesen/schreiben
kmembt	$FF9C	unterste Speichergrenze lesen/schreiben
kscnky	$FF9F	Tastatur-Abfrage
ksetmo	$FFA2	Timeout-Flag IEC-Bus setzen
kacptr	$FFA5	Zeichen von IEC-Bus nach AC
kciout	$FFA8	Ausgabe ac auf IEC-Bus
kuntlk	$FFAB	Ausgabe UNTALK auf IEC-Bus
kunlsn	$FFAE	Ausgabe UNLISTEN auf IEC-Bus
klistn	$FFB1	Ausgabe LISTEN auf IEC-Bus
ktalk	$FFB4	Ausgabe TALK auf IEC-Bus
kreads	$FFB7	Aktueller Status nach AC
kstlfs	$FFBA	la, fa, sa aus ac, xr, yr setzen
kstnam	$FFBD	fnlen, fnadr aus ac, xr, yr setzen
kopen	$FFC0	Logisches File öffnen
kclose	$FFC3	Logisches File schliessen

```
kchkin    $FFC6   Eingabe-Kanal öffnen
kckout    $FFC9   Ausgabe-Kanal öffnen
kclrch    $FFCC   Ein-/Ausgabe-Kanal schließen
kbasin    $FFCF   Zeichen akt. Kanal nach AC
kbsout    $FFD2   Ausgabe AC auf akt. Kanal
kloads    $FFD5   Load/verify (AC) ab Adresse in xr/yr
ksaves    $FFD8   Save ab ac bis xr, yr
ksttim    $FFDB   System-Uhr setzen
krdtim    $FFDE   System-Uhr nach ac, xr, yr
kstop     $FFE1   Stop-Taste lesen
kgetin    $FFE4   Zeichen akt. Kanal nach AC
kclall    $FFE7   Filetabelle löschen, Vorgabe-I/O
kclock    $FFEA   System-Uhr aktualisieren
kscror    $FFED   BS-Organisation lesen
kplot     $FFF0   Cursor-Pos. lesen/schreiben
kiobas    $FFF3   Basisadresse I/O-Bausteine nach xr/yr
```

kspio **$FF47** **Auf I/O schnelle Floppy umschalten**

Mit dieser Funktion wird auf die schnelle Floppy-Routine umgeschaltet, die allerdings nur mit den Floppy-Laufwerken 1570 und 1571 verwendet werden kann. Das Carry-Flag bestimmt, ob die Routine eingeschaltet (Carry gelöscht) oder ausgeschaltet (Carry gesetzt) werden soll.

kclsal **$FF4A** **Alle Files von Gerät in AC schließen**

Mit dieser Funktion können alle Dateien gleichzeitig geschlossen werden, die auf der im Akku angegebenen Geräteadresse geöffnet sind. Ein gesetztes Carry-Flag nach der Rückkehr zeigt einen Fehler innerhalb der Routine an.

kc64md **$FF4D** **Auf C-64-Modus umschalten**

Diese Routine schaltet auf den C-64-Modus um, wobei ein Rücksprung von dort nur mit der Reset-Taste möglich ist. Diese Routine wird auch vom BASIC-Befehl "GO64" aufgerufen.

kdmacl **$FF50** **DMA-Anforderung externen RAMs zulassen**

Diese Routine dient zum DMA (Direct Memory Access = direkter Speicherzugriff) durch externe Geräte. Im Normalbetrieb des C-128 ist sie nicht von Belang.

kbotcl **$FF53** **Boot-Programm von Diskette laden**

Mit dem BASIC-Befehl "BOOT" kann z.B. das CP/M-Betriebssystem geladen werden, in dem der Block 0 in der Spur 1 der Diskette gelesen wird, wo weitere Anweisungen wie Programmname, Startadresse usw. zu finden sind. Diese Routine ruft die entsprechende Funktion auf, wobei im Akkumulator die Laufwerknummer 0 oder 1 im ASCII-Code, also $30 oder $31 und im X-Register die Gerätenummer stehen müssen. Wenn der gelesene Sektor ein Bootsektor ist, wird das Programm ausgeführt, ansonsten wird der Aufruf ignoriert.

kphenx **$FF56** **Aufruf Kaltstartroutinen Funktionskarte**

Diese Routine dient zum Aufruf der zusätzlichen Funktionskarte, die im normalen C-128 nicht enthalten ist. Wird die Routine hier aufgerufen, wird ein normaler "Boot" von Gerät 8, Laufwerk 0 durchgeführt.

klkupl **$FF59** **Geräte- u. Sekundäradr. über la (ac) suchen**

Zu einer vorgegebenen logischen Dateinummer im Akkumulator werden die zugehörige Gerätenummer und die Sekundäradresse in der Dateitabelle gesucht. Das Carry-Flag zeigt an, ob ein entsprechender Eintrag in der Dateitabelle gefunden wurde. Ist das Carry-Flag gesetzt, ist diese Datei nicht geöffnet. Ist das Carry-Flag gelöscht, enthält das X-Register die Geräteadresse, das Y-Register die Sekundäradresse und der Akkumulator die logische Dateinummer.

klkups **$FF5C** **Geräteadresse über Sekundäradr. (yr) suchen**

Zu einer vorgegebenen Sekundäradresse im Y-Register werden die zugehörige logische Dateinummer und die Sekundäradresse gelesen. Das Carry-Flag zeigt an, ob ein entsprechender Eintrag in der Dateitabelle gefunden wurde. Ist das Carry-Flag gesetzt, ist diese Datei nicht geöffnet. Ist das Carry-Flag gelöscht, enthält das X-Register die Geräteadresse, das Y-Register die Sekundäradresse und der Akkumulator die logische Dateinummer.

kswapr **$FF5F** **Umschalten 40/80-Zeichen-Bildschirm**

Diese Routine ruft die unter den Editor-Funktionen beschriebene Routine "jswap" auf.

kdlchr **$FF62** **80-Zeichen-Bildschirm initialisieren**

Diese Routine ruft die unter den Editor-Funktionen beschriebene Routine "jint80" auf.

kpfkey **$FF65** **Funktionstaste belegen**

Diese Routine ruft die unter den Editor-Funktionen beschriebene Routine "jkyset" auf.

kstbnk **$FF68** **Bank für load/sav/verify setzen**

Beim Eröffnen vom Diskettendateien müssen in der Regel Dateinamen angegeben werden, deren Adressen in der Speicherstelle "fnadr" ($BB/$BC) zu finden sind. Beim C-128 muß darüber hinaus noch die Bank angegeben werden, auf die sich diese Adresse bezieht. Die Bank wird mit dieser Funktion gesetzt, wobei das X-Register die entsprechende Bitkombination für das MMU-Register und der Akkumulator die Banknummer enthalten muß.

kgtcfg **$FF6B** **MMU-Wert zu Bank in XR nach AC**

Zu jeder RAM- oder ROM-Bank des C-128 gehört eine entsprechende Bitkombination, die in das Konfigurations-Register der MMU zu schreiben ist, um die gewünschte Speicherverteilung zu erhalten. Hierzu existiert im Kernal-ROM eine entsprechende Tabelle. Diese Routine sucht zu einer gegebenen Banknummer im X-Register die entsprechende Kombination aus der Tabelle und stellt sie im Akkumulator zur Verfügung.

kjsrfr **$FF6E** **JSR, Bank xr**

Diese Routine ruft ein Unterprogramm auf, das sich in einer beliebigen Speicherbank befinden kann. Hierzu muß das X-Register die Banknummer enthalten und die Speicherstellen "areg" und folgende die entsprechenden gewünschten Registerinhalte beim Eintritt in das Unterprogramm. Die Logik entspricht hier einem JSR-Befehl.

kjmpfr **$FF71** **JMP, Bank xr**

Diese Routine ruft ein Programm auf, das sich in einer beliebigen Speicherbank befinden kann. Hierzu muß das X-Register die Banknummer ent-

halten und die Speicherstellen "areg" und folgende die entsprechenden gewünschten Registerinhalte beim Eintritt in das Programm. Die Logik entspricht hier einem JMP-Befehl.

kifetv **$FF74** **LDA (FETVEC),y von Bank in xr**

Mit Hilfe dieser Routine kann ein Byte aus einer anderen Speicherbank als der, in der das aktuelle Programm gerade läuft, gelesen werden. Hierzu muß das X-Register die Banknummer und der Akkumulator die Adresse eines Adressenpaars aus der Zeropage enthalten, in der die zu lesende Adresse steht. Die Funktion entspricht somit in etwa einem "LDA (Adresse),y"-Befehl.

kistav **$FF77** **STA (STAVEC),y in Bank in xr**

Mit Hilfe dieser Routine kann ein Byte in eine andere Speicherbank als die, in der das aktuelle Programm gerade läuft, geschrieben werden. Hierzu muß das X-Register die Banknummer und die Speicherstelle "stashx" ($02B9) die Anfangs-Adresse eines Registerpaars aus der Zero-Page enthalten, in der die zu schreibende Adresse steht. Die Funktion entspricht somit in etwa einem "STA (Adresse),y"-Befehl.

kicmpv **$FF7A** **CMP (CMPVEC),y mit Bank in xr**

Mit Hilfe dieser Routine kann ein Byte im Akkumulator mit einem Byte aus einer anderen Speicherbank als der, in der das aktuelle Programm gerade läuft, verglichen werden. Hierzu muß das X-Register die Banknummer enthalten und die Speicherstelle "cmparx" ($02CD) die Anfangs-Adresse eines Registerpaars aus der Zero-Page, in der die zu vergleichende Adresse steht. Die Funktion entspricht somit in etwa einem "CMP (Adresse),y"-Befehl.

kprimm **$FF7D** **Folgenden Text ausgeben und überspringen**

Mit Hilfe dieser Routine kann ein beliebiger Text, der direkt auf den JSR-Befehl folgt, auf dem Bildschirm ausgegeben werden. Der Text kann beliebig lang sein und muß mit "$00" beendet werden. Anschließend wird die Verarbeitung mit dem auf den Text folgenden Befehl fortgesetzt. Diese Routine ist im übrigen ein Musterbeispiel für trickreiche Programmierung und ist somit durchaus eine genaue Analyse Wert.

kpcint **$FF81** **Video-Chips/Editor initialisieren**

Diese Routine ruft die unter den Editor-Funktionen besprochene Routine "jpcint" auf.

kioini **$FF84** **I/O-Bausteine initialisieren**

Mit dem Aufruf dieser Routine werden alle Ein-/Ausgabe-Bausteine auf ihre Standardwerte nach dem Einschalten des Rechners gesetzt. Wird ein Anwender-Programm von anderen Programmen aufgerufen und herrscht Unklarheit über die aktuelle Einstellung der Peripherie-Bausteine, so kann hiermit ein definierter Zustand hergestellt werden.

kramts **$FF87** **RAM, Kassettenpuffer und Vektoren einrichten**

Mit dieser Routine werden die Zeiger auf die Ein-/Ausgabe-Puffer wie Kassettenpuffer und RS232-Puffer neu eingerichtet und die Speichergrenzen auf die Standardwerte festgelegt.

kresto **$FF8A** **Standard I/O-Vektoren einrichten**

Diese Routine stellt die normalen I/O-Vektoren, z.B. für IRQ, Öffnen einer Datei usw. wieder her. Sie kann aufgerufen werden, wenn diese Vektoren ganz oder teilweise verändert wurden, um den Grundzustand wiederherzustellen.

kvecto **$FF8D** **User I/O-Vektoren einrichten**

Hiermit können die I/O-Vektoren auf eigene Adressen gesetzt werden. Hierzu muß im X-Register das Lowbyte und im Y-Register das Highbyte der Vektorentabelle gesetzt werden. Die Vektorentabelle selbst muß alle 16 möglichen Vektoren enthalten, um Fehler zu vermeiden. Sie baut sich aus jeweils einem Wort (2 Bytes) in der Reihenfolge Lowbyte, Highbyte auf, die auf die Adressen der aufzurufenden Routinen zeigen. Die Original-Vektortabelle "vectss" ($E073) kann hierfür teilweise kopiert werden.

kstmsg **$FF90** **Flag setzen Unterdrückung Meldungen**

Bei der Abarbeitung von System-Routinen können Fehler auftreten, wie z.B. ein nicht angegebener Dateiname und ähnliches. Normalerweise würde

dann auf dem Bildschirm eine entsprechende Fehlermeldung erscheinen. Diese Routine setzt ein Flag, mit dem die Ausgabe dieser Meldungen verhindert (Akku = $40) oder ermöglicht (Akku = $00) werden kann. Zusätzlich kann durch ein gesetztes Bit 7 ($80 bzw. $C0) verhindert werden, daß bei Operationen, wie z.B. dem Laden von Programmteilen, der Text "searching for ..." oder auch "press play ..." auf dem Bildschirm erscheint.

ksecnd **$FF93** **Ausgabe Sekundäradresse nach LISTEN auf IEC-Bus**

Hiermit wird die gewünschte Sekundäradresse nach einem Listen-Befehl auf dem IEC-Bus ausgegeben, um dem Peripherie-Gerät mitzuteilen, über welche Sekundäradresse die folgenden Daten abzulegen sind. Die gewünschte Sekundäradresse muß sich beim Aufruf im Akkumulator befinden.

Hierzu kurz noch eine Anmerkung: Die gesamte Steuerung des IEC-Bus ist recht komplex und kann schnell durcheinander kommen, wenn die korrekte Reihenfolge der Listen-, Unlisten-, Talk- und sonstigen Befehle nicht eingehalten wird. Noch nicht so erfahrene Programmierer sollten daher auf die eigene Anwendung der IEC-Routinen verzichten, und besser mit den weiter unten beschriebenen Funktionen "kchkin", "kckout", "kgetin" und "kbsout" arbeiten. Bei der Verwendung dieser Routinen übernimmt das Betriebssystem die gesamte Steuerung des IEC-Bus; allerdings sind diese Routinen insgesamt etwas langsamer als die direkte Steuerung.

ktksa **$FF96** **Ausgabe Sekundäradresse nach TALK auf IEC-Bus**

Hiermit wird die gewünschte Sekundäradresse nach einem Talk-Befehl auf dem IEC-Bus ausgegeben, um dem Peripherie-Gerät mitzuteilen, über welche Sekundäradresse die folgenden Daten zu senden sind. Die gewünschte Sekundäradresse muß sich beim Aufruf im Akkumulator befinden. Ansonsten gilt auch hier der unter "ksecnd" gegebene Hinweis.

kmemtp **$FF99** **Höchste Speichergrenze lesen/schreiben**

Diese Routine liest oder schreibt wahlweise die höchste Speicheradresse, die vom Benutzerprogramm (und hier ist auch der BASIC-Interpreter ein Benutzer!) verwendet werden kann. Über das Carry-Flag wird gesteuert, ob die Adresse geschrieben (Carry &icht gesetzt) oder gelesen (Carry gesetzt) werden soll. Beim Setzen muß im X-Register das Lowbyte und im Y-Regi-

ster das Highbyte der Adresse stehen; beim Lesen enthalten die angegebenen Register die momentan gültigen Werte.

kmembt **$FF9C** **Unterste Speichergrenze lesen/schreiben**

Diese Routine liest oder schreibt wahlweise die unterste Speicheradresse, die vom Benutzerprogramm verwendet werden kann. Über das Carry-Flag wird gesteuert, ob die Adresse geschrieben (Carry nicht gesetzt) oder gelesen (Carry gesetzt) werden soll. Beim Setzen muß im X-Register das Lowbyte und im Y-Register das Highbyte der Adresse stehen; beim Lesen enthalten die angegebenen Register die momentan gültigen Werte.

kscnky **$FF9F** **Tastatur-Abfrage**

Hiermit wird die unter den Editor-Funktionen besprochene Routine "jkey" aufgerufen.

ksetmo **$FFA2** **Timeout-Flag IEC-Bus setzen**

Diese Funktion wird vom System nicht verwendet. Die entsprechende Speicherstelle für das Verhindern des IEC-Timeout, wie z.B. bei den großen CBM-Rechnern, ist zwar vorhanden, wird aber nicht verwendet, wie auch im Assemblerlistung zu erkennen ist. Somit dürfte die Routine nur als Platzhalter anzusehen sein, um ein Verschieben der restlichen Adressen zu verhindern.

kacptr **$FFA5** **Zeichen von IEC-Bus nach AC**

Diese Routine liest ein Zeichen direkt vom IEC-Bus in den Akkumulator. Zuvor müssen die entsprechenden Talk-Routinen aufgerufen werden. Hierfür gilt ansonsten auch der unter "ksecnd" geschriebene Hinweis.

kciout **$FFA8** **Ausgabe ac auf IEC-Bus**

Das Zeichen im Akkumulator wird direkt auf dem IEC-Bus ausgegeben. Zuvor müssen die entsprechenden Listen-Routinen aufgerufen werden. Hierfür gilt ansonsten auch der unter "ksecnd" gegebene Hinweis.

kuntlk **$FFAB** **Ausgabe UNTALK auf IEC-Bus**

Auf dem IEC-Bus wird ein "Untalk" ausgegeben, also die Aufforderung, das Senden von Daten einzustellen. Hierfür gilt ansonsten auch der unter "ksecnd" gegebene Hinweis.

kunlsn **$FFAE** **Ausgabe UNLISTEN auf IEC-Bus**

Auf dem IEC-Bus wird ein "Unlisten" ausgegeben, also die Kennzeichnung, daß keine weiteren Daten folgen. Hierfür gilt ansonsten auch der unter "ksecnd" gegebene Hinweis.

klistn **$FFB1** **Ausgabe LISTEN auf IEC-Bus**

Hiermit wird auf dem IEC-Bus ein "Listen" ausgegeben, also die Kennzeichnung, daß jetzt Datenbytes folgen. Hierfür gilt ansonsten auch der unter "ksecnd" gegebene Hinweis.

ktalk **$FFB4** **Ausgabe TALK auf IEC-Bus**

Hiermit wird auf dem IEC-Bus ein "Talk" ausgegeben, also die Kennzeichnung, daß jetzt Daten zu senden sind. Hierfür gilt ansonsten auch der unter "ksecnd" gegebene Hinweis.

kreads **$FFB7** **Aktueller Status (RS232 bzw. Syst.) nach AC**

Hiermit kann der aktuelle Status nach einer Ein-/Ausgabe-Operation abgefragt werden. Der Status enthält eine Null, wenn die Operation fehlerfrei abgeschlossen wurde, ansonsten wird, bitweise verschlüsselt, ein Hinweis auf die Art des Fehlers gegeben. Die einzelnen Bits haben, sofern sie auf "1" gesetzt sind, folgende Bedeutung:

Bit	Bedeutung
0	Timeout (Zeitüberschreitung) beim Schreiben
1	Timeout (Zeitüberschreitung) beim Lesen
2	Zu kurzer Datenblock (Nur Kassette)
3	Zu langer Datenblock (Nur Kasette)
4	Schwerer Fehler, nicht weiter spezifiziert
5	Prüfsummenfehler
6	Dateiende, kein Fehler, nur ein Hinweis
7	Angesprochenes Gerät nicht angeschlossen oder Bandende (nur Kassette)

kstlfs **$FFBA** **la, fa, sa aus ac, xr, yr setzen**

Die aktuellen Parameter für eine zu öffnende Datei können hiermit gesetzt werden. Der Akkumulator muß die logische Dateinummer, das X-Register die Gerätenummer und das Y-Register die Sekundäradresse enthalten. Zusätzlich ist im Anschluß an diese Funktion unter Umständen noch der Dateiname mit der im folgenden beschriebenen Funktion "setnam" anzugeben.

kstnam **$FFBD** **fnlen, fnadr aus ac, xr, yr setzen**

Hiermit können die Parameter für einen Dateinamen beim Eröffnen einer Datei gesetzt werden. Der Akkumulator muß die Länge des Dateinamens und, sofern die Länge nicht null - also kein Dateiname vorhanden - ist, das X-Register das Lowbyte der Dateinamenadresse und das Y-Register das Highbyte enthalten. Zusätzlich ist noch mit der Routine "setbnk" (siehe dort) die Banknummer zu definieren, in der der Dateiname steht. Falls nicht schon geschehen, werden außerdem mit der oben beschriebenen Routine "kstlfs" die Parameter für Dateinummer, Geräte- und Sekundäradresse definiert.

kopen **$FFC0** **Logisches File öffnen**

Eine zuvor mit den oben beschriebenen Parametern versehene Datei kann mit dieser Routine geöffnet werden. Nach der Operation zeigt das Carry-Flag an, ob die Funktion erfolgreich (Carry nicht gesetzt) oder fehlerhaft (Carry gesetzt) war.

kclose **$FFC3** **Logisches File schliessen**

Eine zuvor geöffnete Datei kann mit dieser Funktion wieder geschlossen werden. Der Akkumulator muß hierzu die logische Dateinummer enthalten. Nach der Operation zeigt das Carry-Flag an, ob das Eröffnen erfolgreich (Carry nicht gesetzt) oder fehlerhaft (Carry gesetzt) war.

kchkin **$FFC6** **Eingabe-Kanal öffnen**

Diese Routine dient zum Einlesen von Daten von beliebigen Geräten. Zuvor ist eine entsprechende Datei zu eröffnen, wobei hier jedes Gerät verwendet werden kann, das als Eingabegerät zugelassen ist, also IEC-Bus (Gerätenummern 4-15), Kassette (#1), RS232-Schnittstelle (#2) und Bildschirm (#0).

Beim Aufruf der Routine muß das X-Register die logische Dateinummer enthalten. Alle folgenden Aufrufe z.B. der Routine "kgetin" (siehe dort) lesen jeweils ein Zeichen vom angesprochenen Gerät, bis mit der Routine "kclrch" die Eingabe wieder auf die Tastatur zurückgestellt wird.

kckout **$FFC9** **Ausgabe-Kanal öffnen**

Diese Routine dient zum Ausgeben von Daten auf beliebige Geräte. Zuvor ist eine entsprechende Datei zu öffnen. Dabei kann jedes Gerät verwendet werden, das als Ausgabegerät zugelassen ist, also IEC-Bus (Gerätenummern 4-15), Kassette (#1), RS232-Schnittstelle (#2) und Bildschirm (#3).

Beim Aufruf der Routine muß das X-Register die logische Dateinummer enthalten. Alle folgenden Aufrufe z.B. der Routine "kbsout" (siehe dort) schreiben jeweils ein Zeichen auf das angesprochene Gerät, bis mit der Routine "kclrch" die Ausgabe wieder auf den Bildschirm zurückgestellt wird.

kclrch **$FFCC** **Ein-/Ausgabe-Kanal schließen**

Diese Routine stellt nach erfolgter Datenein- oder ausgabe wieder auf den Bildschirm als Ausgabe- und die Tastatur als Eingabegerät zurück.

kbasin **$FFCF** **Zeichen vom aktiven Kanal nach AC**

Hiermit wird ein Zeichen vom aktiven Eingabekanal gelesen. Normalerweise wird ein Zeichen aus der aktuellen Bildschirmposition gelesen, es sei denn, mit "kchkin" ist zuvor eine Datei als Eingabekanal aktiviert worden.

kbsout **$FFD2** **Ausgabe AC auf aktiven Kanal**

Hiermit wird ein Zeichen auf dem aktiven Ausgabekanal ausgegeben. Normalerweise wird ein Zeichen an der aktuellen Bildschirmposition ausgegeben, es sei denn, mit "kckout" ist zuvor eine Datei als Ausgabekanal aktiviert worden.

kloads **$FFD5** **Load/verify (AC) ab Adresse in xr/yr**

Hiermit kann ein Programm geladen (LOAD) bzw. mit einem im Speicher stehenden Programm verglichen werden (VERIFY). Hierzu sind zunächst die Parameter für den Dateinamen zu setzen, und dann die Routine aufzu-

rufen. Im Akkumulator muß angegeben werden, ob ein "Load" (Akku = $00) oder ein "Verify" (Akku <> $00) durchgeführt werden soll. Der Lade- oder Vergleichsvorgang beginnt ab der in der Datei angegebenen Startadresse, sofern die Sekundäradresse null ist. Ist die Sekundäradresse ungleich null, so ist wird die Datei an die Adresse geladen, die im X-Register (Lowbyte) und im Y-Register (Highbyte) angegeben ist.

ksaves $FFD8 Save ab ac bis xr, yr

Mit dieser Routine kann ein im Speicher stehendes Programm in einer Datei gesichert werden. Hierzu sind zunächst die Parameter für den Dateinamen zu setzen, und dann die Routine aufzurufen. Im Akkumulator muß die Adresse eines Zero-Page-Registerpaares angegeben werden, das die Startadresse im Speicher enthält. Die Endeadresse wird beim Aufruf mitgeteilt, wobei das X-Register das Lowbyte und das Y-Register das Highbyte enthalten muß.

ksttim $FFDB Systemuhr setzen

Das Betriebssystem des C-128 stellt eine Systemuhr zur Verfügung, die jeweils im Interrupt fortgeschrieben wird. Mit dieser Routine kann die Uhr auf einen neuen Wert gesetzt werden, wobei die Uhrzeit in 1/60-Sekunden anzugeben ist. Eine Sekunde entspricht somit dem Wert 60, eine Minute dem Wert 3600 und eine Stunde dem Wert 216000. Die Uhrzeit ist aus diesen Werten zu errechnen, wobei z.B. 10 Uhr, 5 Minuten und 20 Sekunden dem Wert 2179200 entspricht. Diese Zahl ist ins binäre Datenformat umzurechnen, und in drei Bytes der Routine zu übergeben, wobei der Akkumulator das höchstwertige Byte, das X-Register das mittlere und das Y-Register das niedrigstwertige Byte enthält. In unserem Beispiel enthalten die Register folgende Werte:

Akku = 33, XR = 64 und YR = 128,

denn (33 * 65536) + (64 * 256) + 128 = 2179200.

krdtim $FFDE Systemuhr nach ac, xr, yr

Die vom C-128 verwaltete Systemuhr kann mit dieser Routine gelesen werden, wobei die Uhrzeit als Anzahl der seit Null Uhr vergangenen 1/60-Sekunden zu interpretieren ist. Eine Beschreibung finden Sie beim vorhergehenden Befehl. Die Register enthalten die Werte in der dort angegebenen Reihenfolge.

kstop **$FFE1** **Stop-Taste lesen**

Durch diese Routine kann festgestellt werden, ob die Stop-Taste gedrückt wurde. Nach dem Aufruf der Funktion ist das Zero-Flag entsprechend gesetzt, d.h. ein BEQ-Befehl direkt nach der Routine verzweigt, wenn die Stop-Taste gedrückt wurde. Zu beachten ist, daß die Routine auch gleichzeitig - natürlich nur bei gedrückter Stop-Taste - die Standard Ein-/Ausgabe-Kanäle Tastatur und Bildschirm wiederherstellt.

kgetin **$FFE4** **Zeichen vom aktiven Kanal nach AC**

Die Routine liest ein Zeichen vom aktiven Kanal in den Akkumulator, also im Normalfall von der Tastatur, es sei denn, mit "kchkin" ist der Eingabekanal zuvor auf ein anderes Gerät umgeleitet worden.

kclall **$FFE7** **Filetabelle löschen, Vorgabe-I/O**

Diese Routine setzt zunächst wieder die Standard Ein-/Ausgabe-Kanäle und schließt gleichzeitig alle eventuell offenen Dateien aller Gerätenummern.

kclock **$FFEA** **Systemuhr aktualisieren**

Diese Routine, die vom Interrupt aufgerufen wird, erhöht die Systemuhr um eine 1/60-Sekunde. Sie sollte nur bernutzt werden, wenn Sie eine eigene Interrupt-Routine und die Systemuhr in Ihrem Programm verwenden. Es muß allerdings in der Interrupt-Routine sichergestellt sein, daß nur der periodisch ausgelöste System-Interrupt und nicht etwa Interrupts durch Ein-/Ausgabe-Bausteine diese Routine aufrufen; ansonsten ist mit großen Abweichungen in der Genauigkeit der Uhr zu rechnen.

kscror **$FFED** **Bildschirm-Organisation lesen**

Diese Routine ruft die unter den Editor-Funktionen beschriebene Funktion "jscorg" auf.

kplot **$FFF0** **Cursor-Position lesen/schreiben**

Diese Routine ruft die unter den Editor-Funktionen beschriebene Funktion "jplot" auf.

kiobas **$FFF3** **Basis-Adresse I/O-Bausteine nach xr/yr**

Hiermit erhalten Sie nach dem Aufruf die Basis-Adresse der Ein-/Ausgabe-Bausteine ($D000) im X-Register (Lowbyte) und Y-Register (Highbyte).

1.3 Beispielprogramme

Im folgenden finden Sie eine Reihe von Beispielprogrammen, die Sie in eigenen Programmen verwenden und die Ihnen als Anregung für eigene Routinen dienen können. Einige Routinen sind bewußt nicht optimiert, um das Verständnis zu erleichtern. Auch sind die Routinen recht allgemein gehalten und können daher auf Ihre eigenen Bedürfnisse angepaßt werden.

1.3.1 Unterprogramme zur VDC-Steuerung

Folgende Unterprogramme werden vorgestellt:

Name	Funktion
setsys	Einblenden der I/O-Bausteine in die aktuelle Speicherbank ($D000-$DFFF).
rstsys	Ausblenden der I/O-Bausteine aus der aktuellen Speicherbank ($D000-$DFFF)
setadr	Setzen der aktuellen Adresse im VDC-RAM über die Variablen "vdcpos".
setit	Setzen der aktuellen Adresse im VDC-RAM über das X- und Y-Register
putvdc	Schreiben eines Zeichens in das VDC-RAM über die Variable "vdcpos" und einen Offset im Y-Register
getvdc	Lesen eines Zeichens aus dem VDC-RAM über die Variable "vdcpos" und einem Offset im Y-Register
putvdb	Schreiben eines Zeichens in das VDC-RAM über das X- und Y-Register
getvdb	Lesen eines Zeichens aus dem VDC-RAM über das X- und Y-Register
movabs	Schreiben von mehreren gleichen Zeichen in das VDC-RAM
putind	Schreiben von mehreren Zeichen aus dem Speicher in das VDC-RAM

getind Lesen von mehreren Zeichen aus dem VDC-RAM in den Speicher

Die Beispielprogramme laufen hier im Bereich ab $1300, der vom BASIC nicht verwendet wird. Die verwendeten Speicheradressen werden weder vom BASIC noch vom Betriebssystem verwendet und stehen daher frei zur Verfügung.

Zunächst die Definition der Variablen:

```
00002  0000
00003  0000  *** Variablen-Definitionen
00004  0000
00005  0000            vdcpos = $fa          aktuelle adresse im
                                              vdc-ram (2 bytes)
00006  0000            rettx  = $fc          rettungsregister für
                                              x-register
00007  0000            index  = $fd          index in speicher (2
                                              bytes)
00008  0000            count  = $ff          zwischenzähler für
                                              operationen
00009  0000
00010  0000            vdcadr = $d600        vdc adress-register
00011  0000            vdcdat = $d601        vdc daten-register
00012  0000            mmu    = $ff00        mmu configuration-register
00013  0000
00014  0000                   *=$1300        startadresse, nur als
                                              beispiel
```

setsys **Einblenden der I/O-Bausteine in die aktuelle Speicherbank ($D000-$DFFF).**

Der aktuelle Inhalt des X-Registers wird in einer separaten Variablen gerettet, da er durch einige der VDC-Routinen verändert wird. Außerdem werden die I/O-Bereiche von $D000 - $DFFF in die aktuelle Bank eingeblendet, indem das Bit 0 des MMU-Konfigurations-Register gelöscht wird.

```
1300
1300  *** Allgemeine Routine zum Einschalten der
1300  *** Ein-/Ausgabe-Bausteine vor I/O-Zugriffen
1300
1300  48          setsys pha              akku auf stapel retten
1301  86 fc             stx rettx         x-register retten
1303  ad 00 ff          lda mmu           mmu-register
1306  29 fe             and #%11111110    i/o-bit löschen
1308  8d 00 ff          sta mmu           mmu-register setzen
130b  68                pla               akku vom stapel zurück
130c  60                rts               zurück zum hauptprogramm
130d
```

rstsys **Ausblenden der I/O-Bausteine aus der aktuellen Speicherbank ($D000-$DFFF).**

Der Inhalt des X-Registers wird wieder zurückgeholt und die I/O-Bereiche von $D000 - $DFFF aus der aktuellen Bank wieder ausgeblendet, indem das Bit 0 des MMU-Konfigurations-Registers gesetzt wird.

```
130d
130d  *** Allgemeine Routine zum Rückschalten auf die
130d  *** Original-Verteilung nach I/O-Zugriffen
130d
130d  08          rstsys php              bedingungsflags auf stapel retten
130e  48                 pha              akku auf stapel retten
130f  ad 00 ff           lda mmu          original
1312  09 01             ora #%00000001   i/o-bit wieder setzen
1314  8d 00 ff          sta mmu          wieder setzen
1317  a6 fc              ldx rettx        x-register zurücksetzen
1319  68                 pla              akku vom stapel zurück
131a  28                 plp              bedingungsflags vom stapel zurück
131b  60                 rts              zurück ins hauptprogramm
131c
```

setadr **Setzen der aktuellen Adresse im VDC-RAM über die Variable "vdcpos".**

Die Speicherstellen "vdcpos" und "vdcpos+1" enthalten die Adresse innerhalb des VDC-RAMs, die zu setzen ist.

```
131c
131c  *** Unterprogramm zum Setzen der
131c  *** aktuellen Position im VDC
131c
131c  ** parameter:        vdcpos   = gewünschte adresse low-byte
131c                       vdcpos+1 = gewünschte adresse high-byte
131c
131c  20 00 13   setadr jsr setsys       i/o-register einblenden
131f  a5 fb             lda vdcpos+1     high-byte der adresse
1321  a2 12             ldx #$12         register 18 im vdc
1323  8e 00 d6          stx vdcadr       register-nummer in vdc
1326  2c 00 d6   setad1 bit vdcadr       vdc fertig ?
1329  10 fb             bpl setad1       nein -> warten
132b  8d 01 d6          sta vdcdat       akku in vdc datenregister
132e  a5 fa             lda vdcpos       low-byte der adresse
1330  e8                inx              register 19 im vdc
1331  8e 00 d6          stx vdcadr       register-nummer in vdc
1334  2c 00 d6   setad2 bit vdcadr       vdc fertig ?
1337  10 fb             bpl setad2       nein -> warten
1339  8d 01 d6          sta vdcdat       akku in vdc datenregister
133c  4c 0d 13          jmp rstsys       mmu wieder zurücksetzen
133f
```

setit **Setzen der aktuellen Adresse im VDC-RAM über das X- und Y-Register**

Die zu setzende Adresse ist im X-Register (Low-Byte) und Y-Register (High-Byte) enthalten.

```
133f
133f  *** Unterprogramm zum Setzen der
133f  *** aktuellen Position im VDC
133f
133f  ** parameter:        x-reg    = gewünschte adresse low-byte
133f                       y-reg    = gewünschte adresse high-byte
133f
133f  20 00 13   setit  jsr setsys      i/o-register einblenden
1342  8a                txa             low-byte der adresse
1343  a2 13             ldx #$13        register 19 im vdc
1345  8e 00 d6          stx vdcadr      register-nummer in vdc
1348  2c 00 d6   setit1 bit vdcadr      vdc fertig ?
134b  10 fb             bpl setit1      nein -> warten
134d  8d 01 d6          sta vdcdat      akku in vdc datenregister
1350  98                tya             low-byte der adresse
1351  ca                dex             register 18 im vdc
1352  8e 00 d6          stx vdcadr      register-nummer in vdc
1355  2c 00 d6   setit2 bit vdcadr      vdc fertig ?
1358  10 fb             bpl setit2      nein -> warten
135a  8d 01 d6          sta vdcdat      akku in vdc datenregister
135d  4c 0d 13          jmp rstsys      mmu wieder zurücksetzen
1360
```

putvdc **Schreiben eines Zeichens in das VDC-RAM über die Variable "vdcpos" und einen Offset im Y-Register**

Die Speicherstellen "vdcpos" und "vdcpos+1" enthalten die Adresse im VDC-RAM, an die das Zeichen geschrieben werden soll. Zusätzlich wird vor dem Schreiben noch der Inhalt des Y-Registers zu dieser Adresse addiert. Hiermit kann durch mehrfaches Aufrufen der Routine durch Erhöhen des Y-Registers fortlaufend in den VDC eingeschrieben werden.

```
1360
1360  *** Unterprogramm zum Schreiben
1360  *** eines Zeichens in das VDC-RAM
1360
1360  ** parameter:        vdcpos   = gewünschte adresse low-byte
1360                       vdcpos+1 = gewünschte adresse high-byte
1360                       y-reg    = offset zu dieser adresse
1360                       accu     = zu schreibendes zeichen
1360
1360  48          putvdc pha             akku auf stapel retten
1361  20 00 13           jsr setsys      i/o-bausteine einblenden
1364  98                 tya             offset aus y-reg nach akku
1365  18                 clc             carry-flag löschen
1366  65 fa              adc vdcpos      zu low-byte addieren
1368  48                 pha             auf stapel retten
```

```
1369  a5 fb          lda vdcpos+1     high-byte der adresse
136b  69 00          adc #0           carry-flag addieren
136d  a2 12          ldx #$12         register 18 des vdc
136f  8e 00 d6       stx vdcadr       in vdc-register bringen
1372  2c 00 d6 putvd1 bit vdcadr       vdc fertig ?
1375  10 fb          bpl putvd1       nein -> warten
1377  8d 01 d6       sta vdcdat       akku in vdc-datenregister
137a  68             pla              low-byte der adresse
137b  e8             inx              register 19 des vdc
137c  8e 00 d6       stx vdcadr       in vdc-register bringen
137f  2c 00 d6 putvd2 bit vdcadr       vdc fertig ?
1382  10 fb          bpl putvd2       nein -> warten
1384  8d 01 d6       sta vdcdat       akku in vdc-datenregister
1387  68             pla              zu schreibendes zeichen
1388  a2 1f          ldx #$1f         register 31
138a  8e 00 d6       stx vdcadr       in vdc-register schreiben
138d  2c 00 d6 putvd3 bit vdcadr       vdc fertig ?
1390  10 fb          bpl putvd3       nein -> warten
1392  8d 01 d6       sta vdcdat       zeichen schreiben
1395  4c 0d 13       jmp rstsys       zurück
1398
```

getvdc **Lesen eines Zeichens aus dem VDC-RAM über die Variable "vdcpos" und einen Offset im Y-Register**

Die Speicherstellen "vdcpos" und "vdcpos+1" enthalten die Adresse im VDC-RAM, aus der das Zeichen gelesen werden soll. Zusätzlich wird zu dieser Adresse noch der Inhalt des Y-Registers addiert. Hiermit können durch einfaches Erhöhen des Y-Registers mehrere Zeichen hintereinander gelesen werden.

```
1398
1398  *** Unterprogramm zum Lesen eines
1398  *** Zeichens aus dem VDC-RAM
1398
1398  ** parameter:        vdcpos   = gewünschte adresse low-byte
1398                       vdcpos+1 = gewünschte adresse high-byte
1398                       y-reg    = offset zu dieser adresse
1398                       accu     = enthält das zeichen
1398
1398  20 00 13  getvdc jsr setsys       i/o-bausteine einblenden
139b  98               tya              offset aus y-reg nach akku
139c  18               clc              carry-flag löschen
139d  65 fa            adc vdcpos       zu low-byte addieren
139f  48               pha              auf stapel retten
13a0  a5 fb            lda vdcpos+1     high-byte der adresse
13a2  69 00            adc #0           carry-flag addieren
13a4  a2 12            ldx #$12         register 18 des vdc
13a6  8e 00 d6         stx vdcadr       in vdc-register bringen
13a9  2c 00 d6 getvd1  bit vdcadr       vdc fertig ?
13ac  10 fb            bpl getvd1       nein -> warten
13ae  8d 01 d6         sta vdcdat       akku in vdc-datenregister
13b1  68               pla              low-byte der adresse
13b2  e8               inx              register 19 des vdc
```

```
13b3  8e 00 d6          stx vdcadr      in vdc-register bringen
13b6  2c 00 d6  getvd2  bit vdcadr      vdc fertig ?
13b9  10 fb             bpl getvd2      nein -> warten
13bb  8d 01 d6          sta vdcdat      akku in vdc-datenregister
13be  a2 1f             ldx #$1f        register 31
13c0  8e 00 d6          stx vdcadr      in vdc-register schreiben
13c3  2c 00 d6  getvd3  bit vdcadr      vdc fertig ?
13c6  10 fb             bpl getvd3      nein -> warten
13c8  ad 01 d6          lda vdcdat      zeichen auslesen
13cb  4c 0d 13          jmp rstsys      zurück
13ce
```

putvdb **Schreiben eines Zeichens in das VDC-RAM über das X- und Y-Register**

Hierbei enthalten die drei Prozessor-Register alle notwendigen Daten. Das X-Register enthält das Low-Byte der Adresse, das Y-Register das High-Byte und der Akkumulator das zu schreibende Zeichen.

```
13ce
13ce  *** Unterprogramm zum Schreiben
13ce  *** eines Zeichens in das VDC-RAM
13ce
13ce  ** parameter:       x-reg    = gewünschte adresse low-byte
13ce                      y-reg    = gewünschte adresse high-byte
13ce                      accu     = zu schreibendes zeichen
13ce
13ce  20 00 13  putvdb jsr setsys       i/o-bereich einblenden
13d1  48               pha              akku auf stapel retten
13d2  8a               txa              low-byte der adresse
13d3  a2 13            ldx #$13         register 19 des vdc
13d5  8e 00 d6         stx vdcadr       registernummer setzen
13d8  2c 00 d6  putvb1 bit vdcadr       vdc fertig ?
13db  10 fb            bpl putvb1       nein -> warten
13dd  8d 01 d6         sta vdcdat       akku in vdc-datenregister
13e0  98               tya              high-byte des adresse
13e1  ca               dex              register 18 des vdc
13e2  8e 00 d6         stx vdcadr       registernummer setzen
13e5  2c 00 d6  putvb2 bit vdcadr       vdc fertig ?
13e8  10 fb            bpl putvb2       nein -> warten
13ea  8d 01 d6         sta vdcdat       akku in vdc-datenregister
13ed  68               pla              zu schreibendes zeichen
13ee  a2 1f            ldx #$1f         register 31
13f0  8e 00 d6         stx vdcadr       in vdc-register schreiben
13f3  2c 00 d6  putvb3 bit vdcadr       vdc fertig ?
13f6  10 fb            bpl putvb3       nein -> warten
13f8  8d 01 d6         sta vdcdat       zeichen einschreiben
13fb  4c 0d 13         jmp rstsys       zurück
13fe
```

getvdb **Lesen eines Zeichens aus dem VDC-RAM über das X- und Y-Register**

Ein Zeichen wird aus dem VDC-RAM gelesen, wobei die Adresse im X-Register (Low-Byte) und Y-Register (High-Byte) angegeben ist. Nach dem Aufruf der Funktion enthält der Akkumulator das gelesene Zeichen.

```
13fe
13fe  *** Unterprogramm zum Lesen eines
13fe  *** Zeichens aus dem VDC-RAM
13fe
13fe  ** parameter:        x-reg    = gewünschte adresse low-byte
13fe                       y-reg    = gewünschte adresse high-byte
13fe                       accu     = enthält das gelesene zeichen
13fe
13fe  20 00 13   getvdb jsr setsys      i/o-bereich einblenden
1401  8a                txa             low-byte der adresse
1402  a2 13             ldx #$13        register 19 des vdc
1404  8e 00 d6          stx vdcadr      registernummer setzen
1407  2c 00 d6   getvb1 bit vdcadr      vdc fertig ?
140a  10 fb             bpl getvb1      nein -> warten
140c  8d 01 d6          sta vdcdat      akku in vdc-datenregister
140f  98                tya             high-byte der adresse
1410  ca                dex             register 18 des vdc
1411  8e 00 d6          stx vdcadr      registernummer setzen
1414  2c 00 d6   getvb2 bit vdcadr      vdc fertig ?
1417  10 fb             bpl getvb2      nein -> warten
1419  8d 01 d6          sta vdcdat      akku in vdc-datenregister
141c  a2 1f             ldx #$1f        register 31
141e  8e 00 d6          stx vdcadr      in vdc-register schreiben
1421  2c 00 d6   getvb3 bit vdcadr      vdc fertig ?
1424  10 fb             bpl getvb3      nein -> warten
1426  ad 01 d6          lda vdcdat      zeichen auslesen
1429  4c 0d 13          jmp rstsys      zurück
142c
```

movabs **Schreiben vom mehreren gleichen Zeichen in das VDC-RAM**

Diese Routine kann mehrere gleiche Zeichen in das VDC-RAM schreiben. Hierzu enthalten das X-Register das Low-Byte der Adresse, das Y-Register das High-Byte der Adresse und der Akkumulator das zu schreibende Zeichen. Außerdem muß in der Speicherstelle "count" die Anzahl zu schreibender Zeichen angegeben werden. Die Routine ist erheblich schneller als das einzelne Einschreiben von Zeichen und sollte immer dann verwendet werden, wenn gleichartige Zeichen, z.B. Leerzeichen, zum Löschen von Bereichen zu schreiben sind.

```
142c
142c  *** Unterprogramm zum Schreiben von mehreren
142c  *** gleichen Zeichen in das VDC-RAM
142c
142c  ** parameter:        x-reg    = gewünschte adresse low-byte
142c                       y-reg    = gewünschte adresse high-byte
142c                       accu     = zu schreibendes zeichen
142c                       count    = anzahl zeichen
142c
142c  20 3f 13   movabs jsr setit       adresse im vdc setzen
142f  20 00 13         jsr setsys      i/o-bereich einblenden
1432  a2 1f            ldx #$1f        register-nummer 31
1434  8e 00 d6         stx vdcadr      in vdc-register schreiben
1437  2c 00 d6   movab1 bit vdcadr     vdc fertig ?
143a  10 fb            bpl movab1      nein -> warten
143c  8d 01 d6         sta vdcdat      zeichen in schreiben
143f  c6 ff            dec count       anzahl - 1
1441  d0 f4            bne movab1      wenn nicht null -> weiter
1443  4c 0d 13         jmp rstsys      zurück ins hauptprogramm
1446
```

putind **Schreiben von mehreren Zeichen aus dem Speicher in das VDC-RAM**

Die Routine übergibt einen gesamten Speicherbereich von maximal 256 Bytes an den VDC. Die Adresse, wo die zu schreibenden Bytes sich befinden, werden in den Speicherstellen "index" und "index+1" angegeben. Die Adresse im VDC befindet sich im X-Register (Low-Byte) und Y-Register (High-Byte). Der Akku enthält die Anzahl der zu schreibenden Zeichen.

```
1464  *** Unterprogramm zum Schreiben von mehreren
1464  *** Zeichen in das VDC-RAM
1464
1464  ** parameter:        index    = gewünschte adresse low-byte
1464                       index+1  = gewünschte adresse high-byte
1464                       accu     = anzahl zeichen
1464                       x-reg    = adresse im vdc low-byte
1464                       y-reg    = adresse im vdc high-byte
1464
1464  85 ff      putind sta count       anzahl zeichen retten
1466  20 3f 13         jsr setit       adresse im vdc setzen
1469  a2 1f            ldx #$1f        register 31
146b  a0 00            ldy #$00        index in daten
146d  b1 fd      putin1 lda (index),y  zeichen aus daten
146f  8e 00 d6         stx vdcadr      vdc-register schreiben
1472  2c 00 d6   putin2 bit vdcadr     vdc fertig ?
1475  10 fb            bpl putin2      nein -> warten
1477  8d 01 d6         sta vdcdat      zeichen schreiben
147a  c8                iny            index erhöhen
147b  c6 ff            dec count       anzahl - 1
147d  d0 ee            bne putin1      wenn nicht null -> weiter
147f  4c 0d 13         jmp rstsys      zurück
1482
```

getind **Lesen von mehreren Zeichen aus dem VDC-RAM in den Speicher**

Die Routine liest einen gesamten Speicherbereich vom maximal 256 Bytes aus dem VDC-RAM. Die Adresse, wohin die gelesenen Bytes zu bringen sind, wird in den Speicherstellen "index" und "index+1" angegeben. Die Adresse im VDC befindet sich im X-Register (Low-Byte) und Y-Register (High-Byte). Der Akku enthält die Anzahl der zu lesenden Zeichen.

```
1446
1446  *** Unterprogramm zum Lesen von mehreren
1446  *** Zeichen aus dem VDC-RAM
1446
1446  ** parameter:        index    = gewünschte adresse low-byte
1446                       index+1  = gewünschte adresse high-byte
1446                       accu     = anzahl zeichen
1446                       x-reg    = adresse im vdc low-byte
1446                       y-reg    = adresse im vdc high-byte
1446
1446  85 ff      getind sta count        anzahl zeichen retten
1448  20 3f 13         jsr setit         adresse im vdc setzen
144b  a0 00            ldy #$00          index in adresse
144d  a2 1f            ldx #$1f          register 31
144f  8e 00 d6   getin1 stx vdcadr        vdc-register schreiben
1452  2c 00 d6   getin2 bit vdcadr        vdc fertig ?
1455  10 fb            bpl getin2        nein -> warten
1457  ad 01 d6         lda vdcdat        zeichen aus lesen
145a  91 fd            sta (index),y     in adresse schreiben
145c  c8               iny               index erhöhen
145d  c6 ff            dec count         anzahl - 1
145f  d0 ee            bne getin1        wenn nicht null -> weiter
1461  4c 0d 13         jmp rstsys        zurück
1464
```

1.3.2 Unterprogramme zum Datentransfer

Folgende Unterprogramme werden vorgestellt:

Name	Funktion
swap	Speicherseiten in beiden Banks vertauschen
getbyt	Ein Byte aus anderer Bank lesen
putbyt	Ein Byte in andere Bank schreiben
cmpbyt	Ein Byte mit anderer Bank vergleichen

Alle diese Unterprogramme müssen in einem Bereich laufen, der beiden RAM-Banks gemeinsam ist. Die Speicherverteilung kann im RAM-Konfigurations-Register der MMU gesetzt werden. Die Beispielprogramme laufen hier im reservierten Bereich ab $1300. Hierzu müßte z.B. der gemeinsame Bereich auf 8 KByte gesetzt werden. Beachten Sie bitte: Wenn Sie Assem-

bler-Routinen vom BASIC her verwenden und hierzu, wie oben erwähnt, den gemeinsamen Bereich vergrößern, müssen *unbedingt* die BASIC-Variablen-Zeiger der neuen Speicherverteilung angepaßt werden, sonst wird Ihr Programm durch die BASIC-Variablen überschrieben.

Sie können hierzu folgende BASIC-Befehle verwenden:

POKE 48,x : POKE 50,x : POKE 52,x : CLR

wobei für "x" folgende Werte einzusetzen sind:

Gemeinsamer Bereich	x	Wert für MMU-Register $D506
1 KByte	4	% xxxx0100
4 KByte	16	% xxxx0101
8 KByte	32	% xxxx0110
16 KByte	64	% xxxx0111

Hier zunächst wieder die Definition der Variablen. Sie liegen wieder in dem weder vom BASIC noch vom Betriebssystem verwendeten Bereich.

```
0000
0000  *** Variablen-Definitionen
0000
0000            index  = $fa
0000            count  = $fc
0000
0000            mmu    = $ff00          mmu konfiguration-register
0000
0000                   *=$1300          startadresse
```

swap **Speicherseiten in beiden Banks vertauschen**

Diese Routine vertauscht die Inhalte der beiden Speicherbanks. Hierzu wird die Startadresse des zu vertauschenden Bereichs im X-Register (Low-Byte) und Y-Register (High-Byte) angegeben sowie im Akkumulator die Anzahl zu vertauschender Speicherseiten zu jeweils 256 Bytes.

```
1300  *** Unterroutine zum Austauschen von Speicherbereichen
1300  *** zwischen den beiden RAM-Banks
1300
1300  ** parameter:     x-reg = startadresse low-byte
1300                    y-reg = startadresse high-byte
1300                    accu  = anzahl pages
1300
1300  78          swap  sei              interrupt sperren
1301  86 fa             stx index        startadresse low
1303  84 fb             sty index+1      startadresse high
1305  85 fc             sta count        anzahl pages
1307  ad 00 ff          lda mmu          mmu-register laden
130a  48                pha              und auf dem stapel retten
130b  a0 00       swap01 ldy #0           index in speicher
```

```
130d  a2 3f      swap02 ldx #$3f         bank 1
130f  8e 00 ff          stx mmu          setzen
1312  b1 fa              lda (index),y    byte aus bank 1
1314  48                pha              auf dem stapel retten
1315  a2 7f              ldx #$7f         bank 2
1317  8e 00 ff          stx mmu          setzen
131a  b1 fa              lda (index),y    byte aus bank 2
131c  a2 3f              ldx #$3f         bank  1
131e  8e 00 ff          stx mmu          setzen
1321  91 fa              sta (index),y    in bank 1 speichern
1323  68                pla              zeichen aus bank 1 zurück
1324  a2 7f              ldx #$7f         bank 2
1326  8e 00 ff          stx mmu          setzen
1329  91 fa              sta (index),y    und in bank 2 speichern
132b  c8                iny              index in speicher erhöhen
132c  d0 df              bne swap02       wenn nicht null -> weiter
132e  e6 fb              inc index+1      index-highbyte erhöhen
1330  c6 fc              dec count        zähler vermindern
1332  d0 d7              bne swap01       wenn nicht null -> weiter
1334  68                pla              mmu-register vom stapel
1335  8d 00 ff          sta mmu          mmu wieder setzen
1338  58                cli              interrupt freigeben
1339  60                rts              zurück zum hauptprogramm
133a
```

getbyt **Ein Byte aus der anderen Bank lesen**

Die Routine liest ein einzelnes Byte aus der jeweils anderen Speicherbank. Hierzu müssen die Speicheradressen "index" und "index+1" die Adresse des Bytes enthalten, wobei noch im Y-Register ein Offset angegeben werden kann. Das X-Register kann auch die Adresse einer anderer Index-Speicherstelle enthalten, dann wird aus dieser Adresse gelesen.

Je nach Funktion kann die Routine "getbyt" oder "getby0" aufgerufen werden. Bei "getbyt" wird im X-Register die Indexadresse erwartet; bei "getby0" wird die zuletzt verwendete angenommen.

```
133a  *** Unterroutine zum Lesen eines Bytes aus der jeweils
133a  *** anderen Speicherbank
133a  ** parameter:    x-reg = indexadresse (wahlweise)
133a                   y-reg = offset zur index-adresse
133a
133a  8e 47 13   getbyt stx getby1+1     gewünschte index-adresse
133d
133d  ae 00 ff   getby0 ldx mmu          mmu-registers
1340  8a                txa              in akku bringen
1341  49 80              eor #%10000000   oberstes bit flippen
1343  8d 00 ff          sta mmu          in mmu-register bringen
1346  b1 fa      getby1 lda (index),y    byte aus anderer bank
1348  8e 00 ff          stx mmu          mmu originalwert setzen
134b  60                rts              zurück ins hauptprogramm
134c
```

putbyt **Ein Byte in die andere Bank schreiben**

Die Routine schreibt ein einzelnes Byte in die jeweils andere Speicherbank. Hierzu müssen die Speicheradressen "index" und "index+1" die Adresse des Bytes enthalten, wobei noch im Y-Register ein Offset angegeben werden kann. Das X-Register kann auch die Adresse einer anderen Index-Speicherstelle enthalten, dann wird in diese Adresse geschrieben.

Je nach Funktion kann die Routine "putbyt" oder "putby0" aufgerufen werden. Bei "putbyt" wird im X-Register die Indexadresse erwartet; bei "putby0" wird die zuletzt verwendete angenommen.

```
134c
134c  *** Unterroutine zum Schreiben eines Bytes in
134c  *** die jeweils andere Speicherbank
134c
134c  ** parameter:     accu  = zu schreibendes byte
134c                    x-reg = indexadresse (wahlweise)
134c                    y-reg = offset zur index-adresse
134c
134c  8e 5b 13   putbyt stx putby1+1     gewünschte index-adresse
134f
134f  48         putby0 pha              akku retten
1350  ae 00 ff          ldx mmu          mmu-registers
1353  8a                txa              in akku bringen
1354  49 80             eor #%10000000   oberstes bit flippen
1356  8d 00 ff          sta mmu          in mmu-register bringen
1359  68                pla              zu schreibendes byte
135a  91 fa      putby1 sta (index),y    byte in andere bank
135c  8e 00 ff          stx mmu          mmu originalwert setzen
135f  60                rts              zurück ins hauptprogramm
1360
```

cmpbyt **Ein Byte mit der anderen Bank vergleichen**

Die Routine vergleicht ein einzelnes Byte mit der jeweils anderen Speicherbank. Hierzu müssen die Speicheradressen "index" und "index+1" die Adresse des Bytes enthalten, wobei noch im Y-Register ein Offset angegeben werden kann. Das X-Register kann auch die Adresse einer anderen Index-Speicherstelle enthalten, dann wird mit dieser Adresse verglichen.

Je nach Funktion kann die Routine "cmpbyt" oder "cmpby0" aufgerufen werden. Bei "cmpbyt" wird im X-Register die Indexadresse erwartet; bei "cmpby0" wird die zuletzt verwendete angenommen.

```
1360
1360  *** Unterroutine zum Vergleichen eines Bytes mit
1360  *** der jeweils anderen Speicherbank
1360
1360  ** parameter:     accu  = zu vergleichendes byte
1360                    x-reg = indexadresse (wahlweise)
```

```
1360                    y-reg = offset zur index-adresse
1360
1360  8e 6f 13   cmpbyt stx cmpby1+1      gewünschte index-adresse
1363
1363  48         cmpby0 pha               akku retten
1364  ae 00 ff          ldx mmu           mmu-registers
1367  8a                 txa               in akku bringen
1368  49 80              eor #%10000000    oberstes bit flippen
136a  8d 00 ff           sta mmu           in mmu-register bringen
136d  68                 pla               zu schreibendes byte
136e  d1 fa      cmpby1 cmp (index),y     byte aus anderer bank
1370  8e 00 ff          stx mmu           mmu originalwert setzen
1373  60                 rts               zurück ins hauptprogramm
1374
```

1.3.3 Beispielprogramm zur CIA-Behandlung

Im folgenden finden Sie ein Programm, daß eine Centronics-Parallel-Schnittstelle am User-Port des C-128 simuliert. Sie können hier mit Hilfe eines einfach herzustellenden Kabels einen entsprechenden Drucker direkt ansprechen, ohne auf teure Hardware-Interfaces angewiesen zu sein.

Die Schnittstelle wird über die Gerätenummer 4 angesprochen, wobei über die Sekundäradresse gewählt werden kann, ob die Zeichen so übertragen werden sollen wie sie sind: also zum Beispiel als Steuerzeichen oder umgewandelt vom Commodore-DIN-Zeichensatz in den Standard-ASCII-Zeichensatz. Zeichen werden ohne Wandlung ausgegeben, wenn die Datei mit der Sekundäradresse 0 eröffnet wurde; jede andere Sekundäradresse wandelt die Zeichen vor der Ausgabe. Sie können natürlich auch zwei Dateien auf der Gerätenummer 4, eine mit Sekundäradresse 0 und die andere mit Sekundäradresse 1 eröffnen, und dann jeweils Steuerzeichen über die eine Datei und Daten über die andere übertragen.

Das Programm schleift sich bei der Initialisierung in zwei Systemroutinen ein, und zwar die Routine zur Eröffnung eines Ausgabe-Kanals und die Routine zur Ausgabe eines Zeichens auf dem aktuellen Kanal. Die Routinen prüfen jeweils, ob die angesprochene Gerätenummer gleich 4 ist und springen dann entweder das beschriebene Programm an oder bei allen anderen Gerätenummern ins Betriebssystem zurück.

Vor dem Listing noch kurz eine Übersicht der Steckerbelegung des notwendigen Kabels. Sie ist im übrigen kompatibel zu den meisten bereits für den C-64 benutzten Kabeln.

Steckerbelegung:

Userport C-128			Druckerschnittstelle	
Pin	Bezeichnung		Pin	Bezeichnung
B	Flag	(Unterseite)	10	Acknowledge
C	PB0	" "	2	Data 1
D	PB1	" "	3	Data 2
E	PB2	" "	4	Data 3
F	PB3	" "	5	Data 4
H	PB4	" "	6	Data 5
J	PB5	" "	7	Data 6
K	PB6	" "	8	Data 7
L	PB7	" "	9	Data 8
M	PA2	" "	1	Strobe
N	GND	" "	19-30	Ground

Simulation einer Centronics-Schnittstelle am User-Port:

```
00001  0000
00002  0000
00003  0000  *** Variablen-Definitionen
00004  0000
00005  0000            bsour  = $95          aktuelles ausgabezeichen
00006  0000            dflto  = $9a          ausgabe-gerätenummer
00007  0000            sa     = $b9          aktuelle sekundäradresse
00008  0000
00009  0000            ickout = $0320        vector: ausgabekanal erröffnen
00010  0000            ibsout = $0326        vector: zeichen ausgeben
00011  0000
00012  0000            d1prb  = $dc01        cia #1 port b
00013  0000            d2pra  = $dd00        cia #2 port a
00014  0000            d2prb  = $dd01        cia #2 port b
00015  0000            d2ddra = $dd02        cia #2 richtungsreg. a
00016  0000            d2ddrb = $dd03        cia #2 richtungsreg. b
00017  0000            d2icr  = $dd0d        cia #2 interupt kontr.reg.
00018  0000
00019  0000            cko232 = $e729        kanal auf rs232 eröffnen
00020  0000            nbsoux = $ef7c        ausgabe nicht auf userport
00021  0000            ckotap = $f163        kanal auf kassette
00022  0000            ckoser = $f16d        kanal auf seriellem bus
00023  0000            lookup = $f202        suchen ob datei schon offen
00024  0000            getlfs = $f212        la, fa und sa setzen
00025  0000            error3 = $f682        fehler: file not open
00026  0000            error7 = $f68e        fehler: not output file
00027  0000
00028  0000                   *=$1300
00029  1300
00030  1300  *** Initialisierung der Routinen und Einbindung
00031  1300  *** in das Betriebs-System
00032  1300
00033  1300  a9 15      init   lda #<pckout   low-byte neue routine
00034  1302  8d 20 03          sta ickout     in vector bringen
```

```
00035  1305  a9 13           lda #>pckout     high-byte neue routine
00036  1307  8d 21 03        sta ickout+1     in vector+1 bringen
00037  130a  a9 40           lda #<pbsout     low-byte neue routine
00038  130c  8d 26 03        sta ibsout       in vector bringen
00039  130f  a9 13           lda #>pbsout     high-byte neue routine
00040  1311  8d 27 03        sta ibsout+1     in vector+1 bringen
00041  1314  60              rts              fertig
00042  1315
00043  1315  *** Einsprung aus dem Betriebs-System bei Aufruf
00044  1315  *** der Funktion "ausgabe-kanal eröffnen"
00045  1315
00046  1315  20 02 f2  pckout jsr lookup      datei in filetabelle suchen
00047  1318  f0 03           beq pckou1       gefunden -> weiter
00048  131a  4c 82 f6        jmp error3       fehler: file not open
00049  131d
00050  131d  20 12 f2  pckou1 jsr getlfs      parameter der datei setzen
00051  1320  d0 03           bne pckou2       gerätenummer > 0 -> weiter
00052  1322  4c 8e f6        jmp error7       fehler: not output file
00053  1325
00054  1325  c9 03      pckou2 cmp #$03       adresse = 3 (bildschirm) ?
00055  1327  f0 10           beq pckou5       ja -> zurück ins system
00056  1329  b0 0a           bcs pckou4       adresse > 3 (seriell) ->weiter
00057  132b  c9 02           cmp #$02         adresse = 2 (rs232) ?
00058  132d  d0 03           bne pckou3       nein -> kassette, weiter
00059  132f  4c 29 e7        jmp cko232       v24-behandeln
00060  1332
00061  1332  4c 63 f1  pckou3 jmp ckotap      kassette behandeln
00062  1335
00063  1335  c9 04      pckou4 cmp #$04       gerätenummer = 4 ?
00064  1337  d0 04           bne pckou6       nein -> serieller bus
00065  1339  85 9a      pckou5 sta dflto      ausgabekanal setzen
00066  133b  18              clc              fehler-flag löschen
00067  133c  60              rts              zurück ins system
00068  133d
00069  133d  4c 6d f1  pckou6 jmp ckoser      seriellen bus behandeln
00070  1340
00071  1340  *** Einsprung aus dem System bei Funktion
00072  1340  *** "bsout", ein Zeichen auf Kanal ausgeben
00073  1340
00074  1340  48         pbsout pha            zeichen retten
00075  1341  a5 9a           lda dflto        ausgabe-kanal
00076  1343  c9 04           cmp #$04         adresse = 4 ?
00077  1345  f0 03           beq pbsou1       ja -> parallelschnittstelle
00078  1347  4c 7c ef        jmp nbsoux       nein -> zurück ins system
00079  134a
00080  134a  68         pbsou1 pla            zeichen zurück
00081  134b  85 95           sta bsour        in ausgabebyte bringen
00082  134d  8a              txa              x-register
00083  134e  48              pha              auf stapel retten
00084  134f  a5 95           lda bsour        aktuelles ausgabezeichen
00085  1351  a6 b9           ldx sa           sekundäradresse
00086  1353  f0 04           beq pbsou2       null -> weiter
00087  1355  aa              tax              zeichen ins x-register
00088  1356  bd 92 13        lda wandtb,x     dazu gewandeltes zeichen
00089  1359  20 61 13  pbsou2 jsr parout      zeichen ausgeben
00090  135c  68              pla              x-register
00091  135d  aa              tax              vom stapel zurückholen
00092  135e  a5 95           lda bsour        aktuelles zeichen
```

```
00093  1360  60              rts             zurück ins system
00094  1361
00095  1361  *** Unterprogramm zur Ausgabe des Akku auf der
00096  1361  *** Parallel-Schnittstelle, kann auch von anderen
00097  1361  *** Programmen direkt aufgerufen werden
00098  1361
00099  1361  48        parout pha             zeichen retten
00100  1362  8d 01 dd        sta d2prb       und nach port b (userport)
00101  1365  a9 3f           lda #%00111111  bit 6+7 eingang, rest ausgang
00102  1367  8d 02 dd        sta d2ddra      port a setzen
00103  136a  a9 ff           lda #%11111111  alle bits ausgänge
00104  136c  8d 03 dd        sta d2ddrb      port b setzen
00105  136f  ad 00 dd        lda d2pra       strobe-leitung
00106  1372  29 fb           and #%11111011  auf bit 2
00107  1374  8d 00 dd        sta d2pra       rücksetzen
00108  1377  ea              nop             verzögerung, wichtig !
00109  1378  2c 01 dc  parou1 bit d1prb       stop-taste gedrückt ?
00110  137b  10 12           bpl parou2      ja -> abbruch
00111  137d  ad 00 dd        lda d2pra       strobe leitung
00112  1380  09 04           ora #%00000100  auf bit 2
00113  1382  8d 00 dd        sta d2pra       setzen
00114  1385  a9 10           lda #%00010000  bitmuster für flag-bit
00115  1387  2c 0d dd        bit d2icr       bit im icr gesetzt ?
00116  138a  f0 ec           beq parou1      nein -> warten
00117  138c  68              pla             zeichen zurück
00118  138d  18              clc             fehlerflag löschen
00119  138e  60              rts             zurück
00120  138f
00121  138f  68        parou2 pla             zeichen zurück
00122  1390  38              sec             fehlerflag setzen
00123  1391  60              rts             zurück
00124  1392
00125  1392  *** Wandlungstabelle für DIN-Tastatur
00126  1392  *** gewandelt wird von Commodore-DIN
00127  1392  *** nach Standard-ASCII
00128  1392  *** Notfalls hier an Drucker anpassen
00129  1392
00130  1392  00 01 02  wandtb .byte $00,$01,$02,$03,$04,$05
00131  1395  03 04 05
00132  1398  06 07 08        .byte $06,$07,$08,$09,$0a,$0b
00133  139b  09 00 0b
00134  139e  0c 0d 0e        .byte $0c,$0d,$0e,$0f,$10,$11
00135  13a1  0f 10 11
00136  13a4  12 13 14        .byte $12,$13,$14,$15,$16,$17
00137  13a7  15 16 17
00138  13aa  18 19 1a        .byte $18,$19,$1a,$1b,$1c,$1d
00139  13ad  1b 1c 1d
00140  13b0  1e 1f 20        .byte $1e,$1f,$20,$21,$22,$23
00141  13b3  21 22 23
00142  13b6  24 25 26        .byte $24,$25,$26,$27,$28,$29
00143  13b9  27 28 29
00144  13bc  2a 2b 2c        .byte $2a,$2b,$2c,$2d,$2e,$2f
00145  13bf  2d 2e 2f
00146  13c2  30 31 32        .byte $30,$31,$32,$33,$34,$35
00147  13c5  33 34 35
00148  13c8  36 37 38        .byte $36,$37,$38,$39,$3a,$3b
00149  13cb  39 3a 3b
00150  13ce  3c 3d 3e        .byte $3c,$3d,$3e,$3f,$40,$61
```

```
00151  13d1  3f 40 61
00152  13d4  62 63 64        .byte $62,$63,$64,$65,$66,$67
00153  13d7  65 66 67
00154  13da  68 69 6a        .byte $68,$69,$6a,$6b,$6c,$6d
00155  13dd  6b 6c 6d
00156  13e0  6e 6f 70        .byte $6e,$6f,$70,$71,$72,$73
00157  13e3  71 72 73
00158  13e6  74 75 76        .byte $74,$75,$76,$77,$78,$79
00159  13e9  77 78 79
00160  13ec  7a 5b 5c        .byte $7a,$5b,$5c,$5d,$5e,$5f
00161  13ef  5d 5e 5f
00162  13f2  60 61 62        .byte $60,$61,$62,$63,$64,$65
00163  13f5  63 64 65
00164  13f8  66 67 68        .byte $66,$67,$68,$69,$6a,$6b
00165  13fb  69 6a 6b
00166  13fe  6c 6d 6e        .byte $6c,$6d,$6e,$6f,$70,$71
00167  1401  6f 70 71
00168  1404  72 73 74        .byte $72,$73,$74,$75,$76,$77
00169  1407  75 76 77
00170  140a  78 79 7a        .byte $78,$79,$7a,$7b,$7c,$7d
00171  140d  7b 7c 7d
00172  1410  7e 7f 80        .byte $7e,$7f,$80,$81,$82,$83
00173  1413  81 82 83
00174  1416  84 85 86        .byte $84,$85,$86,$87,$88,$89
00175  1419  87 88 89
00176  141c  8a 8b 8c        .byte $8a,$8b,$8c,$8d,$8e,$8f
00177  141f  8d 8e 8f
00178  1422  90 91 92        .byte $90,$91,$92,$93,$94,$95
00179  1425  93 94 95
00180  1428  96 97 98        .byte $96,$97,$98,$99,$9a,$9b
00181  142b  99 9a 9b
00182  142e  9c 9d 9e        .byte $9c,$9d,$9e,$9f,$20,$a1
00183  1431  9f 20 a1
00184  1434  a2 a3 a4        .byte $a2,$a3,$a4,$a5,$a6,$a7
00185  1437  a5 a6 a7
00186  143a  a8 a9 aa        .byte $a8,$a9,$aa,$ab,$ac,$ad
00187  143d  ab ac ad
00188  1440  ae af b0        .byte $ae,$af,$b0,$b1,$b2,$b3
00189  1443  b1 b2 b3
00190  1446  b4 b5 b6        .byte $b4,$b5,$b6,$b7,$b8,$b9
00191  1449  b7 b8 b9
00192  144c  ba 7b 7c        .byte $ba,$7b,$7c,$7d,$7e,$bf
00193  144f  7d 7e bf
00194  1452  c0 41 42        .byte $c0,$41,$42,$43,$44,$45
00195  1455  43 44 45
00196  1458  46 47 48        .byte $46,$47,$48,$49,$4a,$4b
00197  145b  49 4a 4b
00198  145e  4c 4d 4e        .byte $4c,$4d,$4e,$4f,$50,$51
00199  1461  4f 50 51
00200  1464  52 53 54        .byte $52,$53,$54,$55,$56,$57
00201  1467  55 56 57
00202  146a  58 59 5a        .byte $58,$59,$5a,$5b,$5c,$5d
00203  146d  5b 5c 5d
00204  1470  5e 5f e0        .byte $5e,$5f,$e0,$e1,$e2,$e3
00205  1473  e1 e2 e3
00206  1476  e4 e5 e6        .byte $e4,$e5,$e6,$e7,$e8,$e9
00207  1479  e7 e8 e9
00208  147c  ea eb ec        .byte $ea,$eb,$ec,$ed,$ee,$ef
```

```
00209  147f  ed ee ef
00210  1482  f0 f1 f2          .byte $f0,$f1,$f2,$f3,$f4,$f5
00211  1485  f3 f4 f5
00212  1488  f6 f7 f8          .byte $f6,$f7,$f8,$f9,$fa,$fb
00213  148b  f9 fa fb
00214  148e  fc fd fe          .byte $fc,$fd,$fe,$ff
00215  1491  ff
00216  1492
00217  1492                    .end
```

2 Grundsätzliches zum Assembler und den Assemblerlistings

Die folgenden Assemblerlistings wurde mit dem Hard + Soft Assembler/Re-Assemblersystem erstellt. Sie enthalten, ausführlich dokumentiert, das gesamte Betriebssystem des C-128 im 128-er Modus, getrennt in Monitor und Editor/Kernal. Zum besseren Verständnis über die Arbeitsweise und Leseart sind hier nun einige Informationen gegeben.

Monitor und Editor/Kernal wurden getrennt aufbereitet und assembliert, da es sich um jeweils eigenständige Programmteile handelt. Editor und Kernal sind eigentlich in sich auch geschlossene Programme, wurden aber der Übersichtlichkeit halber zusammen bearbeitet und assembliert.

Die folgenden Listings sind reine Assemblerlistings und nicht etwa kommentierte Dis-Assembler-Ausdrucke. Das Betriebssystem liegt also in ASCII-Sourcefiles vor und wurde mit einem professionellen Assembler assembliert und ausgedruckt. Dabei entsteht ein komplett lauffähiges Programm, daß bis aufs Bit dem Original-Betriebssystem entspricht.

Die verwendeten Labelnamen sind - soweit bekannt - die Original-Labelnamen, wie sie auch von Commodore verwendet wurden. Die restlichen Namen wurden so gewählt, daß möglichst aufgrund des Namens bereits auf die mögliche Funktion der Routine geschlossen werden kann.

Für alle, die bisher noch keine Erfahrungen mit einem Assembler haben, sondern ihre Programme z.B. mit Hilfe eines Monitorprogrammes geschrieben haben, hier einige Hinweise zur Arbeitsweise eines Assemblers und zu Eigenheiten des verwendeten Assemblersystems.

2.1 Einführung in die Assemblerprogrammierung

Den Vorgang, ein mnemonisches bzw. symbolisches Computerprogramm in einen prozessortypischen Maschinencode umzusetzen, nennt man Assemblierung. Ein Programm, welches die Assemblierung durchführt, nennt man Assembler. Die dabei verwendeten Symbole, Befehle und Anwendungsregeln sind die Assemblersprache. Generell wird jeder Begriff einer Assemblersprache vom Assembler in eine Maschineninstruktion übersetzt. Hier ist der Unterschied zum Compiler, der aus einem einzigen Begriff eine Anzahl von Maschineninstruktionen erzeugt. Wird ein Assembler auf

einem anderen Computer (anderer Mikroprozessor) betrieben, als auf dem, für den der Maschinencode generiert werden soll, dann spricht man von einem Cross-Assembler.

Im Prinzip verwenden Computer, und damit auch alle Mikroprozessoren, das duale Zahlensystem (Basis 2) und binäre Codierung für Daten und Befehle. Für den Menschen als Anwender und speziell als Programmierer eignen sich zur Darstellung andere Zahlensysteme besser, wie das oktale (Basis 8), dezimale (Basis 10), oder das hexadezimale (Basis 16) System. Für die Maschinen- bzw. Assemblerprogrammierung scheint die hexadezimale Darstellung am geeignetsten.

Die verschiedenen Zahlensysteme sollen an dem folgenden Befehlsbeispiel gezeigt werden:

'Lade den Akkumulator direkt mit dem Wert "Hex 56" - Befehlscode "Hex A9"

Zahlendarstellung	Befehl	Wert
Hexadezimal	A9	56
Dezimal	169	86
Dual	10101001	01010110

Die numerische Darstellung wie im Beispiel ist in der Anwendung zu mühselig, da der Programmierer dann alle Befehle und deren numerischen Wert wissen müßte. Es wurden deshalb symbolische Ausdrücke, sogenannte Mnemonics, entwickelt.

Bei Verwendung dieser Mnemonics kann das vorhergehende Beispiel folgendermaßen dargestellt werden:

Zahlendarstellung	Befehl und Wert
Hexadezimal	LDA #$56
Dezimal	LDA #86
Dual	LDA #%01010110

Bei dieser Darstellung ist LDA der Mnemonic für den Befehl Lade Akku = $A9, das "#" steht für die Adressierungsart "direkt" und die Bezeichnungen $ für hexadezimal und % für dual. Eine dezimale Angabe erfolgt ohne Vorzeichen.

Der Assembler hat nun die Aufgabe, das Mnemonic "LDA #" in die numerische Form "$A9" zu übersetzen.

Das mnemonische Programm wird als Source-Code oder Quell-Code, das vom Assembler in Maschinencode umgesetzte Programm als Object-Code

bezeichnet. Der Prozessor kann nur den Object-Code verstehen und ausführen.

So wie im Beispiel gezeigt, hat jeder Maschinencode einen symbolischen Namen, den Operations-Code, kurz mit Opcode bezeichnet. Wie bereits ausgeführt, verfügt der Mikroprozessor "650x" und damit auch der "8502" über 56 verschiedene Opcodes. Bei vielen dieser Opcodes sind zusätzlich mehrere Adressierungsarten möglich.

Die meisten 65xx-Befehle (Opcodes) benötigen als zusätzliche Angabe Daten, mit denen der Befehl auszuführen ist. Diese Daten werden als Operand bezeichnet. Der Operand kann auch, frei wählbar, symbolisch angegeben sein. Man kann als Operand auch einen ASCII-Code angeben. Zur Kennzeichnung des ASCII-Codes muß dieser zwischen zwei Apostrophe gestellt werden (z.B. 'a').

Symbolische Adressen oder Werte werden als Label bezeichnet. Um eventuelle Ansprünge aus dem Programm zu einem Befehl kenntlich zu machen, kann auch vor dem Opcode linksbündig ein Label stehen.

Beispiel :

```
Label       Opcode      Operand

anfang      LDA         $90
            STA         abc
            LDA         'a'
...         ...         ...
ende        STA         zero90
```

2.2 Erklärung des Assemblerlistings

Das Assemblerlisting hat folgenden grundsätzlichen Aufbau:

Zeilennummer Hexadresse Objekt-Code Quell-Code Kommentar

2.2.1 Zeilennummern-Feld

Die Zeilennummer ist fünfstellig. Nach jeder Assemblerzeile wird die Zeilennummer um 1 erhöht. Sie ist im Zusammenhang mit der später beschriebenen Crossreferenzliste (Verweistabelle) sehr wichtig.

2.2.2 Hex-Adressen-Feld

An der nächsten Position in der Assemblerzeile steht die hexadezimale Adresse. Da der Prozessor nur 64 KByte direkt adressieren kann, genügt hier eine vierstellige Darstellung von $0000 bis $FFFF. Je nachdem, ob in der Zeile ein 1-Byte-, 2-Byte- oder 3-Byte-Befehl steht, wird die Adresse in der Folgezeile um 1, 2 oder 3 erhöht (inkrementiert).

2.2.3 Objekt-Code-Feld

Hier steht der vom Assembler erzeugte Maschinen-Code, der vom Prozessor direkt verarbeitet werden kann. Es ist hier Platz für drei Bytes reserviert. Stehen in der Zeile Deklarationen, dann ist dieser Platz frei, weil kein Maschinen-Code erzeugt wird.

2.2.4 Quell-Code-Feld

Der Quell-Code kann in verschiedener Form im Assemblerlisting erscheinen.

2.2.4.1 Deklaration und Zuweisung von Label

Wird im Assemblerlisting einem Label ein bestimmter Wert zugewiesen, dann erfolgt dies, ähnlich wie in BASIC, mit dem Gleichheits-Zeichen "=".

Beispiel: test = $85

In diesem Beispiel wird dem Label mit dem Namen "test" der hexadezimale Wert "85" zugewiesen. Wird nun im Listing z.B. das Label "test" als Operand verwendet, dann wird im Objekt-Code hierfür der Wert "85" eingesetzt.

Eine weitere Möglichkeit besteht mit "* = Adresse". Man kann hiermit problemlos ab diesem Zeitpunkt eine neue Programm-Adresse vorgeben.

Beispiel: * = $1234

Ab diesem Zeitpunkt wird die Programm-Adresse auf hexadezimal "$1234" gesetzt.

Beispiel: * = * + $100

War bei diesem Beispiel die Adresse z.B. "$F100", so wird nach dieser Zeile die Adresse auf "$F200" gesetzt.

2.2.4.2 Linksbündiges Label

Beginnt der Quell-Code mit einem Label (linksbündiges Label) und darauffolgendem Mnemonic, so wird diesem Label vom Assembler die in dieser Zeile aktuelle Adresse zugewiesen.

2.2.4.3 Mnemonics

Die Mnemonics werden aus Gründen der Übersichtlichkeit an einer eigenen Tabulatorposition im Listing ausgegeben. Es ist dadurch sehr leicht eine Unterscheidung vom linksbündigen Label möglich.

2.2.4.4 Assembler-Direktiven

Die Assembler-Direktiven veranlassen den Assembler, bestimmte Funktionen auszuführen. Die Direktiven stehen stets im Feld "Quell-Code" und beginnen mit einem Punkt.

.byte label	Der Assembler setzt im Objekt-Code den Wert des Labels (1 Byte) ein
.byte nn, nn	Der Assembler setzt im Objekt-Code die Werte der hinter der Direktive folgenden Bytes ein.
.byte 'abcd'	Der Assembler setzt im Objekt-Code die ASCII-Codes der Zeichen 'abcd' ein (4 Bytes)
.byte "abcd"	Der Assembler setzt im Objekt-Code die Bildschirm-Codes der Zeichen 'abcd' ein.
.byte <label	Der Assembler setzt im Objekt-Code den niederwertigen Teil der Adresse, = Low-Byte des Adress-Labels (1 Byte), ein

.byte >label Der Assembler setzt im Objekt-Code den höherwertigen Teil der Adresse, = High-Byte des Adress-Labels (1 Byte), ein

.word label Der Assembler setzt im Objekt-Code den Wert des Adress-Labels (2 Bytes) in der Reihenfolge Low-Byte, High-Byte ein

2.2.4.5 Operand

Der Operand kann als Label aber auch als Wert angegeben sein. Werte ohne Vorzeichen sind Dezimalwerte, Werte mit vorgestelltem "$" sind Hexwerte und Werte mit "%" sind Binär-Werte (8-stellig).

Der Operand kann auch berechnet werden, wie beispielsweise "test + $80". Es wird hierbei generell ohne Beachtung von Rechenregeln von links nach rechts gerechnet.

Als Operand können auch in Apostrophe eingeschlossene ASCII-Zeichen eingesetzt sein, bzw. Zeichen im Commodore-Bildschirm-Format, eingeschlossen in Anführungszeichen. Bei Immediate-Befehlen (der Operand beginnt mit einem "#") kann auch durch ein folgendes "<" das Low-Byte bzw. durch ein ">" das High-Byte einer Adresse im Objekt-Code eingesetzt werden.

2.2.5 Kommentar-Feld

Im Kommentar-Feld stehen Hinweise und Erklärungen, die sich auf diese Zeile beziehen. Reicht der Platz für die Kommentare nicht aus, dann wird in der nächsten Zeile im Kommentar-Feld fortgesetzt. Die Zeilennummer erhöht sich in solchen Fällen nicht zusätzlich.

2.3 Crossreferenzliste bzw. Verweistabelle

Anschließend an das Assemblerlisting folgt eine Crossreferenzliste, eingedeutscht auch Verweistabelle genannt. Sie enthält in alphabetischer Reihenfolge alle im Assemblerlisting vorkommenden Labelnamen (symbolische Zuordnung einer Adresse oder Wert), die Adresse als Hex-Zahl und darauf folgend, durch einen Doppelpunkt getrennt, die fünfstelligen Zeilennummern, in denen das Label vorkommt.

Mit Hilfe der Crossreferenzliste ist es beispielsweise jederzeit möglich festzustellen, ob und von wo eine bestimmte Unterroutine aus anderen Programmteilen angesprungen wird, bzw. wo eine bestimmte Speicherstelle definiert, benutzt oder verändert wird.

Eine zusätzliche Information bieten die Zeichen "#" und "$". Folgt direkt hinter einer Zeilennummer ein "#", dann wird in der angegebenen Zeile das Label definiert. Zur Erklärung ein Beispiel einer Assemblerzeile, in der ein Label definiert wird.

```
Zeile  Adr.  Obj.-Code  Quell-Code

00123  0000             flag = $CA   Ein/Ausgabe-Flag
```

Es wird hier dem Label "flag" der hexadezimale Wert "CA" zugewiesen.

Eine weitere Möglichkeit einer Label-Definition ist ein sogenanntes linksbündiges Label. Hier wird dem Label die gerade aktuelle Programm-Adresse als Wert zugewiesen. Auch hierzu ein Beispiel einer möglichen Assemblerzeile:

```
Zeile  Adr.  Obj.-Code  Quell-code

06065  F011     8A      beginn TXA   Inhalt von X-Reg. in Accu
```

In diesem Beispiel wird dem Label "beginn" der hexadezimale Wert "F011" der aktuellen Adresse zugewiesen. Wird nun irgendwo im Listing dieses Label verwendet, dann setzt der Assembler dafür den Wert "$F011" ein.

Folgt in der Crossreferenzliste direkt hinter der Zeilennummer ein "$", dann wird in dieser Zeile im Assemblerlisting der Inhalt der Speicherstelle, die durch das Label definiert ist, durch einen entsprechenden Prozessorbefehl geändert, z.B.:

```
Zeile  Adr.  Obj.-Code  Source-Code

06156  F169    85 55     STA zwert    Zwischenwert für Zähler
```

Es wird hier in die Speicherstelle mit der Adresse "$55" und dem Labelnamen "zwert" der Inhalt des Akkumulators eingeschrieben.

Die folgenden Zeilen zeigen im Beispiel den Aufbau der Crossreferenzliste bzw. Verweistabelle:

= Definition, $ = Schreibzugriff, leer = Lesezugriff/Sprung

```
Label     Adr.  Zeilennummern

beginn = $F011: 00955, 00971, 06065#, 06269
:
:
flag   =  $CA : 00123#, 00845, 0139$, 0180$, 04960
:
:
zwert  =  $55 : 00089#, 02790, 03531$, 03840, 06156$
```

2.3.1 Handhabungsbeispiel

Beim Durcharbeiten des Assemblerlistings ist es für Sie von großem Wert zu wissen, ob z.B. die Unterroutine "beginn" (linksbündiges Label) mehrfach verwendet wird. Dazu schlagen Sie in der nach Labelnamen sortierten Crossreferenzliste unter "beginn" nach. Dort finden Sie, wie unser Beispiel zeigt, den zugewiesenen Wert des Labels, hexadezimal "$F011", weiterhin die Zeilennummern und außerdem, durch ein "#" markiert, die Zeilennummer, in der das Label definiert wurde. In unserem Beispiel Zeile 06065.

3 Der C-128-Monitor

3.1 Kommentiertes Assemblerlisting

```
 Pass 1

00003  0000                    .lib mon.lbl
00094  0000                    .lib mon.m01
00757  b406                    .lib mon.m02
01394  b7a5                    .lib mon.m03

 Pass 2

00001  0000
00002  0000
00003  0000                    .lib mon.lbl
00004  0000
00005  0000
00006  0000  *** Konstanten
00007  0000
00008  0000            eom    = $00          Ende von Direkt-Meldungen
00009  0000            cr     = $0d          Carriage-Return
00010  0000            esc    = $1b          Escape
00011  0000            skip1  = $24          'Bit'-Befehl für Überspringen
                                              1-Byte-Befehle
00012  0000            skip2  = $2c          'Bit'-Befehl für Überspringen
                                              2-Byte-Befehle
00013  0000
00014  0000  *** Zero-Page Adressen
00015  0000
00016  0000            bank   = $02          Banknummer
00017  0000            pchi   = $03          8502 CPU PC hi
00018  0000            pclo   = $04          8502 CPU PC lo
00019  0000            sreg   = $05          8502 CPU sr
00020  0000            areg   = $06          8502 CPU ac
00021  0000            xreg   = $07          8502 CPU xr
00022  0000            yreg   = $08          8502 CPU yr
00023  0000            stkptr = $09          8502 CPU sp
00024  0000            to     = $60          addrl gelesene Adresse für
                                              Monitor-Befehl
00025  0000            t1     = $63          addrl Hilfszeiger Monitor
00026  0000            t2     = $66          addrl aktuelle Adresse
00027  0000            txtptr = $7a          Zeiger in aktuellen Monitor-Befehl
00028  0000            status = $90          I/O-Status 'st'
00029  0000            stkey  = $91          Flag für Stop- und RVS-Taste
00030  0000            verck  = $93          Flag für load/verify
00031  0000            ptr2   = $9f
00032  0000            fnlen  = $b7          Länge akt. Filename
00033  0000            sa     = $b9          akt. Sekundäradresse
00034  0000            fa     = $ba          akt. Gerätenummer
```

```
00035  0000            fnadr  = $bb        addrl akt. Filename
00036  0000            ba     = $c6        akt. Bank für Speicher-Operationen
00037  0000            fnbank = $c7        Bank des akt. Filenamens
00038  0000            ndx    = $d0        Bytes im Tastaturpuffer
00039  0000            mode   = $d7        BS-Modus 40/80-Zeichen
00040  0000
00041  0000  *** RAM-Adressen
00042  0000
00043  0000            stack  = $0100      8502 CPU-Stapel
00044  0000
00045  0000            buf    = $0200      BASIC-Eingabepuffer
00046  0000            stashx = $02b9      STAVEC
00047  0000            cmparx = $02c8      CMPVEC
00048  0000
00049  0000            imoncd = $032e      Monitor-Kommando ausf. ($b006)
00050  0000            keyd   = $034a      Tastaturpuffer
00051  0000
00052  0000            sysvec = $0a00      Vector (BASIC-) Warmstart ($4003)
00053  0000            xcnt   = $0a80      Puffer für Vergleiche und Dateinamen
00054  0000            hulp   = $0aa0      Monitor-Hilfsbereich
00055  0000            hulp2  = $0aa4      Monitor-Hilfsbereich 2
00056  0000            format = $0aaa      aktuelle Adressierungsart bei
                                            Ass./Dis-Ass.
00057  0000            length = $0aab      Länge des aktuellen Befehls bei
                                            Ass./Dis-Ass.
00058  0000            msal   = $0aac      Hilfsregister für Adressen
00059  0000            sxreg  = $0aaf      Rettungsbyte für X-Register
00060  0000            wrap   = $0ab1      Zwischenspeicher
00061  0000            xsave  = $0ab2      Rettungsbyte für X-Register
00062  0000            direct = $0ab3      Richtung für Block-Verschiebung
00063  0000            count  = $0ab4      Zwischenzähler für Block-Operationen
00064  0000            number = $0ab5      Hilfsadresse Monitor
00065  0000            shift  = $0ab6      Hilfszähler für Basis-Umrechnungen
00066  0000            temps  = $0ab7      Zwischenspeicher
00067  0000
00068  0000  *** Externe ROM-Adressen
00069  0000
00070  0000            mmucr  = $ff00      MMU 8722 Config. Reg. (Kopie)
00071  0000
00072  0000            kstbnk = $ff68      Bank für load/sav/verify setzen
                                            (Sprungv.)
00073  0000            kjsrfr = $ff6e      JSR in andere Bank, RTS in akt. Bank
                                            (Sprungv.)
00074  0000            kjmpfr = $ff71      JMP in andere Bank (Sprungv.)
00075  0000            kifetv = $ff74      LDA (FETVEC),y von Bank in xr
                                            (Sprungv.)
00076  0000            kistav = $ff77      STA (STAVEC),y in Bank in xr
                                            (Sprungv.)
00077  0000            kicmpv = $ff7a      CMP (CMPVEC),y mit Bank in xr
                                            (Sprungv.)
00078  0000            kprimm = $ff7d      folg. Text ausgeben und
                                            überspringen (Sprungv.)
00079  0000            kstmsg = $ff90      Flag setzen Ausgabe/Unterdr.
                                            OPSYS-Meldung (Sprungv.)
00080  0000            kstlfs = $ffba      la, fa, sa aus ac, xr, yr setzen
                                            (Sprungv.)
00081  0000            kstnam = $ffbd      fnlen, fnadr/+1 aus ac, xr, yr
                                            setzen (Sprungv.)
```

```
00082  0000             kopen  = $ffc0          log. File üb. Vector öffnen,
                                                Ausgabe fn (sprungv.)

00083  0000             kclose = $ffc3          log. File üb. Vector schließen
                                                (Sprungv.)
00084  0000             kchkin = $ffc6          Eingabekanal üb. Vector öffnen
                                                (Sprungv.)
00085  0000             kckout = $ffc9          Ausgabe-Kanal üb. Vector öffnen
                                                (Sprungv.)
00086  0000             kclrch = $ffcc          Ein-/Ausgabe-Kanäle üb. Vector
                                                schließen (Sprungv.)
00087  0000             kbasin = $ffcf          Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
00088  0000             kbsout = $ffd2          Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
00089  0000             kloads = $ffd5          load/verify (ac) ab Adresse in
                                                xr/yr (Sprungv.)
00090  0000             ksaves = $ffd8          save ab ac bis xr, yr (Sprungv.)
00091  0000             kstop  = $ffe1          Stopp-Taste üb. Vector abfragen
                                                (Sprungv.)
00092  0000
00093  0000                      .end
00094  0000                      .lib mon.m01
00095  0000
00096  0000
00097  0000                      *=$b000
00098  b000
00099  b000  4c 21 b0   moncal jmp call         Monitor-Einsprung über CALL
00100  b003  4c 09 b0   monbrk jmp break        Monitor-Einsprung über BREAK
00101  b006  4c b2 b0   moncmd jmp evalcm       Monitor-Kommando ausführen
00102  b009
00103  b009
00104  b009  ===> Monitor-Aufruf durch 'BRK'-Befehl <===
00105  b009
00106  b009  20 7d ff   break  jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
00107  b00c
00108  b00c  0d 42 52           .byte cr,'break',$07,eom
00109  b00f  45 41 4b
00110  b012  07 00
00111  b014
00112  b014  68                 pla             Bank vom Stapel holen
00113  b015  85 02              sta bank        Banknummer
00114  b017  a2 05              ldx #5          Anzahl Bytes
00115  b019
00116  b019  ===> Prozessor-Register vom Stapel lesen <===
00117  b019
00118  b019  68         break1  pla             Register vom Stapel holen
00119  b01a  95 03              sta pchi,x      8502 CPU PC hi
00120  b01c  ca                 dex
00121  b01d  10 fa              bpl break1
00122  b01f  30 25              bmi start       Monitor-Initialisierung
00123  b021
00124  b021
00125  b021  ===> Monitor-Aufruf durch 'Monitor' oder 'SYS'-Befehl <===
00126  b021
00127  b021  a9 00      call    lda #0          ROM's einschalten u. Register
                                                löschen
```

```
00128  b023  8d 00 ff          sta mmucr        MMU 8722 Config. Reg. (Kopie)
00129  b026  85 06             sta areg         8502 CPU ac
00130  b028  85 07             sta xreg         8502 CPU xr
00131  b02a  85 08             sta yreg         8502 CPU yr
00132  b02c  85 05             sta sreg         8502 CPU sr
00133  b02e  a9 00             lda #<moncal     Monitor-Einsprung über CALL
00134  b030  a0 b0             ldy #>moncal     Monitor-Einsprung über CALL
00135  b032  85 04             sta pclo         8502 CPU PC lo
00136  b034  84 03             sty pchi         8502 CPU PC hi
00137  b036  a9 0f             lda #15
00138  b038  85 02             sta bank         Banknummer
00139  b03a  20 7d ff          jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
00140  b03d
00141  b03d  0d 4d 4f          .byte cr,'monitor',eom
00142  b040  4e 49 54
00143  b043  4f 52 00
00144  b046
00145  b046
00146  b046  ===> Monitor-Initialisierung <===
00147  b046
00148  b046  d8         start  cld              Dezimalflag löschen
00149  b047  ba                tsx              Stapelzeiger retten
00150  b048  86 09             stx stkptr       8502 CPU sp
00151  b04a  a9 c0             lda #%11000000   Keine Fehlermeldungen zulassen
00152  b04c  20 90 ff          jsr kstmsg       Flag setzen Ausgabe/Unterdr.
                                                OPSYS-Meldung (Sprungv.)
00153  b04f  58                cli              Interrupt freigeben
00154  b050
00155  b050
00156  b050  ===> Register-Inhalte anzeigen <===
00157  b050
00158  b050  20 7d ff   dspreg jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
00159  b053
00160  b053  0d 20 20          .byte cr,'    pc  sr ac xr yr sp',cr
00161  b056  20 20 50
00162  b059  43 20 20
00163  b05c  53 52 20
00164  b05f  41 43 20
00165  b062  58 52 20
00166  b065  59 52 20
00167  b068  53 50 0d
00168  b06b  3b 20 1b          .byte '; ',esc,'q',eom
00169  b06e  51 00
00170  b070
00171  b070  a5 02             lda bank         Banknummer
00172  b072  20 d2 b8          jsr makhex       Byte in AC in Hexa umwandeln nach
                                                AC und XR
00173  b075  8a                txa              Low-Nybble der Bankadresse
00174  b076  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
00175  b079  a5 03             lda pchi         8502 CPU PC hi
00176  b07b  20 c2 b8          jsr puthex       Hex-Byte in AC als 2 ASCII-Zeichen
                                                ausgeben
00177  b07e  a0 02             ldy #2           Zeiger in Register-Kopien
00178  b080
00179  b080
```

```
00180  b080  ===> Inhalte der Register-Kopien als Hex-Bytes ausgeben <===
00181  b080
00182  b080  b9 02 00   dsprg2 lda bank,y       Prozessor-Register aus Speicher
00183  b083  20 a5 b8           jsr puthxs      Accu als Hex-Bytes ausgeben + SPACE
00184  b086  c8                 iny
00185  b087  c0 08              cpy #8
00186  b089  90 f5              bcc dsprg2      Inhalte der Register-Kopien als
                                                Hex-Bytes ausgeben
00187  b08b
00188  b08b
00189  b08b  ===> Monitor-Warteschleife <===
00190  b08b
00191  b08b  20 b4 b8   main    jsr crlf        Ausgabe 'carriage return'
00192  b08e  a2 00              ldx #0
00193  b090  86 7a              stx txtptr      Zeiger in aktuellen Monitor-Befehl
00194  b092  20 cf ff   main01  jsr kbasin      Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
00195  b095  9d 00 02           sta buf,x       BASIC-Eingabepuffer
00196  b098  e8                 inx
00197  b099  e0 a1              cpx #161        mehr als 160 Zeichen ?
00198  b09b  b0 1f              bcs error       Fehler ausgeben, falscher
                                                Monitorbefehl
00199  b09d  c9 0d              cmp #cr         Carriage Return ?
00200  b09f  d0 f1              bne main01      nein -> weiter einlesen
00201  b0a1  a9 00              lda #0          Puffer mit  Null abschließen
00202  b0a3  9d ff 01           sta buf-1,x     BASIC-Eingabepuffer
00203  b0a6
00204  b0a6
00205  b0a6  ===> Monitor-Befehl auswerten <===
00206  b0a6
00207  b0a6  20 e9 b8   main02  jsr gnc         Ein Zeichen aus Eingabe-Puffer
                                                holen
00208  b0a9  f0 e0              beq main        Monitor-Warteschleife
00209  b0ab  c9 20              cmp #' '        Space ?, überlesen
00210  b0ad  f0 f7              beq main02      Monitor-Befehl auswerten
00211  b0af  6c 2e 03           jmp (imoncd)    Monitor-Kommando ausführen ($b006)
00212  b0b2
00213  b0b2
00214  b0b2  ===> Befehl in Tabelle suchen <===
00215  b0b2
00216  b0b2  a2 15      evalcm  ldx #21         Anzahl Monitorbefehle - 1
00217  b0b4  dd e6 b0   evalc1  cmp cmdchr,x    Tabelle der Monitor-Befehle
00218  b0b7  f0 0c              beq main1       Befehl gefunden, Adresse auf Stapel
                                                legen
00219  b0b9  ca                 dex
00220  b0ba  10 f8              bpl evalc1
00221  b0bc
00222  b0bc
00223  b0bc  ===> Fehler ausgeben, falscher Monitorbefehl <===
00224  b0bc
00225  b0bc  20 7d ff   error   jsr kprimm      folg. Text ausgeben und
                                                überspringen (Sprungv.)
00226  b0bf
00227  b0bf  1d 3f 00           .byte $1d,'?',eom      Cursor rechts + '?'
00228  b0c2
00229  b0c2  4c 8b b0           jmp main        Monitor-Warteschleife
00230  b0c5
00231  b0c5
```

```
00232  b0c5  ===> Befehl gefunden, Adresse auf Stapel legen <===
00233  b0c5
00234  b0c5  e0 13      main1  cpx #19          Floppy-Operation ?
00235  b0c7  b0 12             bcs main11       ja -> getrennt ausführen
00236  b0c9  e0 0f             cpx #15          Basis-Umrechnung ?
00237  b0cb  b0 13             bcs main12       ja -> getrennt ausführen
00238  b0cd  8a                txa              Zeiger in Befehlstabelle
00239  b0ce  0a                asl a            mit zwei multiplizieren
00240  b0cf  aa                tax              = Zeiger in Adresstabelle
00241  b0d0  bd fd b0          lda cmdtbl+1,x   Tabelle der Anfangsadressen der
                                                Befehls-Routinen
00242  b0d3  48                pha              High-Byte auf Stapel legen
00243  b0d4  bd fc b0          lda cmdtbl,x     Tabelle der Anfangsadressen der
                                                Befehls-Routinen
00244  b0d7  48                pha              Low-Byte auf Stapel legen
00245  b0d8  4c a7 b7          jmp parse        Operanden ermitteln u. auf
                                                gült.Trennnzeichen prüfen
00246  b0db
00247  b0db  85 93      main11 sta verck        Flag für load/verify
00248  b0dd  4c 37 b3          jmp lodsav       Ausführung 'Load/Save/Verify'
00249  b0e0
00250  b0e0  4c b1 b9   main12 jmp convrt       Basis-Umrechnungsbefehle ausführen
00251  b0e3
00252  b0e3  6c 00 0a   exit   jmp (sysvec)     Vector (BASIC-) Warmstart ($4003)
00253  b0e6
00254  b0e6
00255  b0e6  ===> Tabelle der Monitor-Befehle <===
00256  b0e6
00257  b0e6  41 43 44   cmdchr .byte 'acdfghjmrtx@.>;'
00258  b0e9  46 47 48
00259  b0ec  4a 4d 52
00260  b0ef  54 58 40
00261  b0f2  2e 3e 3b
00262  b0f5
00263  b0f5
00264  b0f5  ===> Tabelle der Umrechnungs-Befehle <===
00265  b0f5
00266  b0f5  24 2b 26   cmdnum .byte '$+&%'
00267  b0f8  25
00268  b0f9
00269  b0f9  4c 53 56           .byte 'lsv'
00270  b0fc
00271  b0fc
00272  b0fc  ===> Tabelle der Anfangsadressen der Befehls-Routinen <===
00273  b0fc
00274  b0fc  05 b4      cmdtbl .word assem-1    Direkt-Assembler aufrufen
00275  b0fe  30 b2             .word compar-1   Speicherbereiche vergleichen
00276  b100  98 b5             .word disasm-1   Speicherbereich Dis-Assemblieren
00277  b102  da b3             .word fill-1     Speicherbereich mit Konstante
                                                füllen
00278  b104  d5 b1             .word go-1       Maschinenprogramm aufrufen
00279  b106  cd b2             .word hunt-1     Speicherbereich nach Bytes
                                                durchsuchen
00280  b108  de b1             .word gosub-1    Programm in anderer Speicherbank
                                                ausführen
00281  b10a  51 b1             .word dspmem-1   Speicherbereich Hex u. ASCII
                                                anzeigen
00282  b10c  4f b0             .word dspreg-1   Register-Inhalte anzeigen
```

```
00283  b10e  33 b2           .word trnsfr-1   Speicherbereich verschieben
00284  b110  e2 b0           .word exit-1     Monitor verlassen, zurück zu BASIC
00285  b112  8f ba           .word disk-1     Floppy-Kommando übergeben
00286  b114  05 b4           .word assem-1    Direkt-Assembler aufrufen
00287  b116  aa b1           .word setmem-1   Byte-Werte in Speicher übernehmen
00288  b118  93 b1           .word setreg-1   Register-Inhalte übernehmen
00289  b11a
00290  b11a
00291  b11a  ===> Ein Byte aus beliebiger Bank lesen <===
00292  b11a
00293  b11a  8e b2 0a   fetch  stx xsave       Rettungsbyte für X-Register
00294  b11d  a6 68             ldx t2+2        bank aktuelle Adresse
00295  b11f  a9 66             lda #t2         addrl aktuelle Adresse
00296  b121  78                sei
00297  b122  20 74 ff          jsr kifetv      LDA (FETVEC),y von Bank in xr
                                               (Sprungv.)
00298  b125  58                cli
00299  b126  ae b2 0a          ldx xsave       Rettungsbyte für X-Register
00300  b129  60                rts
00301  b12a
00302  b12a
00303  b12a  ===> Ein Byte in beliebige Bank schreiben <===
00304  b12a
00305  b12a  8e b2 0a   stash  stx xsave       Rettungsbyte für X-Register
00306  b12d  a2 66             ldx #t2         addrl aktuelle Adresse
00307  b12f  8e b9 02          stx stashx      STAVEC
00308  b132  a6 68             ldx t2+2        bank aktuelle Adresse
00309  b134  78                sei
00310  b135  20 77 ff          jsr kistav      STA (STAVEC),y in Bank in xr
                                               (Sprungv.)
00311  b138  58                cli
00312  b139  ae b2 0a          ldx xsave       Rettungsbyte für X-Register
00313  b13c  60                rts
00314  b13d
00315  b13d
00316  b13d  ===> Byte in Accu mit beliebiger Bank vergleichen <===
00317  b13d
00318  b13d  8e b2 0a   cmpare stx xsave       Rettungsbyte für X-Register
00319  b140  a2 66             ldx #t2         addrl aktuelle Adresse
00320  b142  8e c8 02          stx cmparx      CMPVEC
00321  b145  a6 68             ldx t2+2        bank aktuelle Adresse
00322  b147  78                sei
00323  b148  20 7a ff          jsr kicmpv      CMP (CMPVEC),y mit Bank in xr
                                               (Sprungv.)
00324  b14b  58                cli
00325  b14c  08                php
00326  b14d  ae b2 0a          ldx xsave       Rettungsbyte für X-Register
00327  b150  28                plp
00328  b151  60                rts
00329  b152
00330  b152
00331  b152  ===> Speicherbereich Hex u. ASCII anzeigen <===
00332  b152
00333  b152  b0 08      dspmem bcs dspme1      Keine End-Adresse -> Standardwerte
                                               setzen
00334  b154  20 01 b9          jsr totot2      Zeiger kopieren
00335  b157  20 a7 b7          jsr parse       Operanden ermitteln u. auf
                                               gült.Trennnzeichen prüfen
```

```
00336  b15a  90 06           bcc dspme2
00337  b15c  a9 0b     dspme1 lda #11          11 Bildschirmzeilen sind Standard

00338  b15e  85 60           sta to            addrl gelesene Adresse für
                                                Monitor-Befehl
00339  b160  d0 15           bne dspme4        Prüfen auf Stop-Taste, eine Zeile
                                                ausgeben
00340  b162
00341  b162  ===> Anzahl Schirmzeilen aus Anfang- u. Endadresse bilden <===
00342  b162
00343  b162  20 0e b9  dspme2 jsr subom2       Zeiger von aktueller Adresse
                                                subtrahieren
00344  b165  90 2a           bcc dsperr        Fehler, da Endadresse <
                                                Anfangsadresse
00345  b167  a2 03           ldx #3            Durch 8 teilen bei 40 Zeichen/Zeile
00346  b169  24 d7           bit mode          BS-Modus 40/80-Zeichen
00347  b16b  10 01           bpl dspme3        40 Zeichen -> weiter
00348  b16d  e8              inx               Durch 16 teilen bei 80
                                                Zeichen/Zeile
00349  b16e
00350  b16e  ===> Wert 'X'-mal durch zwei teilen <===
00351  b16e
00352  b16e  46 62      dspme3 lsr to+2         bank gelesene Adresse für
                                                Monitor-Befehl
00353  b170  66 61           ror to+1          addrh gelesene Adresse für
                                                Monitor-Befehl
00354  b172  66 60           ror to            addrl gelesene Adresse für
                                                Monitor-Befehl
00355  b174  ca              dex
00356  b175  d0 f7           bne dspme3
00357  b177
00358  b177
00359  b177  ===> Prüfen auf Stop-Taste, eine Zeile ausgeben <===
00360  b177
00361  b177  20 e1 ff  dspme4 jsr kstop        Stop-Taste üb. Vector abfragen
                                                (Sprungv.)
00362  b17a  f0 12           beq shifts        Stop-Taste gedrückt, Abbruch
00363  b17c  20 e8 b1        jsr dmpone        Eine Zeile als Hex-Dump anzeigen
00364  b17f  a9 08           lda #8            8 Bytes Offset bei 40 Zeichen/Zeile
00365  b181  24 d7           bit mode          BS-Modus 40/80-Zeichen
00366  b183  10 01           bpl dspme5        Adresse erhöhen, auf Ende Prüfen
00367  b185  0a              asl a             16 Bytes Offset bei 80
                                                Zeichen/Zeile
00368  b186
00369  b186
00370  b186  ===> Adresse erhöhen, auf Ende Prüfen <===
00371  b186
00372  b186  20 52 b9  dspme5 jsr addt2        Zeiger um AC erhöhen
00373  b189  20 22 b9        jsr decto         Anzahl Schirmzeilen - 1
00374  b18c  b0 e9           bcs dspme4        Prüfen auf Stop-Taste, eine Zeile
                                                ausgeben
00375  b18e  4c 8b b0  shifts jmp main         Monitor-Warteschleife
00376  b191
00377  b191  4c bc b0  dsperr jmp error        Fehlerbehandlung
00378  b194
00379  b194
00380  b194  ===> Register-Inhalte übernehmen <===
00381  b194
```

```
00382  b194  20 74 b9   setreg jsr totopc       Adresse in Prozessor-Register
                                                kopieren
00383  b197  a0 00             ldy #0           Zeiger in Register-Kopien
00384  b199
00385  b199
00386  b199  ===> Ein Byte prüfen und in Register-Kopie speichern <===
00387  b199
00388  b199  20 a7 b7   setrg1 jsr parse        Operanden ermitteln u. auf
                                                gült.Trennnzeichen prüfen
00389  b19c  b0 0a             bcs setrg2       Fehlerhafter Wert, Abbruch
00390  b19e  a5 60             lda to           addrl gelesene Adresse für
                                                Monitor-Befehl
00391  b1a0  99 05 00          sta sreg,y       Register-Kopie schreiben
00392  b1a3  c8                iny              Zeiger erhöhen
00393  b1a4  c0 05             cpy #5           Alle Werte gelesen ?
00394  b1a6  90 f1             bcc setrg1       Ein Byte prüfen und in
                                                Register-Kopie speichern
00395  b1a8  4c 8b b0   setrg2 jmp main         Monitor-Warteschleife
00396  b1ab
00397  b1ab
00398  b1ab  ===> Byte-Werte in Speicher übernehmen <===
00399  b1ab
00400  b1ab  b0 1c      setmem bcs setme3       Keine gültige Adresse, Abbruch
00401  b1ad  20 01 b9          jsr totot2       Zeiger kopieren
00402  b1b0  a0 00             ldy #0           Zeiger in RAM
00403  b1b2
00404  b1b2
00405  b1b2  ===> Ein Byte prüfen und in Speicher übernehmen <===
00406  b1b2
00407  b1b2  20 a7 b7   setme1 jsr parse        Operanden ermitteln u. auf
                                                gült. Trennnzeichen prüfen
00408  b1b5  b0 12             bcs setme3       Fehler, nur eine Zeile ausgeben
00409  b1b7  a5 60             lda to           addrl gelesene Adresse für
                                                Monitor-Befehl
00410  b1b9  20 2a b1          jsr stash        Ein Byte in beliebige Bank
                                                schreiben
00411  b1bc  c8                iny              Zeiger erhöhen
00412  b1bd  24 d7             bit mode         BS-Modus 40/80-Zeichen
00413  b1bf  10 04             bpl setme2       40 Zeichen/Zeile -> weiter
00414  b1c1  c0 10             cpy #16          16 Bytes bei 80 Zeichen/Zeile
00415  b1c3  90 ed             bcc setme1       Ein Byte prüfen und in Speicher
                                                übernehmen
00416  b1c5  c0 08      setme2 cpy #8           8 Bytes bei 40 Zeichen/Zeile
00417  b1c7  90 e9             bcc setme1       Ein Byte prüfen und in Speicher
                                                übernehmen
00418  b1c9  20 7d ff   setme3 jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
00419  b1cc
00420  b1cc  1b 4f 91          .byte esc,'o',$91,eom
00421  b1cf  00
00422  b1d0
00423  b1d0  20 e8 b1          jsr dmpone       Eine Zeile als Hex-Dump anzeigen
00424  b1d3  4c 8b b0          jmp main         Monitor-Warteschleife
00425  b1d6
00426  b1d6
00427  b1d6  ===> Maschinenprogramm aufrufen <===
00428  b1d6
```

```
00429  b1d6  20 74 b9   go      jsr totopc       Adresse in Prozessor-Register
                                                 kopieren
00430  b1d9  a6 09              ldx stkptr       8502 CPU sp
00431  b1db  9a                 txs              Stapelzeiger setzen
00432  b1dc  4c 71 ff           jmp kjmpfr       JMP in andere Bank (Sprungv.)
00433  b1df
00434  b1df
00435  b1df  ===> Programm in anderer Speicherbank ausführen <===
00436  b1df
00437  b1df  20 74 b9   gosub   jsr totopc       Adresse in Prozessor-Register
                                                 kopieren
00438  b1e2  20 6e ff           jsr kjsrfr       JSR in andere Bank, RTS in akt.
                                                 Bank (Sprungv.)
00439  b1e5  4c 8b b0           jmp main         Monitor-Warteschleife
00440  b1e8
00441  b1e8
00442  b1e8  ===> Eine Zeile als Hex-Dump anzeigen <===
00443  b1e8
00444  b1e8  20 b4 b8   dmpone  jsr crlf         Ausgabe 'carriage return'
00445  b1eb  a9 3e              lda #'>'
00446  b1ed  20 d2 ff           jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                 Kanal (Sprungv.)
00447  b1f0  20 92 b8           jsr putt2        Adresse als Hex-Bytes ausgeben
00448  b1f3  a0 00              ldy #0           Zeiger in RAM
00449  b1f5  f0 03              beq dmpon2       Ein Byte als Hex-Byte ausgeben
00450  b1f7  20 a8 b8   dmpon1  jsr putspc       Ausgabe 'space'
00451  b1fa
00452  b1fa
00453  b1fa  ===> Ein Byte als Hex-Byte ausgeben <===
00454  b1fa
00455  b1fa  20 1a b1   dmpon2  jsr fetch        Ein Byte aus beliebiger Bank lesen
00456  b1fd  20 c2 b8           jsr puthex       Hex-Byte in AC als 2 ASCII-Zeichen
                                                 ausgeben
00457  b200  c8                 iny              Zeiger erhöhen
00458  b201  c0 08              cpy #8           8 Bytes bei 40 Zeichen/Zeile
00459  b203  24 d7              bit mode         BS-Modus 40/80-Zeichen
00460  b205  10 02              bpl dmpon3       40 Zeichen/Zeile -> weiter
00461  b207  c0 10              cpy #16          16 Bytes bei 80 Zeichen/Zeile
00462  b209  90 ec      dmpon3  bcc dmpon1       nächstes Byte ausgeben
00463  b20b  20 7d ff           jsr kprimm       folg. Text ausgeben und
                                                 überspringen (Sprungv.)
00464  b20e
00465  b20e  3a 12 00           .byte ':',$12,eom
00466  b211
00467  b211  a0 00              ldy #0           Zeiger in RAM
00468  b213
00469  b213
00470  b213  ===> ASCII-Zeichen ausgeben <===
00471  b213
00472  b213  20 1a b1   dmpasc  jsr fetch        Ein Byte aus beliebiger Bank lesen
00473  b216  48                 pha              Zeichen retten
00474  b217  29 7f              and #$7f         Auf Kleinbuchstaben reduzieren
00475  b219  c9 20              cmp #$20         Test auf Steuerzeichen
00476  b21b  68                 pla              Zeichen vom Stapel zurück
00477  b21c  b0 02              bcs dmpas1       Kein Steuerzeichen -> ausgeben
00478  b21e  a9 2e              lda #'.'         Steuerzeichen durch Punkt ersetzen
00479  b220  20 d2 ff   dmpas1  jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                 Kanal (Sprungv.)
```

```
00480  b223  c8              iny             Zeiger erhöhen
00481  b224  24 d7           bit mode        BS-Modus 40/80-Zeichen
00482  b226  10 04           bpl dmpas2      40 Zeichen/Zeile -> weiter
00483  b228  c0 10           cpy #16         16 Bytes bei 80 Zeichen/Zeile
00484  b22a  90 e7           bcc dmpasc      ASCII-Zeichen ausgeben
00485  b22c  c0 08     dmpas2 cpy #8          8 Bytes bei 40 Zeichen/Zeile
00486  b22e  90 e3           bcc dmpasc      ASCII-Zeichen ausgeben
00487  b230  60              rts
00488  b231
00489  b231
00490  b231  ===> Speicherbereiche vergleichen <===
00491  b231
00492  b231  a9 00     compar lda #0          Flag für Vergleichen
00493  b233  2c              .byte skip2
00494  b234
00495  b234
00496  b234  ===> Speicherbereich verschieben <===
00497  b234
00498  b234  a9 80     trnsfr lda #$80        Flag für Verschieben
00499  b236  85 93           sta verck       Flag für load/verify
00500  b238  a9 00           lda #0
00501  b23a  8d b3 0a        sta direct      Richtung für Block-Verschiebeung
00502  b23d  20 83 b9        jsr range       Adresse und Blocklänge holen
00503  b240  b0 05           bcs trns01      Fehler -> Abbruch
00504  b242  20 a7 b7        jsr parse       Operanden ermitteln u. auf
                                             gült.Trennnzeichen prüfen
00505  b245  90 03           bcc trns02      Ausführen Verschieben/Vergleichen
00506  b247  4c bc b0  trns01 jmp error      Fehler ausgeben, falscher
                                             Monitorbefehl
00507  b24a
00508  b24a
00509  b24a  ===> Ausführen Verschieben/Vergleichen <===
00510  b24a
00511  b24a  24 93     trns02 bit verck       Flag für Vergleichen/Verschieben
00512  b24c  10 2c           bpl trns04      Vergleichen -> weiter
00513  b24e  38              sec             Anzahl Bytes errechnen
00514  b24f  a5 66           lda t2          addrl aktuelle Adresse
00515  b251  e5 60           sbc to          addrl gelesene Adresse für
                                             Monitor-Befehl
00516  b253  a5 67           lda t2+1        addrh aktuelle Adresse
00517  b255  e5 61           sbc to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
00518  b257  b0 21           bcs trns04      Vorwärts-Verschiebung -> weiter
00519  b259  a5 63           lda t1          addrl Hilfszeiger Monitor
00520  b25b  65 60           adc to          addrl gelesene Adresse für
                                             Monitor-Befehl
00521  b25d  85 60           sta to          addrl gelesene Adresse für
                                             Monitor-Befehl
00522  b25f  a5 64           lda t1+1        addrh Hilfzeiger Monitor
00523  b261  65 61           adc to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
00524  b263  85 61           sta to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
00525  b265  a5 65           lda t1+2        bank Hilfszeiger Monitor
00526  b267  65 62           adc to+2        bank gelesene Adresse für
                                             Monitor-Befehl
00527  b269  85 62           sta to+2        bank gelesene Adresse für
                                             Monitor-Befehl
```

```
00528  b26b  a2 02           ldx #2          Zeiger für Rückwärtsverschiebung
                                             setzen
00529  b26d  bd b7 0a  trns03 lda temps,x     Zwischenspeicher
00530  b270  95 66           sta t2,x        addrl aktuelle Adresse
00531  b272  ca              dex
00532  b273  10 f8           bpl trns03
00533  b275  a9 80           lda #$80        Flag für Rückwärtsverschiebung
00534  b277  8d b3 0a        sta direct      Richtung für Block-Verschiebung
00535  b27a  20 b4 b8  trns04 jsr crlf        Ausgabe 'carriage return'
00536  b27d  a0 00           ldy #0          Zeiger in RAM
00537  b27f
00538  b27f
00539  b27f  ===> Ein Byte Kopieren/Vergleichen <===
00540  b27f
00541  b27f  20 e1 ff  trns05 jsr kstop       Stop-Taste üb. Vector abfragen
                                             (Sprungv.)
00542  b282  f0 47           beq trns12      Stop-Taste gedrückt, Abbruch
00543  b284  20 1a b1        jsr fetch       Ein Byte aus beliebiger Bank lesen
00544  b287  a2 60           ldx #to         addrl gelesene Adresse für
                                             Monitor-Befehl
00545  b289  8e b9 02        stx stashx      STAVEC
00546  b28c  8e c8 02        stx cmparx      CMPVEC
00547  b28f  a6 62     trns06 ldx to+2       bank gelesene Adresse für
                                             Monitor-Befehl
00548  b291  78              sei
00549  b292  24 93           bit verck       Flag für Vergleichen/Verschieben
00550  b294  10 03           bpl trns07      Vergleich durchführen, Abweichung
                                             anzeigen
00551  b296  20 77 ff        jsr kistav      STA (STAVEC),y in Bank in xr
                                             (Sprungv.)
00552  b299
00553  b299
00554  b299  ===> Vergleich durchführen, Abweichung anzeigen <===
00555  b299
00556  b299  a6 62     trns07 ldx to+2       bank gelesene Adresse für
                                             Monitor-Befehl
00557  b29b  20 7a ff         jsr kicmpv      CMP (CMPVEC),y mit Bank in xr
                                             (Sprungv.)
00558  b29e  58               cli
00559  b29f  f0 09            beq trns08      Bytes sind gleich, weiter ->
00560  b2a1  20 92 b8         jsr putt2       Adresse als Hex-Bytes ausgeben
00561  b2a4  20 a8 b8         jsr putspc      Ausgabe 'space'
00562  b2a7  20 a8 b8         jsr putspc      Ausgabe 'space'
00563  b2aa  2c b3 0a  trns08 bit direct      Richtung für Block-Verschiebung
00564  b2ad  30 0b            bmi trns09      Adresse bei Rückwärtskopierung
                                             vermindern
00565  b2af  e6 60            inc to          addrl gelesene Adresse für
                                             Monitor-Befehl
00566  b2b1  d0 10            bne trns10
00567  b2b3  e6 61            inc to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
00568  b2b5  d0 0c            bne trns10
00569  b2b7  4c bc b0         jmp error       Fehler ausgeben, falscher
                                             Monitorbefehl
00570  b2ba
00571  b2ba
00572  b2ba  ===> Adresse bei Rückwärtskopierung vermindern <===
00573  b2ba
```

```
00574  b2ba  20 22 b9   trns09 jsr decto        Adresse um 1 vermindern
00575  b2bd  20 60 b9          jsr dect2        Aktuelle Adresse um 1 vermindern
00576  b2c0  4c c6 b2          jmp trns11       Anzahl Bytes vermindern
00577  b2c3
00578  b2c3  20 50 b9   trns10 jsr inct2        Zeiger um 1 erhöhen
00579  b2c6
00580  b2c6
00581  b2c6  ===> Anzahl Bytes vermindern <===
00582  b2c6
00583  b2c6  20 3c b9   trns11 jsr dect1        Hilfszeiger um 1 vermindern
00584  b2c9  b0 b4             bcs trns05       Ein Byte Kopieren/Vergleichen
00585  b2cb  4c 8b b0   trns12 jmp main         Monitor-Warteschleife
00586  b2ce
00587  b2ce
00588  b2ce  ===> Speicherbereich nach Bytes durchsuchen <===
00589  b2ce
00590  b2ce  20 83 b9   hunt   jsr range        Adresse und Blocklänge holen
00591  b2d1  b0 61             bcs hunter       Fehlerbehandlung
00592  b2d3  a0 00             ldy #0
00593  b2d5  20 e9 b8          jsr gnc          Ein Zeichen aus Eingabe-Puffer
                                                holen
00594  b2d8  c9 27             cmp #'''         Apostroph, ASCII-Vergleich ?
00595  b2da  d0 16             bne hunt2        Nein, -> Einzelbytes holen
00596  b2dc  20 e9 b8          jsr gnc          Ein Zeichen aus Eingabe-Puffer
                                                holen
00597  b2df  c9 00             cmp #0           Zeichen vorhanden ?
00598  b2e1  f0 51             beq hunter       Nein, -> Fehlerbehandlung
00599  b2e3
00600  b2e3
00601  b2e3  ===> Zu suchende Bytes in Zwischenspeicher übernehmen <===
00602  b2e3
00603  b2e3  99 80 0a   hunt1  sta xcnt,y       Puffer für Vergleiche und
                                                Dateinamen
00604  b2e6  c8                iny
00605  b2e7  20 e9 b8          jsr gnc          Ein Zeichen aus Eingabe-Puffer
                                                holen
00606  b2ea  f0 1b             beq hunt4        Ende-Zeichen ->
00607  b2ec  c0 20             cpy #32          mehr als 32 Zeichen ?
00608  b2ee  d0 f3             bne hunt1        Zu suchende Bytes in
                                                Zwischenspeicher übernehmen
00609  b2f0  f0 15             beq hunt4        -> Vergleich beginnen
00610  b2f2  8c 00 01   hunt2  sty stack        8502 CPU-Stapel
00611  b2f5  20 a5 b7          jsr pargot       Zeiger rücksetzen, Operanden
                                                ermitteln
00612  b2f8  a5 60      hunt3  lda to           addrl gelesene Adresse für
                                                Monitor-Befehl
00613  b2fa  99 80 0a          sta xcnt,y       Puffer für Vergleiche und
                                                Dateinamen
00614  b2fd  c8                iny
00615  b2fe  20 a7 b7          jsr parse        Operanden ermitteln u. auf
                                                gült.Trennnzeichen prüfen
00616  b301  b0 04             bcs hunt4        Fehlerhaftes Byte, Vergleich
                                                beginnen
00617  b303  c0 20             cpy #32          Mehr als 32 Bytes ?
00618  b305  d0 f1             bne hunt3        Nein -> Weiter einlesen
00619  b307  84 93      hunt4  sty verck        Anzahl Bytes merken
00620  b309  20 b4 b8          jsr crlf         Ausgabe 'carriage return'
00621  b30c
```

```
00622  b30c
00623  b30c  ===> Vergleich Speicher mit gesuchten Bytes <===
00624  b30c
00625  b30c  a0 00      hunt5  ldy #0          Zeiger in RAM
00626  b30e
00627  b30e
00628  b30e  ===> Byte aus Speicher lesen <===
00629  b30e
00630  b30e  20 1a b1   hunt6  jsr fetch       Ein Byte aus beliebiger Bank lesen
00631  b311  d9 80 0a          cmp xcnt,y      Vergleichen mit gesuchten Bytes
00632  b314  d0 0e             bne hunt7       Nicht gleich, -> weiter
00633  b316  c8                iny             Zeiger erhöhen
00634  b317  c4 93             cpy verck       Anzahl Bytes erreicht ?
00635  b319  d0 f3             bne hunt6       Nein -> weitervergleichen
00636  b31b  20 92 b8          jsr putt2       Adresse als Hex-Bytes ausgeben
00637  b31e  20 a8 b8          jsr putspc      Ausgabe 'space'
00638  b321  20 a8 b8          jsr putspc      Ausgabe 'space'
00639  b324
00640  b324
00641  b324  ===> Prüfen auf Stop-Taste, Adressen erhöhen <===
00642  b324
00643  b324  20 e1 ff   hunt7  jsr kstop       Stop-Taste üb. Vector abfragen
                                               (Sprungv.)
00644  b327  f0 08             beq hunt8       Stop gedrückt, Abbruch
00645  b329  20 50 b9          jsr inct2       Zeiger um 1 erhöhen
00646  b32c  20 3c b9          jsr dect1       Hilfszeiger um 1 vermindern
00647  b32f  b0 db             bcs hunt5       Vergleich Speicher mit gesuchten
                                               Bytes
00648  b331  4c 8b b0   hunt8  jmp main        Monitor-Warteschleife
00649  b334
00650  b334  4c bc b0   hunter jmp error       Fehlerbehandlung
00651  b337
00652  b337
00653  b337
00654  b337  ===> Ausführung 'Load/Save/Verify' <===
00655  b337
00656  b337  a0 01      lodsav ldy #1          Standardwerte setzen
00657  b339  84 ba             sty fa          akt. Gerätenummer
00658  b33b  84 b9             sty sa          akt. Sekundäradresse
00659  b33d  88                dey
00660  b33e  84 c6             sty ba          akt. Bank für Speicher-Operationen
00661  b340  84 b7             sty fnlen       Länge akt. Filename
00662  b342  84 c7             sty fnbank      Bank des akt. Filenamens
00663  b344  84 90             sty status      I/O-Status 'st'
00664  b346  a9 0a             lda #>xcnt      Puffer für Vergleiche und
                                               Dateinamen
00665  b348  85 bc             sta fnadr+1     addrh akt. Filename
00666  b34a  a9 80             lda #<xcnt      Puffer für Vergleiche und
                                               Dateinamen
00667  b34c  85 bb             sta fnadr       addrl akt. Filename
00668  b34e
00669  b34e
00670  b34e  ===> Dateinamen auslesen <===
00671  b34e
00672  b34e  20 e9 b8   lodsv1 jsr gnc         Ein Zeichen aus Eingabe-Puffer
                                               holen
00673  b351  f0 58             beq lodsv6      Ausführung 'Load/Verify'
00674  b353  c9 20             cmp #' '        Space ?
```

```
00675  b355  f0 f7           beq lodsv1      überlesen ->
00676  b357  c9 22           cmp #'"'        Anführungszeichen ?
00677  b359  d0 15           bne lodsv3      Nein -> Dateiname fehlt, Fehler ->
00678  b35b  a6 7a           ldx txtptr      Zeiger in aktuellen Monitor-Befehl
00679  b35d  bd 00 02 lodsv2 lda buf,x       BASIC-Eingabepuffer
00680  b360  f0 49           beq lodsv6      Ausführung 'Load/Verify'
00681  b362  e8              inx
00682  b363  c9 22           cmp #'"'        Anführungszeichen ?
00683  b365  f0 0c           beq lodsv4      Gerätenummer ermitteln und auf
                                             Gültigkeit prüfen
00684  b367  91 bb           sta (fnadr),y   addrl akt. Filename
00685  b369  e6 b7           inc fnlen       Länge akt. Filename
00686  b36b  c8              iny
00687  b36c  c0 11           cpy #17         Mehr als 17 Zeichen ?
00688  b36e  90 ed           bcc lodsv2      Nein -> weitereinlesen
00689  b370  4c bc b0  lodsv3 jmp error      Fehler ausgeben, falscher
                                             Monitorbefehl
00690  b373
00691  b373
00692  b373  ===> Gerätenummer ermitteln und auf Gültigkeit prüfen <===
00693  b373
00694  b373  86 7a      lodsv4 stx txtptr    Zeiger in aktuellen Monitor-Befehl
00695  b375  20 e9 b8          jsr gnc       Ein Zeichen aus Eingabe-Puffer
                                             holen
00696  b378  f0 31           beq lodsv6      Ausführung 'Load/Verify'
00697  b37a  20 a7 b7          jsr parse     Operanden ermitteln u. auf
                                             gült.Trennnzeichen prüfen
00698  b37d  b0 2c           bcs lodsv6      Ausführung 'Load/Verify'
00699  b37f  a5 60           lda to          addrl gelesene Adresse für
                                             Monitor-Befehl
00700  b381  85 ba           sta fa          akt. Gerätenummer
00701  b383  20 a7 b7          jsr parse     Operanden ermitteln u. auf
                                             gült.Trennnzeichen prüfen
00702  b386  b0 23           bcs lodsv6      Ausführung 'Load/Verify'
00703  b388  20 01 b9          jsr totot2    Zeiger kopieren
00704  b38b  85 c6           sta ba          akt. Bank für Speicher-Operationen
00705  b38d  20 a7 b7          jsr parse     Operanden ermitteln u. auf
                                             gült.Trennnzeichen prüfen
00706  b390  b0 3f           bcs lodsv8      'Load' ohne Adress-Angabe
                                             durchführen
00707  b392  20 b4 b8          jsr crlf      Ausgabe 'carriage return'
00708  b395  a6 60           ldx to          addrl gelesene Adresse für
                                             Monitor-Befehl
00709  b397  a4 61           ldy to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
00710  b399  a5 93           lda verck       Flag für load/verify
00711  b39b  c9 53           cmp #'s'        Sichern ?
00712  b39d  d0 d1           bne lodsv3      Nein -> Bearbeiten Load/Verify
00713  b39f  a9 00           lda #0
00714  b3a1  85 b9           sta sa          akt. Sekundäradresse
00715  b3a3  a9 66           lda #t2         addrl aktuelle Adresse
00716  b3a5  20 d8 ff          jsr ksaves    save ab ac bis xr, yr (Sprungv.)
00717  b3a8  4c 8b b0  lodsv5 jmp main       Monitor-Warteschleife
00718  b3ab
00719  b3ab
00720  b3ab  ===> Ausführung 'Load/Verify' <===
00721  b3ab
00722  b3ab  a5 93      lodsv6 lda verck      Flag für load/verify
```

```
00723  b3ad  c9 56           cmp #'v'          Verify ?
00724  b3af  f0 06           beq lodsv7        Ja -> Bearbeiten
00725  b3b1  c9 4c           cmp #'l'          Load ?
00726  b3b3  d0 bb           bne lodsv3        Nein -> Fehler ausgeben
00727  b3b5  a9 00           lda #0            Flag für 'Load' setzen
00728  b3b7  20 d5 ff  lodsv7 jsr kloads       load/verify (ac) ab Adresse in
                                               xr/yr (Sprungv.)
00729  b3ba  a5 90           lda status        I/O-Status 'st'
00730  b3bc  29 10           and #%00010000    Fehler während lesen ?
00731  b3be  f0 e8           beq lodsv5        Ja -> Abbruch
00732  b3c0  a5 93           lda verck         Flag für load/verify
00733  b3c2  f0 ac           beq lodsv3        Kein Verify-Error
00734  b3c4  20 7d ff        jsr kprimm        folg. Text ausgeben und
                                               überspringen (Sprungv.)
00735  b3c7
00736  b3c7  20 45 52        .byte ' error', eom
00737  b3ca  52 4f 52
00738  b3cd  00
00739  b3ce
00740  b3ce  4c 8b b0        jmp main          Monitor-Warteschleife
00741  b3d1
00742  b3d1
00743  b3d1  ===> 'Load' ohne Adress-Angabe durchführen <===
00744  b3d1
00745  b3d1  a6 66     lodsv8 ldx t2           addrl aktuelle Adresse
00746  b3d3  a4 67            ldy t2+1         addrh aktuelle Adresse
00747  b3d5  a9 00            lda #0
00748  b3d7  85 b9            sta sa           akt. Sekundäradresse
00749  b3d9  f0 d0            beq lodsv6       Ausführung 'Load/Verify'
00750  b3db
00751  b3db
00752  b3db  ===> Speicherbereich mit Konstante füllen <===
00753  b3db
00754  b3db  20 83 b9   fill  jsr range        Adresse und Blocklänge holen
00755  b3de  b0 23            bcs filerr       Fehler -> Abbruch
00756  b3e0  a5 68            lda t2+2         bank aktuelle Adresse
00757  b3e2  cd b9 0a         cmp temps+2      Zwischenspeicher
00758  b3e5  d0 1c            bne filerr       Fehlerbehandlung
00759  b3e7  20 a7 b7         jsr parse        Operanden ermitteln u. auf
                                               gült.Trennnzeichen prüfen
00760  b3ea  b0 17            bcs filerr       Fehlerbehandlung
00761  b3ec  a0 00            ldy #0
00762  b3ee
00763  b3ee
00764  b3ee  ===> Zeichen in Speicher bringen <===
00765  b3ee
00766  b3ee  a5 60      fill1 lda to           Zu schreibendes Zeichen
00767  b3f0  20 2a b1         jsr stash        Ein Byte in beliebige Bank
                                               schreiben
00768  b3f3  20 e1 ff         jsr kstop        Stop-Taste üb. Vector abfragen
                                               (Sprungv.)
00769  b3f6  f0 08            beq fill2        Stop gedrückt, Abbruch ->
00770  b3f8  20 50 b9         jsr inct2        Zeiger um 1 erhöhen
00771  b3fb  20 3c b9         jsr dect1        Hilfszeiger um 1 vermindern
00772  b3fe  b0 ee            bcs fill1        Zeichen in Speicher bringen
00773  b400  4c 8b b0   fill2 jmp main         Monitor-Warteschleife
00774  b403
00775  b403  4c bc b0   filerr jmp error       Fehlerbehandlung
```

```
00776  b406
00777  b406                    .end
00778  b406                    .lib mon.m02
00779  b406
00780  b406
00781  b406  ===> Direkt-Assembler aufrufen <===
00782  b406
00783  b406  b0 3a      assem  bcs as60         Adresse fehlt, Fehler ->
00784  b408  20 01 b9          jsr totot2       Zeiger kopieren
00785  b40b
00786  b40b
00787  b40b  ===> Zwischenspeicher löschen <===
00788  b40b
00789  b40b  a2 00       as10   ldx #0
00790  b40d  8e a1 0a           stx hulp+1       Monitor-Hilfsbereich
00791  b410  8e b4 0a           stx count        Zwischenzähler für
                                                 Block-Operationen
00792  b413  20 e9 b8    as20   jsr gnc          Ein Zeichen aus Eingabe-Puffer
                                                 holen
00793  b416  d0 07              bne as30         Mnemonic auslesen
00794  b418  e0 00              cpx #0           Kein Mnemonic eingegeben ?
00795  b41a  d0 03              bne as30         Mnemonic auslesen
00796  b41c  4c 8b b0           jmp main         Monitor-Warteschleife
00797  b41f
00798  b41f
00799  b41f  ===> Mnemonic auslesen <===
00800  b41f
00801  b41f  c9 20       as30   cmp #' '         Leerzeichen überlesen
00802  b421  f0 e8              beq as10         Zwischenspeicher löschen
00803  b423  9d ac 0a           sta msal,x       Hilfsregister für Adressen
00804  b426  e8                 inx
00805  b427  e0 03              cpx #3           3 Bytes gelesen ?
00806  b429  d0 e8              bne as20         Nein -> weiterlesen
00807  b42b
00808  b42b  ===> Mnemonic in 2 Bytes verschlüsseln <===
00809  b42b
00810  b42b  ca          as40   dex              Alle drei Bytes bearbeitet ?
00811  b42c  30 17              bmi as70         Ja -> Operanden auswerten
00812  b42e  bd ac 0a           lda msal,x       Hilfsregister für Adressen
00813  b431  38                 sec
00814  b432  e9 3f              sbc #$3f         ASCII-Zeichen auf 5 Bits reduzieren
00815  b434  a0 05              ldy #5           Anzahl Bits/Zeichen
00816  b436  4a          as50   lsr a            Bits in 'hulp' einshiften
00817  b437  6e a1 0a           ror hulp+1       Monitor-Hilfsbereich
00818  b43a  6e a0 0a           ror hulp         Monitor-Hilfsbereich
00819  b43d  88                 dey              Alle Bits bearbeitet ?
00820  b43e  d0 f6              bne as50         Nein, -> weiter
00821  b440  f0 e9              beq as40         Ja, -> nächster Buchstabe
00822  b442
00823  b442  4c bc b0    as60   jmp error        Fehler ausgeben, falscher
                                                 Monitorbefehl
00824  b445
00825  b445
00826  b445
00827  b445  ===> Operanden auswerten <===
00828  b445
00829  b445  a2 02       as70   ldx #2           Zeiger in Zwischenbereich setzen
```

```
00830  b447  ad b4 0a   as71   lda count       Zwischenzähler für
                                               Block-Operationen
00831  b44a  d0 30             bne as93
00832  b44c  20 ce b7          jsr eval        Einen Operanden (Adresse) auswerten
00833  b44f  f0 29             beq as92
00834  b451  b0 ef             bcs as60        Fehler -> Abbruch
00835  b453  a9 24             lda #'$'        'Hexa'-Zeichen in Puffer bringen
00836  b455  9d a0 0a          sta hulp,x      Monitor-Hilfsbereich
00837  b458  e8                inx             Zeiger erhöhen
00838  b459  a5 62             lda to+2        bank gelesene Adresse für
                                               Monitor-Befehl
00839  b45b  d0 e5             bne as60        Nicht Null -> Fehler
00840  b45d  a0 04             ldy #4          Anzahl Nybbles für Hexwert
00841  b45f  ad b6 0a          lda shift       Hilfszähler für Basis-Umrechnungen
00842  b462  c9 08             cmp #8          Basis des Operanden 8 ?
00843  b464  90 05             bcc as80        Kleiner -> weiter
00844  b466  cc b4 0a          cpy count       Zwischenzähler für
                                               Block-Operationen
00845  b469  f0 06             beq as90
00846  b46b  a5 61      as80   lda to+1        addrh gelesene Adresse für
                                               Monitor-Befehl
00847  b46d  d0 02             bne as90        High-Byte nicht null -> Weiter
00848  b46f  a0 02             ldy #2          Nur 2 Bytes
00849  b471  a9 30      as90   lda #'0'        Hilfsbereich mit Nullen auffüllen
00850  b473  9d a0 0a   as91   sta hulp,x      Monitor-Hilfsbereich
00851  b476  e8                inx             Zeiger erhöhen
00852  b477  88                dey             Anzahl -1
00853  b478  d0 f9             bne as91        Wenn nicht Null, weiter ->
00854  b47a  c6 7a      as92   dec txtptr      Zeiger in aktuellen Monitor-Befehl
00855  b47c  20 e9 b8   as93   jsr gnc         Ein Zeichen aus Eingabe-Puffer
                                               holen
00856  b47f  f0 0e             beq as100       Adressierungsart ermitteln
00857  b481  c9 20             cmp #' '        Space ?
00858  b483  f0 c2             beq as71        Ja -> nächstes Zeichen lesen
00859  b485  9d a0 0a          sta hulp,x      Monitor-Hilfsbereich
00860  b488  e8                inx             Zeiger erhöhen
00861  b489  e0 0a             cpx #10         Mehr als 9 Zeichen ?
00862  b48b  90 ba             bcc as71        Nein -> nächstes Zeichen
00863  b48d  b0 b3             bcs as60        Ja, Fehler ->
00864  b48f
00865  b48f
00866  b48f  ===> Adressierungsart ermitteln <===
00867  b48f
00868  b48f  86 63      as100  stx t1          addrl Hilfszeiger Monitor
00869  b491  a2 00             ldx #0
00870  b493  8e b1 0a          stx wrap        Zwischenspeicher
00871  b496  a2 00      as110  ldx #0
00872  b498  86 9f             stx ptr2
00873  b49a  ad b1 0a          lda wrap        Zwischenspeicher
00874  b49d  20 59 b6          jsr dset        OP-Code in AC analysieren und
                                               Zeiger auf Tabellen setzen
00875  b4a0  ae aa 0a          ldx format      aktuelle Adressierungsart bei
                                               Ass./Dis-Ass.
00876  b4a3  86 64             stx t1+1        addrh Hilfzeiger Monitor
00877  b4a5  aa                tax
00878  b4a6  bd 61 b7          lda mnemr,x     Tabelle 2 verschlüsselte OP-Codes,
                                               3 ASCII -> 2 Bytes
```

```
00879  b4a9  20 7f b5          jsr tstrx       Byte auf Gleichheit testen, Fehler:
                                                Sprung nach as110 od.Error
00880  b4ac  bd 21 b7          lda mneml,x     Tabelle 1 verschlüsselte OP-Codes,
                                                3 ASCII -> 2 Bytes
00881  b4af  20 7f b5          jsr tstrx       Byte auf Gleichheit testen, Fehler:
                                                Sprung nach as110 od.Error
00882  b4b2  a2 06             ldx #6          Index in Zeichentabellen
00883  b4b4  e0 03       as210 cpx #3          Adressposition erreicht ?
00884  b4b6  d0 14             bne as230       nein -> weiter
00885  b4b8  ac ab 0a          ldy length      Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
00886  b4bb  f0 0f             beq as230
00887  b4bd  ad aa 0a  as220   lda format      aktuelle Adressierungsart bei
                                                Ass./Dis-Ass.
00888  b4c0  c9 e8             cmp #$e8        Relativer Sprungbefehl ?
00889  b4c2  a9 30             lda #'0'
00890  b4c4  b0 1e             bcs as250       Ja -> weiter
00891  b4c6  20 7c b5          jsr tst2        Zwei Bytes auf Gleichheit testen
00892  b4c9  88                dey
00893  b4ca  d0 f1             bne as220
00894  b4cc  0e aa 0a  as230   asl format      aktuelle Adressierungsart bei
                                                Ass./Dis-Ass.
00895  b4cf  90 0e             bcc as240       Bit nicht gesetzt -> weiter
00896  b4d1  bd 14 b7          lda char1-1,x   Vergleichstabelle 1 für OP-Code
                                                Analyse
00897  b4d4  20 7f b5          jsr tstrx       Byte auf Gleichheit testen, Fehler:
                                                Sprung nach as110 od.Error
00898  b4d7  bd 1a b7          lda char2-1,x   Vergleichstabelle 2 für OP-Code
                                                Analyse
00899  b4da  f0 03             beq as240       Kein zweites Zeichen -> weiter
00900  b4dc  20 7f b5          jsr tstrx       Byte auf Gleichheit testen, Fehler:
                                                Sprung nach as110 od.Error
00901  b4df  ca          as240 dex             index - 1
00902  b4e0  d0 d2             bne as210       nicht Null -> weiter testen
00903  b4e2  f0 06             beq as300       Okay, -> Befehl erkannt
00904  b4e4
00905  b4e4  ===> Relativen Sprungbefehl aufbereiten <===
00906  b4e4
00907  b4e4  20 7c b5  as250   jsr tst2        Zwei Bytes auf Gleichheit testen
00908  b4e7  20 7c b5          jsr tst2        Zwei Bytes auf Gleichheit testen
00909  b4ea  a5 63       as300 lda t1          addrl Hilfszeiger Monitor
00910  b4ec  c5 9f             cmp ptr2
00911  b4ee  f0 03             beq as310
00912  b4f0  4c 8b b5          jmp tsto5
00913  b4f3
00914  b4f3  ac ab 0a  as310   ldy length      Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
00915  b4f6  f0 32             beq as500
00916  b4f8  a5 64             lda t1+1        addrh Hilfzeiger Monitor
00917  b4fa  c9 9d             cmp #$9d        war es relativer Sprungbefehl ?
00918  b4fc  d0 23             bne as400       nein -> weiter
00919  b4fe  a5 60             lda to          addrl gelesene Adresse für
                                                Monitor-Befehl
00920  b500  e5 66             sbc t2          addrl aktuelle Adresse
00921  b502  aa                tax
00922  b503  a5 61             lda to+1        addrh gelesene Adresse für
                                                Monitor-Befehl
00923  b505  e5 67             sbc t2+1        addrh aktuelle Adresse
```

```
00924  b507  90 08           bcc as320
00925  b509  d0 6e           bne aerr        Unterlauf -> falsche Adresse,
                                             Fehler !
00926  b50b  e0 82           cpx #$82        Rückwärts > $81 ?
00927  b50d  b0 6a           bcs aerr        Ja -> Fehler
00928  b50f  90 08           bcc as340       Nein -> Okay
00929  b511  a8        as320 tay
00930  b512  c8              iny
00931  b513  d0 64           bne aerr
00932  b515  e0 82           cpx #$82        Offset > $81 ?
00933  b517  90 60           bcc aerr        Nein -> Fehler
00934  b519  ca        as340 dex             Offset korrigieren
00935  b51a  ca              dex
00936  b51b  8a              txa
00937  b51c  ac ab 0a        ldy length      Länge des aktuellen Befehls bei
                                             Ass./Dis-Ass.
00938  b51f  d0 03           bne as420
00939  b521  b9 5f 00  as400 lda to-1,y      addrl gelesene Adresse für
                                             Monitor-Befehl
00940  b524  20 2a b1  as420 jsr stash       Ein Byte in beliebige Bank
                                             schreiben
00941  b527  88              dey
00942  b528  d0 f7           bne as400
00943  b52a  ad b1 0a  as500 lda wrap        Zwischenspeicher
00944  b52d  20 2a b1        jsr stash       Ein Byte in beliebige Bank
                                             schreiben
00945  b530  20 ad b8        jsr cronly      Ausgabe 'carriage return + cursur
                                             aufwärts'
00946  b533  20 7d ff        jsr kprimm      folg. Text ausgeben und
                                             überspringen (Sprungv.)
00947  b536
00948  b536  41 20 1b        .byte 'a ',esc,'q',eom
00949  b539  51 00
00950  b53b
00951  b53b  20 dc b5        jsr dis400      Aktuelle Adresse Dis-assemblieren
00952  b53e  ee ab 0a        inc length      Länge des aktuellen Befehls bei
                                             Ass./Dis-Ass.
00953  b541  ad ab 0a        lda length      Länge des aktuellen Befehls bei
                                             Ass./Dis-Ass.
00954  b544  20 52 b9        jsr addt2       Zeiger um AC erhöhen
00955  b547
00956  b547  ===> Neue Zeile für Assembler vorbereiten <===
00957  b547
00958  b547  a9 41           lda #'a'        'a' in Tastaturpuffer
00959  b549  8d 4a 03        sta keyd        Tastaturpuffer
00960  b54c  a9 20           lda #' '        Space
00961  b54e  8d 4b 03        sta keyd+1      Tastaturpuffer
00962  b551  8d 51 03        sta keyd+7      Tastaturpuffer
00963  b554  a5 68           lda t2+2        bank aktuelle Adresse
00964  b556  20 d2 b8        jsr makhex      Byte in AC in Hexa umwandeln nach
                                             AC und XR
00965  b559  8e 4c 03        stx keyd+2      Tastaturpuffer
00966  b55c  a5 67           lda t2+1        addrh aktuelle Adresse
00967  b55e  20 d2 b8        jsr makhex      Byte in AC in Hexa umwandeln nach
                                             AC und XR
00968  b561  8d 4d 03        sta keyd+3      Tastaturpuffer
00969  b564  8e 4e 03        stx keyd+4      Tastaturpuffer
00970  b567  a5 66           lda t2          addrl aktuelle Adresse
```

```
00971  b569  20 d2 b8          jsr makhex       Byte in AC in Hexa umwandeln nach
                                                AC und XR
00972  b56c  8d 4f 03          sta keyd+5       Tastaturpuffer
00973  b56f  8e 50 03          stx keyd+6       Tastaturpuffer
00974  b572  a9 08             lda #8           Insgesamt 8 Bytes
00975  b574  85 d0             sta ndx          Bytes im Tastaturpuffer
00976  b576  4c 8b b0          jmp main         Monitor-Warteschleife
00977  b579
00978  b579  4c bc b0    aerr  jmp error        Fehler ausgeben
00979  b57c
00980  b57c
00981  b57c  ===> Zwei Bytes auf Gleichheit testen <===
00982  b57c
00983  b57c  20 7f b5    tst2  jsr tstrx        Routine zweimal durchlaufen
00984  b57f
00985  b57f
00986  b57f  ===> Byte auf Gleichheit testen, Fehler: Sprung n.as110 od.Error <===
00987  b57f
00988  b57f  8e af 0a    tstrx stx sxreg        Rettungsbyte für X-Register
00989  b582  a6 9f             ldx ptr2         Zeiger in Hilfsbereich
00990  b584  dd a0 0a          cmp hulp,x       Monitor-Hilfsbereich
00991  b587  f0 0a             beq tst10        Gleich -> Okay
00992  b589  68                pla              Stapel korrigieren
00993  b58a  68                pla              'RTS' eliminieren
00994  b58b  ee b1 0a    tsto5 inc wrap         Zwischenspeicher
00995  b58e  f0 e9             beq aerr         Fehler, Abbruch
00996  b590  4c 96 b4          jmp as110        zurück
00997  b593
00998  b593  e6 9f       tst10 inc ptr2         Zeiger in Hilfsbereich erhöhen
00999  b595  ae af 0a          ldx sxreg        Rettungsbyte für X-Register
01000  b598  60                rts
01001  b599
01002  b599
01003  b599  ===> Speicherbereich Dis-Assemblieren <===
01004  b599
01005  b599  b0 08       disasm bcs dis10       Standardanzahl (20 Bytes) setzen
01006  b59b  20 01 b9          jsr totot2       Zeiger kopieren
01007  b59e  20 a7 b7          jsr parse        Operanden ermitteln u. auf
                                                gült.Trennnzeichen prüfen
01008  b5a1  90 06             bcc dis20        Ende-Adresse vorhanden -> weiter
01009  b5a3
01010  b5a3
01011  b5a3  ===> Standardanzahl (20 Bytes) setzen <===
01012  b5a3
01013  b5a3  a9 14       dis10 lda #20          20 Bytes sind Standard
01014  b5a5  85 60             sta to           addrl gelesene Adresse für
                                                Monitor-Befehl
01015  b5a7  d0 05             bne dis100       Neue Zeile (carriage return) und
                                                Zeile löschen
01016  b5a9
01017  b5a9  20 0e b9    dis20 jsr subom2       Zeiger von aktueller Adresse
                                                subtrahieren
01018  b5ac  90 23             bcc dis200       Fehlerbehandlung
01019  b5ae
01020  b5ae
01021  b5ae  ===> Neue Zeile (carriage return) und Zeile löschen <===
01022  b5ae
```

```
01023  b5ae  20 7d ff   dis100 jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
01024  b5b1
01025  b5b1  0d 1b 51          .byte cr, esc,'q', eom
01026  b5b4  00
01027  b5b5

01028  b5b5  20 e1 ff          jsr kstop        Stop-Taste üb. Vector abfragen
                                                (Sprungv.)
01029  b5b8  f0 14             beq dis110       Stop-Taste gedrückt, Abbruch ->
01030  b5ba  20 d4 b5          jsr dis300       Eine Zeile Dis-Assemblieren
01031  b5bd  ee ab 0a          inc length       Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
01032  b5c0  ad ab 0a          lda length       Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
01033  b5c3  20 52 b9          jsr addt2        Zeiger um AC erhöhen
01034  b5c6  ad ab 0a          lda length       Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
01035  b5c9  20 24 b9          jsr subto        Adresse um AC vermindern
01036  b5cc  b0 e0             bcs dis100       Neue Zeile (carriage return) und
                                                Zeile löschen
01037  b5ce  4c 8b b0   dis110 jmp main         Monitor-Warteschleife
01038  b5d1
01039  b5d1  4c bc b0   dis200 jmp error        Fehlerbehandlung
01040  b5d4
01041  b5d4
01042  b5d4  ===> Eine Zeile Dis-Assemblieren <===
01043  b5d4
01044  b5d4  a9 2e      dis300 lda #'.'         Punkt für Befehl 'Assembler'
                                                ausgeben
01045  b5d6  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01046  b5d9  20 a8 b8          jsr putspc       Ausgabe 'space'
01047  b5dc  20 92 b8   dis400 jsr putt2        Adresse als Hex-Bytes ausgeben
01048  b5df  20 a8 b8          jsr putspc       Ausgabe 'space'
01049  b5e2  a0 00             ldy #0           Zeiger in RAM
01050  b5e4  20 1a b1          jsr fetch        Ein Byte aus beliebiger Bank lesen
01051  b5e7  20 59 b6          jsr dset         OP-Code in AC analysieren und
                                                Zeiger auf Tabellen setzen
01052  b5ea  48                pha              Index in Mnemo-Tabellen retten
01053  b5eb  ae ab 0a          ldx length       Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
01054  b5ee  e8                inx              Anzahl Befehls-Bytes + 1
01055  b5ef  ca         pradro dex              Anzahl Befehlsbytes - 1
01056  b5f0  10 0a             bpl pradrl       Einen Befehl auswerten
01057  b5f2  20 7d ff          jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
01058  b5f5
01059  b5f5  20 20 20          .byte '   ', eom
01060  b5f8  00
01061  b5f9
01062  b5f9  4c 02 b6          jmp pradrm       Keine Adresse -> weiter
01063  b5fc
01064  b5fc
01065  b5fc  ===> Einen Befehl auswerten <===
01066  b5fc
01067  b5fc  20 1a b1   pradrl jsr fetch        Ein Byte aus beliebiger Bank lesen
01068  b5ff  20 a5 b8          jsr puthxs       Accu als Hex-Bytes ausgeben + SPACE
```

```
01069  b602  c8          pradrm iny              Zeiger erhöhen
01070  b603  c0 03              cpy #3           3 Bytes erreicht ?
01071  b605  90 e8              bcc pradro       Nein -> Testen und weiterausgeben
01072  b607  68                 pla
01073  b608  a2 03              ldx #3           3  Zeichen sind auszugeben
01074  b60a  20 a1 b6           jsr prmne        OP-Code aus komprimierter Tabelle
                                                 ausgeben
01075  b60d  a2 06              ldx #6           Zähler für Adressbytes
01076  b60f
01077  b60f
01078  b60f  ===> Adressierungsart ausgeben <===
01079  b60f
01080  b60f  e0 03       pradr1 cpx #$03         Schon Adresse erreicht ?
01081  b611  d0 17              bne pradr3       Nein -> weiter
01082  b613  ac ab 0a           ldy length       Länge des aktuellen Befehls bei
                                                 Ass./Dis-Ass.
01083  b616  f0 12              beq pradr3       Impliziter Befehl -> keine Adresse
01084  b618  ad aa 0a    pradr2 lda format       aktuelle Adressierungsart bei
                                                 Ass./Dis-Ass.
01085  b61b  c9 e8              cmp #$e8         Relativer Sprungbefehl ?
01086  b61d  08                 php              Bedingungsflags retten
01087  b61e  20 1a b1           jsr fetch        Ein Byte aus beliebiger Bank lesen
01088  b621  28                 plp              Bedingungsflags zurück
01089  b622  b0 1d              bcs relad1       Zieladresse für Branch-Befehl
                                                 errechnen
01090  b624  20 c2 b8           jsr puthex       Hex-Byte in AC als 2 ASCII-Zeichen
                                                 ausgeben
01091  b627  88                 dey
01092  b628  d0 ee              bne pradr2
01093  b62a
01094  b62a  0e aa 0a    pradr3 asl format       aktuelle Adressierungsart bei
                                                 Ass./Dis-Ass.
01095  b62d  90 0e              bcc pradr4       Keine besondere Adressierungart,
                                                 weiter ->
01096  b62f  bd 14 b7           lda char1-1,x    Zeichentabelle 1 für Adr.Art
                                                 Analyse
01097  b632  20 d2 ff           jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                 Kanal (Sprungv.)
01098  b635  bd 1a b7           lda char2-1,x    Zeichentabelle 2 für Adr.Art
                                                 Analyse
01099  b638  f0 03              beq pradr4       Kein Zeichen auszugeben, -> weiter
01100  b63a  20 d2 ff           jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                 Kanal (Sprungv.)
01101  b63d  ca          pradr4 dex
01102  b63e  d0 cf              bne pradr1       Adressierungsart ausgeben
01103  b640  60                 rts
01104  b641
01105  b641
01106  b641  ===> Zieladresse für Branch-Befehl errechnen <===
01107  b641
01108  b641  20 4d b6    relad1 jsr pcadj3       Offset in Accu (+/-) zu aktueller
                                                 Adresse errechnen
01109  b644  18                 clc
01110  b645  69 01              adc #1           um 1 erhöhen
01111  b647  d0 01              bne relad2       Ermittelte Zieladresse als
                                                 Hex-Bytes ausgeben
01112  b649  e8                 inx
```

```
01113  b64a  4c 9f b8   relad2 jmp putwrd       Ermittelte Zieladresse als
                                                Hex-Bytes ausgeben
01114  b64d
01115  b64d
01116  b64d  ===> Offset in Accu (+/-) zu aktueller Adresse errechnen <===
01117  b64d
01118  b64d  a6 67      pcadj3 ldx t2+1         addrh aktuelle Adresse
01119  b64f  a8                tay             Offset zur Adresse
01120  b650  10 01             bpl pcadj4      Vorwärtsverzeigung -> weiter
01121  b652  ca                dex             High-Byte um 1 vermindern
01122  b653  65 66      pcadj4 adc t2          addrl aktuelle Adresse
01123  b655  90 01             bcc pcrts       Kein Übertrag -> weiter
01124  b657  e8                inx             High-Byte um 1 erhöhen
01125  b658  60         pcrts  rts
01126  b659
01127  b659
01128  b659  ===> OP-Code in AC analysieren und Zeiger auf Tabellen setzen <===
01129  b659
01130  b659  a8         dset   tay             OP-Code retten
01131  b65a  4a                lsr a           bit 0 Null ?
01132  b65b  90 0b             bcc ieven       ja -> weiter
01133  b65d  4a                lsr a           bit 1 Null ?
01134  b65e  b0 17             bcs err         Fehler, kein Befehl mit '11' am
                                                Ende zulässig
01135  b660  c9 22             cmp #$22        OP-Code $89 ?
01136  b662  f0 13             beq err         Ja, Fehler ->
01137  b664  29 07             and #%00000111  Bits 2-4 isolieren
01138  b666  09 80             ora #%10000000  Bit 7 setzen
01139  b668
01140  b668
01141  b668  ===> Adressierungsart feststellen <===
01142  b668
01143  b668  4a         ieven  lsr a           Um 1 Bit shiften
01144  b669  aa                tax             Zeiger in Tabelle
01145  b66a  bd c3 b6           lda nmode,x     Typ-Tabelle 1 für Analyse
                                                Adressierungsart
01146  b66d  b0 04             bcs rtmode      Wenn bit 2 gesetzt war, weiter ->
01147  b66f  4a                lsr a           Oberes  Nybble -> Unteres Nybble
01148  b670  4a                lsr a
01149  b671  4a                lsr a
01150  b672  4a                lsr a
01151  b673
01152  b673
01153  b673  ===> Format feststellen <===
01154  b673
01155  b673  29 0f      rtmode and #%00001111  Unteres Nybble isolieren
01156  b675  d0 04             bne getfmt      Ungleich Null -> Weiter
01157  b677  a0 80      err    ldy #$80        Zeiger setzen für '???'
01158  b679  a9 00             lda #0
01159  b67b  aa         getfmt tax
01160  b67c  bd 07 b7           lda nmode2,x    Typ-Tabelle 2 für Analyse
                                                Adressierungsart
01161  b67f  8d aa 0a           sta format      aktuelle Adressierungsart bei
                                                Ass./Dis-Ass.
01162  b682  29 03             and #%00000011  Länge des Befehls -1
01163  b684  8d ab 0a           sta length      Länge des aktuellen Befehls bei
                                                Ass./Dis-Ass.
01164  b687  98                tya             Operationscode zurück in Accu
```

```
01165  b688  29 8f           and #%10001111   Bits maskieren
01166  b68a  aa              tax              Bits retten
01167  b68b  98              tya              OP-Code wieder in Accu
01168  b68c  a0 03           ldy #3           Schleifenzähler setzen
01169  b68e  e0 8a           cpx #%10001010   TXS,TAX,TSX,DEX,NOP ?
01170  b690  f0 0b           beq mnndx3       Ja, eine Schleifer überspringen ->
01171  b692  4a        mnndx1 lsr a            Bit 0 prüfen
01172  b693  90 08           bcc mnndx3       wenn Null -> weiter
01173  b695  4a              lsr a            Bit 0 ignorieren
01174  b696  4a        mnndx2 lsr a            Bit 0 ignorieren
01175  b697  09 20           ora #%00100000   Bit 5 setzen
01176  b699  88              dey              Zähler - 1
01177  b69a  d0 fa           bne mnndx2       Wenn nicht Null -> weiter
01178  b69c  c8              iny              Zähler erhöhen
01179  b69d  88        mnndx3 dey              Zähler vermindern
01180  b69e  d0 f2           bne mnndx1       Wenn nicht Null -> weiter
01181  b6a0  60              rts
01182  b6a1
01183  b6a1
01184  b6a1  ===> OP-Code aus komprimierter Tabelle ausgeben <===
01185  b6a1
01186  b6a1  a8         prmne tay              Index in Texttabellen
01187  b6a2  b9 21 b7        lda mneml,y      Tabelle 1 verschlüsselte OP-Codes,
                                              3 ASCII -> 2 Bytes
01188  b6a5  85 63           sta t1           addrl Hilfszeiger Monitor
01189  b6a7  b9 61 b7        lda mnemr,y      Tabelle 2 verschlüsselte OP-Codes,
                                              3 ASCII -> 2 Bytes
01190  b6aa  85 64           sta t1+1         addrh Hilfzeiger Monitor
01191  b6ac
01192  b6ac
01193  b6ac  ===> OP-Code in 't1' als ASCII-Zeichen ausgeben <===
01194  b6ac
01195  b6ac  a9 00      prmn1 lda #0           Accu löschen
01196  b6ae  a0 05           ldy #5           Anzahl Bits / Buchstabe
01197  b6b0  06 64      prmn2 asl t1+1         16 Bit-Wert nach links schiften
01198  b6b2  26 63           rol t1
01199  b6b4  2a              rol a            1 Bit in Accu einbringen
01200  b6b5  88              dey              Anzahl -1
01201  b6b6  d0 f8           bne prmn2        Nicht Null, weiter ->
01202  b6b8  69 3f           adc #$3f         Zeichen auf ASCII-Format bringen
01203  b6ba  20 d2 ff        jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                              Kanal (Sprungv.)
01204  b6bd  ca              dex              Anzahl Zeichen - 1
01205  b6be  d0 ec           bne prmn1        OP-Code in 't1' als ASCII-Zeichen
                                              ausgeben
01206  b6c0  4c a8 b8        jmp putspc       Ausgabe 'space'
01207  b6c3
01208  b6c3
01209  b6c3  ===> Typ-Tabelle 1 für Analyse Adressierungsart <===
01210  b6c3
01211  b6c3  Aufbau:    76543210
01212  b6c3              xxxx....  Index in 'nmode2' (Wird auf unteres Nybble
                                                      geshiftet)
01213  b6c3              ....xxxx  Index in 'nmode2'
01214  b6c3
01215  b6c3  40 02 45    nmode  .byte $40, $02, $45, $03, $d0, $08
01216  b6c6  03 d0 08
01217  b6c9  40 09 30           .byte $40, $09, $30, $22, $45, $33
```

```
01218  b6cc  22 45 33
01219  b6cf  d0 08 40          .byte $d0, $08, $40, $09, $40, $02
01220  b6d2  09 40 02
01221  b6d5  45 33 d0          .byte $45, $33, $d0, $08, $40, $09
01222  b6d8  08 40 09
01223  b6db  40 02 45          .byte $40, $02, $45, $b3, $d0, $08
01224  b6de  b3 d0 08
01225  b6e1  40 09 00          .byte $40, $09, $00, $22, $44, $33
01226  b6e4  22 44 33
01227  b6e7  d0 8c 44          .byte $d0, $8c, $44, $00, $11, $22
01228  b6ea  00 11 22
01229  b6ed  44 33 d0          .byte $44, $33, $d0, $8c, $44, $9a
01230  b6f0  8c 44 9a
01231  b6f3  10 22 44          .byte $10, $22, $44, $33, $d0, $08
01232  b6f6  33 d0 08
01233  b6f9  40 09 10          .byte $40, $09, $10, $22, $44, $33
01234  b6fc  22 44 33
01235  b6ff  d0 08 40          .byte $d0, $08, $40, $09, $62, $13
01236  b702  09 62 13
01237  b705  78 a9             .byte $78, $a9
01238  b707
01239  b707
01240  b707  ===> Typ-Tabelle 2 für Analyse Adressierungsart <===
01241  b707
01242  b707  Aufbau:                  76543210
01243  b707                           x.......  ;Führendes '$' ausgeben
01244  b707                           .x......  ;'($' ausgeben
01245  b707                           ..x.....  ;'#$' ausgeben
01246  b707                           ...x....  ;',x' ausgeben
01247  b707                           ....x...  ;')' ausgeben
01248  b707                           .....x..  ;',y' ausgeben
01249  b707                           ......xx  ;Anzahl Befehlsbytes -1
01250  b707
01251  b707  00         nmode2 .byte %00000000  Implizit
01252  b708  21                .byte %00100001  Immediate      #$
01253  b709  81                .byte %10000001  Zeropage       $
01254  b70a  82                .byte %10000010  Absolute       $
01255  b70b  00                .byte %00000000  Implizit
01256  b70c  00                .byte %00000000  Implizit
01257  b70d  59                .byte %01011001  Indiziert,x    ($xx,X)
01258  b70e  4d                .byte %01001101  Indirekt,y     ($xx),Y
01259  b70f  91                .byte %10010001  Zeropage,x     $xx,X
01260  b710  92                .byte %10010010  Absolut,x      $xxxx,X
01261  b711  86                .byte %10000110  Absolut,y      $xxxx,Y
01262  b712  4a                .byte %01001010  Indirekt       ($xxxx)
01263  b713  85                .byte %10000101  Zeropage,y     $xx,Y
01264  b714  9d                .byte %10011101  Relativ        $xxxx
01265  b715                  -------- ergibt nach 3-maligem Shiften $E8 !
01266  b715
01267  b715  ===> Zeichentabelle 1 für Adr.Art Analyse <===
01268  b715
01269  b715  2c           char1  .byte ','        ->  ,y
01270  b716  29                  .byte ')'        ->  )
01271  b717  2c                  .byte ','        ->  ,x
01272  b718  23                  .byte '#'        ->  #$
01273  b719  28                  .byte '('        ->  ($
01274  b71a  24                  .byte '$'        ->  $
01275  b71b
```

```
01276  b71b
01277  b71b  ===> Zeichentabelle 2 für Adr.Art Analyse <===
01278  b71b
01279  b71b  59          char2  .byte 'y'          ->  ,y
01280  b71c  00                 .byte $00          ->  )
01281  b71d  58                 .byte 'x'          ->  ,x
01282  b71e  24                 .byte '$'          ->  #$
01283  b71f  24                 .byte '$'          ->  ($
01284  b720  00                 .byte $00          ->  $
01285  b721
01286  b721
01287  b721  ===> Tabelle 1 verschlüsselte OP-Codes, 3 ASCII -> 2 Bytes <===
01288  b721
01289  b721  Aufbau:
01290  b721                                xxxxx.....  erstes Zeichen  - 63
01291  b721                                .....xxxxx  zweites Zeichen - 63
01292  b721                                .....xxxxx  drittes Zeichen - 63
01293  b721                                .....x      immer null
01294  b721
01295  b721  1c          mneml  .byte >%0001110011011000   -> BRK
01296  b722  8a                 .byte >%1000101001100010   -> PHP
01297  b723  1c                 .byte >%0001110001011010   -> BPL
01298  b724  23                 .byte >%0010001101001000   -> CLC
01299  b725  5d                 .byte >%0101110100100110   -> JSR
01300  b726  8b                 .byte >%1000101101100010   -> PLP
01301  b727  1b                 .byte >%0001101110010100   -> BMI
01302  b728  a1                 .byte >%1010000110001000   -> SEC
01303  b729  9d                 .byte >%1001110101010100   -> RTI
01304  b72a  8a                 .byte >%1000101001000100   -> PHA
01305  b72b  1d                 .byte >%0001110111001000   -> BVC
01306  b72c  23                 .byte >%0010001101010100   -> CLI
01307  b72d  9d                 .byte >%1001110101101000   -> RTS
01308  b72e  8b                 .byte >%1000101101000100   -> PLA
01309  b72f  1d                 .byte >%0001110111101000   -> BVS
01310  b730  a1                 .byte >%1010000110010100   -> SEI
01311  b731  00                 .byte >%0000000000000000   -> ???
01312  b732  29                 .byte >%0010100110110100   -> DEY
01313  b733  19                 .byte >%0001100100001000   -> BCC
01314  b734  ae                 .byte >%1010111010000100   -> TYA
01315  b735  69                 .byte >%0110100101110100   -> LDY
01316  b736  a8                 .byte >%1010100010110100   -> TAY
01317  b737  19                 .byte >%0001100100101000   -> BCS
01318  b738  23                 .byte >%0010001101101110   -> CLV
01319  b739  24                 .byte >%0010010001110100   -> CPY
01320  b73a  53                 .byte >%0101001111110100   -> INY
01321  b73b  1b                 .byte >%0001101111001100   -> BNE
01322  b73c  23                 .byte >%0010001101001010   -> CLD
01323  b73d  24                 .byte >%0010010001110010   -> CPX
01324  b73e  53                 .byte >%0101001111110010   -> INX
01325  b73f  19                 .byte >%0001100110100100   -> BEQ
01326  b740  a1                 .byte >%1010000110001010   -> SED
01327  b741  00                 .byte >%0000000000000000   -> ???
01328  b742  1a                 .byte >%0001101010101010   -> BIT
01329  b743  5b                 .byte >%0101101110100010   -> JMP
01330  b744  5b                 .byte >%0101101110100010   -> JMP
01331  b745  a5                 .byte >%1010010101110100   -> STY
01332  b746  69                 .byte >%0110100101110100   -> LDY
01333  b747  24                 .byte >%0010010001110100   -> CPY
```

```
01334  b748  24              .byte >%0010010001110010   -> CPX
01335  b749  ae              .byte >%1010111001000100   -> TXA
01336  b74a  ae              .byte >%1010111001101000   -> TXS
01337  b74b  a8              .byte >%1010100010110010   -> TAX
01338  b74c  ad              .byte >%1010110100110010   -> TSX
01339  b74d  29              .byte >%0010100110110010   -> DEX
01340  b74e  00              .byte >%0000000000000000   -> ???
01341  b74f  7c              .byte >%0111110000100010   -> NOP
01342  b750  00              .byte >%0000000000000000   -> ???
01343  b751  15              .byte >%0001010100011010   -> ASL
01344  b752  9c              .byte >%1001110000011010   -> ROL
01345  b753  6d              .byte >%0110110100100110   -> LSR
01346  b754  9c              .byte >%1001110000100110   -> ROR
01347  b755  a5              .byte >%1010010101110010   -> STX
01348  b756  69              .byte >%0110100101110010   -> LDX
01349  b757  29              .byte >%0010100110001000   -> DEC
01350  b758  53              .byte >%0101001111001000   -> INC
01351  b759  84              .byte >%1000010011000100   -> ORA
01352  b75a  13              .byte >%0001001111001010   -> AND
01353  b75b  34              .byte >%0011010000100110   -> EOR
01354  b75c  11              .byte >%0001000101001000   -> ADC
01355  b75d  a5              .byte >%1010010101000100   -> STA
01356  b75e  69              .byte >%0110100101000100   -> LDA
01357  b75f  23              .byte >%0010001110100010   -> CMP
01358  b760  a0              .byte >%1010000011001000   -> SBC
01359  b761
01360  b761
01361  b761  ===> Tabelle 2 verschlüsselte OP-Codes, 3 ASCII -> 2 Bytes <===
01362  b761
01363  b761  d8        mnemr .byte <%0001110011011000   -> BRK
01364  b762  62              .byte <%1000101001100010   -> PHP
01365  b763  5a              .byte <%0001110001011010   -> BPL
01366  b764  48              .byte <%0010001101001000   -> CLC
01367  b765  26              .byte <%0101110100100110   -> JSR
01368  b766  62              .byte <%1000101101100010   -> PLP
01369  b767  94              .byte <%0001101110010100   -> BMI
01370  b768  88              .byte <%1010000110001000   -> SEC
01371  b769  54              .byte <%1001110101010100   -> RTI
01372  b76a  44              .byte <%1000101001000100   -> PHA
01373  b76b  c8              .byte <%0001110111001000   -> BVC
01374  b76c  54              .byte <%0010001101010100   -> CLI
01375  b76d  68              .byte <%1001110101101000   -> RTS
01376  b76e  44              .byte <%1000101101000100   -> PLA
01377  b76f  e8              .byte <%0001110111101000   -> BVS
01378  b770  94              .byte <%1010000110010100   -> SEI
01379  b771  00              .byte <%0000000000000000   -> ???
01380  b772  b4              .byte <%0010100110110100   -> DEY
01381  b773  08              .byte <%0001100100001000   -> BCC
01382  b774  84              .byte <%1010111010000100   -> TYA
01383  b775  74              .byte <%0110100101110100   -> LDY
01384  b776  b4              .byte <%1010100010110100   -> TAY
01385  b777  28              .byte <%0001100100101000   -> BCS
01386  b778  6e              .byte <%0010001101101110   -> CLV
01387  b779  74              .byte <%0010010001110100   -> CPY
01388  b77a  f4              .byte <%0101001111110100   -> INY
01389  b77b  cc              .byte <%0001101111001100   -> BNE
01390  b77c  4a              .byte <%0010001101001010   -> CLD
01391  b77d  72              .byte <%0010010001110010   -> CPX
```

```
01392  b77e  f2                .byte <%0101001111110010   -> INX
01393  b77f  a4                .byte <%0001100110100100   -> BEQ
01394  b780  8a                .byte <%1010000110001010   -> SED
01395  b781  00                .byte <%0000000000000000   -> ???
01396  b782  aa                .byte <%0001101010101010   -> BIT
01397  b783  a2                .byte <%0101101110100010   -> JMP
01398  b784  a2                .byte <%0101101110100010   -> JMP
01399  b785  74                .byte <%1010010101110100   -> STY
01400  b786  74                .byte <%0110100101110100   -> LDY
01401  b787  74                .byte <%0010010001110100   -> CPY
01402  b788  72                .byte <%0010010001110010   -> CPX
01403  b789  44                .byte <%1010111001000100   -> TXA
01404  b78a  68                .byte <%1010111001101000   -> TXS
01405  b78b  b2                .byte <%1010100010110010   -> TAX
01406  b78c  32                .byte <%1010110100110010   -> TSX
01407  b78d  b2                .byte <%0010100110110010   -> DEX
01408  b78e  00                .byte <%0000000000000000   -> ???
01409  b78f  22                .byte <%0111110000100010   -> NOP
01410  b790  00                .byte <%0000000000000000   -> ???
01411  b791  1a                .byte <%0001010100011010   -> ASL
01412  b792  1a                .byte <%1001110000011010   -> ROL
01413  b793  26                .byte <%0110110100100110   -> LSR
01414  b794  26                .byte <%1001110000100110   -> ROR
01415  b795  72                .byte <%1010010101110010   -> STX
01416  b796  72                .byte <%0110100101110010   -> LDX
01417  b797  88                .byte <%0010100110001000   -> DEC
01418  b798  c8                .byte <%0101001111001000   -> INC
01419  b799  c4                .byte <%1000010011000100   -> ORA
01420  b79a  ca                .byte <%0001001111001010   -> AND
01421  b79b  26                .byte <%0011010000100110   -> EOR
01422  b79c  48                .byte <%0001000101001000   -> ADC
01423  b79d  44                .byte <%1010010101000100   -> STA
01424  b79e  44                .byte <%0110100101000100   -> LDA
01425  b79f  a2                .byte <%0010001110100010   -> CMP
01426  b7a0  c8                .byte <%1010000011001000   -> SBC
01427  b7a1
01428  b7a1  0d 20 20   regk    .byte cr ,'   '
01429  b7a4  20
01430  b7a5
01431  b7a5                     .end
01432  b7a5                     .lib mon.m03
01433  b7a5
01434  b7a5
01435  b7a5  ===> Zeiger rücksetzen, Operanden ermitteln <===
01436  b7a5
01437  b7a5  c6 7a      pargot dec txtptr        Zeiger in aktuellen Monitor-Befehl
01438  b7a7
01439  b7a7
01440  b7a7  ===> Operanden ermitteln u. auf gült.Trennnzeichen prüfen <===
01441  b7a7
01442  b7a7  20 ce b7   parse   jsr eval         Einen Operanden (Adresse) auswerten
01443  b7aa  b0 16              bcs parse2       Fehler, Abbruch
01444  b7ac  20 e7 b8           jsr glc          Letztes Zeichen aus Eingabepuffer
                                                 nochmals lesen
01445  b7af  d0 09              bne parse1       Zeichen vorhanden -> weiter
01446  b7b1  c6 7a              dec txtptr       Zeiger in aktuellen Monitor-Befehl
01447  b7b3  ad b4 0a           lda count        Zwischenzähler für
                                                 Block-Operationen
```

```
01448  b7b6  d0 11           bne parse4
01449  b7b8  f0 0d           beq parse3
01450  b7ba
01451  b7ba  c9 20     parse1 cmp #' '        Space ?
01452  b7bc  f0 0b           beq parse4       Ja, Okay ->
01453  b7be  c9 2c           cmp #','         Komma ?
01454  b7c0  f0 07           beq parse4       Ja, Okay ->
01455  b7c2  68       parse2 pla              Stapel korrigieren
01456  b7c3  68              pla
01457  b7c4  4c bc b0        jmp error        Fehler ausgeben, falscher
                                              Monitorbefehl
01458  b7c7
01459  b7c7  38       parse3 sec              Flag für Ende setzen
01460  b7c8  24              .byte skip1
01461  b7c9
01462  b7c9  18       parse4 clc              Flag für Ende löschen
01463  b7ca  ad b4 0a        lda count        Zwischenzähler für
                                              Block-Operationen
01464  b7cd  60              rts
01465  b7ce
01466  b7ce
01467  b7ce  ===> Einen Operanden (Adresse) auswerten <===
01468  b7ce
01469  b7ce  a9 00     eval  lda #0
01470  b7d0  85 60           sta to           addrl gelesene Adresse für
                                              Monitor-Befehl
01471  b7d2  85 61           sta to+1         addrh gelesene Adresse für
                                              Monitor-Befehl
01472  b7d4  85 62           sta to+2         bank gelesene Adresse für
                                              Monitor-Befehl
01473  b7d6  8d b4 0a        sta count        Zwischenzähler für
                                              Block-Operationen
01474  b7d9  8a              txa              X-Register
01475  b7da  48              pha              auf Stapel retten
01476  b7db  98              tya              Y-Register
01477  b7dc  48              pha              auf Stapel retten
01478  b7dd  20 e9 b8  eval1 jsr gnc          Ein Zeichen aus Eingabe-Puffer
                                              holen
01479  b7e0  d0 03           bne eval2        Auf gültige Basis-Angabe prüfen
01480  b7e2  4c 7e b8        jmp evalok       Fehlerflag nach Auswertung
                                              zurücksetzen
01481  b7e5
01482  b7e5
01483  b7e5  ===> Auf gültige Basis-Angabe prüfen <===
01484  b7e5
01485  b7e5  c9 20     eval2 cmp #' '         Space ?
01486  b7e7  f0 f4           beq eval1        Ja, überlesen ->
01487  b7e9  a2 03           ldx #3           Anzahl Basis-Umrechnungsbefehle
01488  b7eb  dd f5 b0  eval3 cmp cmdnum,x     Tabelle der Umrechnungs-Befehle
01489  b7ee  f0 06           beq eval4        Umrechnungskonstanten setzen
01490  b7f0  ca              dex
01491  b7f1  10 f8           bpl eval3
01492  b7f3  e8              inx              Standard ist Hexa-Dezimal !
01493  b7f4  c6 7a           dec txtptr       Zeiger in aktuellen Monitor-Befehl
01494  b7f6
01495  b7f6
01496  b7f6  ===> Umrechnungskonstanten setzen <===
01497  b7f6
```

```
01498  b7f6  bc 8a b8   eval4  ldy bases,x      Tabelle der Zahlensysteme für
                                                Umrechnungen
01499  b7f9  bd 8e b8          lda shiftb,x     Tabelle der Shift-Faktoren für
                                                Umrechnungen
01500  b7fc  8d b6 0a          sta shift        Hilfszähler für Basis-Umrechnungen
01501  b7ff
01502  b7ff

01503  b7ff  ===> Adresswert auswerten <===
01504  b7ff
01505  b7ff  20 e9 b8   eval5  jsr gnc          Ein Zeichen aus Eingabe-Puffer
                                                holen
01506  b802  f0 7a             beq evalok       Fehlerflag nach Auswertung
                                                zurücksetzen
01507  b804  38                sec
01508  b805  e9 30             sbc #'0'         ASCII-Null subtrahieren
01509  b807  90 75             bcc evalok       Fehlerflag nach Auswertung
                                                zurücksetzen
01510  b809  c9 0a             cmp #10          Ziffer 0 - 9 ?
01511  b80b  90 06             bcc eval6        Ja -> weiter
01512  b80d  e9 07             sbc #7           Buchstaben korrigieren
01513  b80f  c9 10             cmp #16          Buchstabe a - f ?
01514  b811  b0 6b             bcs evalok       Fehlerflag nach Auswertung
                                                zurücksetzen
01515  b813  8d b5 0a   eval6  sta number       aktuelle Ziffer
01516  b816  cc b5 0a          cpy number       Vergleichen mit Zahlenbasis
01517  b819  90 61             bcc evalng       wenn größer -> Fehler
01518  b81b  f0 5f             beq evalng       wenn gleich -> Fehler
01519  b81d  ee b4 0a          inc count        Zwischenzähler für
                                                Block-Operationen
01520  b820  c0 0a             cpy #10          Dezimalwert (Basis 10) ?
01521  b822  d0 0a             bne eval8        Umrechnung in Hexawerte durchführen
01522  b824  a2 02             ldx #2
01523  b826  b5 60      eval7  lda to,x         addrl gelesene Adresse für
                                                Monitor-Befehl
01524  b828  9d b7 0a          sta temps,x      Zwischenspeicher
01525  b82b  ca                dex
01526  b82c  10 f8             bpl eval7
01527  b82e
01528  b82e
01529  b82e  ===> Umrechnung in Hexawerte durchführen <===
01530  b82e
01531  b82e  ae b6 0a   eval8  ldx shift        Hilfszähler für Basis-Umrechnungen
01532  b831
01533  b831
01534  b831  ===> Wert mit zwei multiplizieren <===
01535  b831
01536  b831  06 60      eval9  asl to           addrl gelesene Adresse für
                                                Monitor-Befehl
01537  b833  26 61             rol to+1         addrh gelesene Adresse für
                                                Monitor-Befehl
01538  b835  26 62             rol to+2         bank gelesene Adresse für
                                                Monitor-Befehl
01539  b837  b0 43             bcs evalng       Fehlerflag während Auswertung
                                                setzen
01540  b839  ca                dex
01541  b83a  d0 f5             bne eval9        Wert mit zwei multiplizieren
01542  b83c  c0 0a             cpy #10          Dezimalwert (Basis 10) ?
```

```
01543  b83e  d0 22          bne evala       Neue Ziffer hinzuaddieren
01544  b840  0e b7 0a       asl temps       Zwischenspeicher
01545  b843  2e b8 0a       rol temps+1     Zwischenspeicher
01546  b846  2e b9 0a       rol temps+2     Zwischenspeicher
01547  b849  b0 31          bcs evalng      Fehlerflag während Auswertung
                                            setzen
01548  b84b  ad b7 0a       lda temps       Zwischenspeicher

01549  b84e  65 60          adc to          addrl gelesene Adresse für
                                            Monitor-Befehl
01550  b850  85 60          sta to          addrl gelesene Adresse für
                                            Monitor-Befehl
01551  b852  ad b8 0a       lda temps+1     Zwischenspeicher
01552  b855  65 61          adc to+1        addrh gelesene Adresse für
                                            Monitor-Befehl
01553  b857  85 61          sta to+1        addrh gelesene Adresse für
                                            Monitor-Befehl
01554  b859  ad b9 0a       lda temps+2     Zwischenspeicher
01555  b85c  65 62          adc to+2        bank gelesene Adresse für
                                            Monitor-Befehl
01556  b85e  85 62          sta to+2        bank gelesene Adresse für
                                            Monitor-Befehl
01557  b860  b0 1a          bcs evalng      Fehlerflag während Auswertung
                                            setzen
01558  b862
01559  b862
01560  b862  ===> Neue Ziffer hinzuaddieren <===
01561  b862
01562  b862  18          evala  clc
01563  b863  ad b5 0a           lda number   Hilfsadresse Monitor
01564  b866  65 60              adc to       addrl gelesene Adresse für
                                            Monitor-Befehl
01565  b868  85 60              sta to       addrl gelesene Adresse für
                                            Monitor-Befehl
01566  b86a  8a                 txa
01567  b86b  65 61              adc to+1     addrh gelesene Adresse für
                                            Monitor-Befehl
01568  b86d  85 61              sta to+1     addrh gelesene Adresse für
                                            Monitor-Befehl
01569  b86f  8a                 txa
01570  b870  65 62              adc to+2     bank gelesene Adresse für
                                            Monitor-Befehl
01571  b872  85 62              sta to+2     bank gelesene Adresse für
                                            Monitor-Befehl
01572  b874  b0 06              bcs evalng   Fehlerflag während Auswertung
                                            setzen
01573  b876  29 f0              and #%11110000
01574  b878  d0 02              bne evalng   Fehlerflag während Auswertung
                                            setzen
01575  b87a  f0 83              beq eval5    Adresswert auswerten
01576  b87c
01577  b87c
01578  b87c  ===> Fehlerflag während Auswertung setzen <===
01579  b87c
01580  b87c  38          evalng sec
01581  b87d  24                 .byte skip1
01582  b87e
01583  b87e
```

```
01584  b87e  ===> Fehlerflag nach Auswertung zurücksetzen <===
01585  b87e
01586  b87e  18          evalok clc
01587  b87f  8c b6 0a           sty shift        Hilfszähler für Basis-Umrechnungen
01588  b882  68                 pla
01589  b883  a8                 tay              Y-Register vom Stapel laden
01590  b884  68                 pla
01591  b885  aa                 tax              X-Register vom Stapel laden
01592  b886  ad b4 0a           lda count        Zwischenzähler für
                                                 Block-Operationen
01593  b889  60                 rts
01594  b88a
01595  b88a
01596  b88a  ===> Tabelle der Zahlensysteme für Umrechnungen <===
01597  b88a
01598  b88a  10          bases  .byte 16         Hexadezimal
01599  b88b  0a                 .byte 10         Dezimal
01600  b88c  08                 .byte 8          Oktal
01601  b88d  02                 .byte 2          Binär
01602  b88e
01603  b88e
01604  b88e  ===> Tabelle der Shift-Faktoren für Umrechnungen <===
01605  b88e
01606  b88e  04 03 03    shiftb .byte 4, 3, 3, 1
01607  b891  01
01608  b892
01609  b892
01610  b892  ===> Adresse als Hex-Bytes ausgeben <===
01611  b892
01612  b892  a5 68       putt2  lda t2+2          bank aktuelle Adresse
01613  b894  20 d2 b8           jsr makhex       Byte in AC in Hexa umwandeln nach
                                                 AC und XR
01614  b897  8a                 txa              Low-Nybble in Accu
01615  b898  20 d2 ff           jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                 Kanal (Sprungv.)
01616  b89b  a5 66              lda t2           addrl aktuelle Adresse
01617  b89d  a6 67              ldx t2+1         addrh aktuelle Adresse
01618  b89f
01619  b89f
01620  b89f  ===> Adresse in Accu und X-Reg. als Hex-Bytes ausgeben <===
01621  b89f
01622  b89f  48          putwrd pha              Accu retten
01623  b8a0  8a                 txa              X-Register nach Accu
01624  b8a1  20 c2 b8           jsr puthex       Hex-Byte in AC als 2 ASCII-Zeichen
                                                 ausgeben
01625  b8a4  68                 pla              Accu zurück
01626  b8a5
01627  b8a5
01628  b8a5  ===> Accu als Hex-Bytes ausgeben + SPACE <===
01629  b8a5
01630  b8a5  20 c2 b8    puthxs jsr puthex       Hex-Byte in AC als 2 ASCII-Zeichen
                                                 ausgeben
01631  b8a8
01632  b8a8
01633  b8a8  ===> Ausgabe 'space' <===
01634  b8a8
01635  b8a8  a9 20       putspc lda #' '         Space
```

```
01636  b8aa  4c d2 ff          jmp kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01637  b8ad
01638  b8ad
01639  b8ad  ===> Ausgabe 'carriage return + cursur aufwärts' <===
01640  b8ad
01641  b8ad  20 7d ff   cronly jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
01642  b8b0
01643  b8b0  0d 91 00          .byte cr ,$91, eom
01644  b8b3
01645  b8b3  60                rts
01646  b8b4
01647  b8b4
01648  b8b4  ===> Ausgabe 'carriage return' <===
01649  b8b4
01650  b8b4  a9 0d      crlf   lda #cr          Carriage Return
01651  b8b6  4c d2 ff          jmp kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01652  b8b9
01653  b8b9
01654  b8b9  ===> Ausgabe 'carriage  return' und Zeile löschen <===
01655  b8b9
01656  b8b9  20 7d ff   newlin jsr kprimm       folg. Text ausgeben und
                                                überspringen (Sprungv.)
01657  b8bc
01658  b8bc  0d 1b 51          .byte cr , esc,'q ', eom
01659  b8bf  20 00
01660  b8c1
01661  b8c1  60                rts
01662  b8c2
01663  b8c2
01664  b8c2  ===> Hex-Byte in AC als 2 ASCII-Zeichen ausgeben <===
01665  b8c2
01666  b8c2  8e af 0a   puthex stx sxreg        Rettungsbyte für X-Register
01667  b8c5  20 d2 b8          jsr makhex       Byte in AC in Hexa umwandeln nach
                                                AC und XR
01668  b8c8  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01669  b8cb  8a                txa              zweites Nybble nach Accu
01670  b8cc  ae af 0a          ldx sxreg        Rettungsbyte für X-Register
01671  b8cf  4c d2 ff          jmp kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01672  b8d2
01673  b8d2
01674  b8d2
01675  b8d2  ===> Byte in AC in Hexa umwandeln nach AC und XR <===
01676  b8d2
01677  b8d2  48         makhex pha              Byte  retten
01678  b8d3  20 dc b8          jsr makhx2       Ein Nybble in Hexa-ASCII wandeln
01679  b8d6  aa                tax              Nach X-Register retten
01680  b8d7  68                pla              Byte zurück
01681  b8d8  4a                lsr a            Oberes Nybble -> Unteres Nybble
01682  b8d9  4a                lsr a
01683  b8da  4a                lsr a
01684  b8db  4a                lsr a
01685  b8dc
01686  b8dc
```

```
01687  b8dc  ===> Ein Nybble in Hexa-ASCII wandeln <===
01688  b8dc
01689  b8dc  29 0f      makhx2 and #%00001111   unteres Nybble isolieren
01690  b8de  c9 0a             cmp #10          größer als 9 ?
01691  b8e0  90 02             bcc makhx3       Nein -> Okay, Ziffer 0 - 9
01692  b8e2  69 06             adc #6           6 addieren (=7, da Carry!)
01693  b8e4  69 30      makhx3 adc #'0'         Nach ASCII wandeln
01694  b8e6  60                rts
01695  b8e7
01696  b8e7
01697  b8e7  ===> Letztes Zeichen aus Eingabepuffer nochmals lesen <===
01698  b8e7
01699  b8e7  c6 7a       glc   dec txtptr       Zeiger in aktuellen Monitor-Befehl
01700  b8e9
01701  b8e9
01702  b8e9  ===> Ein Zeichen aus Eingabe-Puffer holen <===
01703  b8e9
01704  b8e9  8e af 0a    gnc   stx sxreg        Rettungsbyte für X-Register
01705  b8ec  a6 7a             ldx txtptr       Zeiger in aktuellen Monitor-Befehl
01706  b8ee  bd 00 02          lda buf,x        BASIC-Eingabepuffer
01707  b8f1  f0 06             beq gnc1         Kein Zeichen -> weiter
01708  b8f3  c9 3a             cmp #':'         Doppelpunkt ?
01709  b8f5  f0 02             beq gnc1         Ja -> weiter
01710  b8f7  c9 3f             cmp #'?'         Fragezeichen ?
01711  b8f9  08          gnc1  php              Bedingungsflags retten
01712  b8fa  e6 7a             inc txtptr       Zeiger in aktuellen Monitor-Befehl
01713  b8fc  ae af 0a          ldx sxreg        Rettungsbyte für X-Register
01714  b8ff  28                plp              Bedingungsflags zurück
01715  b900  60                rts
01716  b901
01717  b901
01718  b901  ===> Zeiger kopieren <===
01719  b901
01720  b901  a5 60       totot2 lda to           addrl gelesene Adresse für
                                                 Monitor-Befehl
01721  b903  85 66             sta t2           addrl aktuelle Adresse
01722  b905  a5 61             lda to+1         addrh gelesene Adresse für
                                                 Monitor-Befehl
01723  b907  85 67             sta t2+1         addrh aktuelle Adresse
01724  b909  a5 62             lda to+2         bank gelesene Adresse für
                                                 Monitor-Befehl
01725  b90b  85 68             sta t2+2         bank aktuelle Adresse
01726  b90d  60                rts
01727  b90e
01728  b90e
01729  b90e  ===> Zeiger von aktueller Adresse subtrahieren <===
01730  b90e
01731  b90e  38          subom2 sec
01732  b90f  a5 60             lda to           addrl gelesene Adresse für
                                                 Monitor-Befehl
01733  b911  e5 66             sbc t2           addrl aktuelle Adresse
01734  b913  85 60             sta to           addrl gelesene Adresse für
                                                 Monitor-Befehl
01735  b915  a5 61             lda to+1         addrh gelesene Adresse für
                                                 Monitor-Befehl
01736  b917  e5 67             sbc t2+1         addrh aktuelle Adresse
01737  b919  85 61             sta to+1         addrh gelesene Adresse für
                                                 Monitor-Befehl
```

```
01738  b91b  a5 62           lda to+2        bank gelesene Adresse für
                                             Monitor-Befehl
01739  b91d  e5 68           sbc t2+2        bank aktuelle Adresse
01740  b91f  85 62           sta to+2        bank gelesene Adresse für
                                             Monitor-Befehl
01741  b921  60              rts
01742  b922
01743  b922

01744  b922  ===> Adresse um 1 vermindern <===
01745  b922
01746  b922  a9 01     decto  lda #1
01747  b924
01748  b924
01749  b924  ===> Adresse um AC vermindern <===
01750  b924
01751  b924  8d af 0a  subto  sta sxreg       Rettungsbyte für X-Register
01752  b927  38               sec
01753  b928  a5 60            lda to          addrl gelesene Adresse für
                                             Monitor-Befehl
01754  b92a  ed af 0a         sbc sxreg       Rettungsbyte für X-Register
01755  b92d  85 60            sta to          addrl gelesene Adresse für
                                             Monitor-Befehl
01756  b92f  a5 61            lda to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
01757  b931  e9 00            sbc #0
01758  b933  85 61            sta to+1        addrh gelesene Adresse für
                                             Monitor-Befehl
01759  b935  a5 62            lda to+2        bank gelesene Adresse für
                                             Monitor-Befehl
01760  b937  e9 00            sbc #0
01761  b939  85 62            sta to+2        bank gelesene Adresse für
                                             Monitor-Befehl
01762  b93b  60               rts
01763  b93c
01764  b93c
01765  b93c  ===> Hilfszeiger um 1 vermindern <===
01766  b93c
01767  b93c  38        dect1  sec
01768  b93d  a5 63            lda t1          addrl Hilfszeiger Monitor
01769  b93f  e9 01            sbc #1
01770  b941  85 63            sta t1          addrl Hilfszeiger Monitor
01771  b943  a5 64            lda t1+1        addrh Hilfzeiger Monitor
01772  b945  e9 00            sbc #0
01773  b947  85 64            sta t1+1        addrh Hilfzeiger Monitor
01774  b949  a5 65            lda t1+2        bank Hilfszeiger Monitor
01775  b94b  e9 00            sbc #0
01776  b94d  85 65            sta t1+2        bank Hilfszeiger Monitor
01777  b94f  60               rts
01778  b950
01779  b950
01780  b950  ===> Zeiger um 1 erhöhen <===
01781  b950
01782  b950  a9 01     inct2  lda #1
01783  b952
01784  b952
01785  b952  ===> Zeiger um AC erhöhen <===
01786  b952
```

```
01787  b952  18          addt2  clc
01788  b953  65 66              adc t2          addrl aktuelle Adresse
01789  b955  85 66              sta t2          addrl aktuelle Adresse
01790  b957  90 06              bcc addt2x
01791  b959  e6 67              inc t2+1        addrh aktuelle Adresse
01792  b95b  d0 02              bne addt2x
01793  b95d  e6 68              inc t2+2        bank aktuelle Adresse
01794  b95f  60          addt2x rts
01795  b960
01796  b960
01797  b960  ===> Aktuelle Adresse um 1 vermindern <===
01798  b960
01799  b960  38          dect2  sec
01800  b961  a5 66              lda t2          addrl aktuelle Adresse
01801  b963  e9 01              sbc #1
01802  b965  85 66              sta t2          addrl aktuelle Adresse
01803  b967  a5 67              lda t2+1        addrh aktuelle Adresse
01804  b969  e9 00              sbc #0
01805  b96b  85 67              sta t2+1        addrh aktuelle Adresse
01806  b96d  a5 68              lda t2+2        bank aktuelle Adresse
01807  b96f  e9 00              sbc #0
01808  b971  85 68              sta t2+2        bank aktuelle Adresse
01809  b973  60                 rts
01810  b974
01811  b974
01812  b974  ===> Adresse in Prozessor-Register kopieren <===
01813  b974
01814  b974  b0 0c       totopc bcs totopx      Wenn Fehlerflag gesetzt ->
                                                Ignorieren
01815  b976  a5 60              lda to          addrl gelesene Adresse für
                                                Monitor-Befehl
01816  b978  a4 61              ldy to+1        addrh gelesene Adresse für
                                                Monitor-Befehl
01817  b97a  a6 62              ldx to+2        bank gelesene Adresse für
                                                Monitor-Befehl
01818  b97c  85 04              sta pclo        8502 CPU PC lo
01819  b97e  84 03              sty pchi        8502 CPU PC hi
01820  b980  86 02              stx bank        Banknummer
01821  b982  60          totopx rts
01822  b983
01823  b983
01824  b983  ===> Adresse und Blocklänge holen <===
01825  b983
01826  b983  b0 2a       range  bcs ranger      Fehlerflag gesetzt -> ignorieren
01827  b985  20 01 b9           jsr totot2      Zeiger kopieren
01828  b988  20 a7 b7           jsr parse       Operanden ermitteln u. auf
                                                gült.Trennnzeichen prüfen
01829  b98b  b0 22              bcs ranger      Fehler -> Abbruch
01830  b98d  a5 60              lda to          addrl gelesene Adresse für
                                                Monitor-Befehl
01831  b98f  8d b7 0a           sta temps       Zwischenspeicher
01832  b992  a5 61              lda to+1        addrh gelesene Adresse für
                                                Monitor-Befehl
01833  b994  8d b8 0a           sta temps+1     Zwischenspeicher
01834  b997  a5 62              lda to+2        bank gelesene Adresse für
                                                Monitor-Befehl
01835  b999  8d b9 0a           sta temps+2     Zwischenspeicher
```

```
01836  b99c  20 0e b9          jsr subom2      Zeiger von aktueller Adresse
                                               subtrahieren
01837  b99f  a5 60             lda to          addrl gelesene Adresse für
                                               Monitor-Befehl
01838  b9a1  85 63             sta t1          addrl Hilfszeiger Monitor
01839  b9a3  a5 61             lda to+1        addrh gelesene Adresse für
                                               Monitor-Befehl
01840  b9a5  85 64             sta t1+1        addrh Hilfzeiger Monitor
01841  b9a7  a5 62             lda to+2        bank gelesene Adresse für
                                               Monitor-Befehl
01842  b9a9  85 65             sta t1+2        bank Hilfszeiger Monitor
01843  b9ab  90 02             bcc ranger      Fehler ->
01844  b9ad  18                clc             Fehlerflag löschen
01845  b9ae  24                .byte skip1
01846  b9af
01847  b9af  38          ranger sec            Fehlerflag setzen
01848  b9b0  60                 rts
01849  b9b1
01850  b9b1
01851  b9b1  ===> Basis-Umrechnungsbefehle ausführen <===
01852  b9b1
01853  b9b1  20 a5 b7   convrt jsr pargot      Zeiger rücksetzen, Operanden
                                               ermitteln
01854  b9b4  20 b9 b8          jsr newlin      Ausgabe 'carriage  return' und
                                               Zeile löschen
01855  b9b7  a9 24             lda #'$'        Prefix für Hexa ausgeben
01856  b9b9  20 d2 ff          jsr kbsout      Ausgabe ac üb. Vector auf akt.
                                               Kanal (Sprungv.)
01857  b9bc  a5 62             lda to+2        bank gelesene Adresse für
                                               Monitor-Befehl
01858  b9be  f0 07             beq cnvrt1      Aktuellen Wert in Hexa asgeben
01859  b9c0  20 d2 b8          jsr makhex      Byte in AC in Hexa umwandeln nach
                                               AC und XR
01860  b9c3  8a                txa
01861  b9c4  20 d2 ff          jsr kbsout      Ausgabe ac üb. Vector auf akt.
                                               Kanal (Sprungv.)
01862  b9c7
01863  b9c7
01864  b9c7  ===> Aktuellen Wert in Hexa ausgeben <===
01865  b9c7
01866  b9c7  a5 60      cnvrt1 lda to          addrl gelesene Adresse für
                                               Monitor-Befehl
01867  b9c9  a6 61             ldx to+1        addrh gelesene Adresse für
                                               Monitor-Befehl
01868  b9cb  20 9f b8          jsr putwrd      Adresse in Accu und X-Reg. als
                                               Hex-Bytes ausgeben
01869  b9ce
01870  b9ce  ===> Aktuellen Wert in Dezimal ausgeben <===
01871  b9ce
01872  b9ce  20 b9 b8          jsr newlin      Ausgabe 'carriage  return' und
                                               Zeile löschen
01873  b9d1  a9 2b             lda #'+'        Prefix für Dezimal
01874  b9d3  20 d2 ff          jsr kbsout      Ausgabe ac üb. Vector auf akt.
                                               Kanal (Sprungv.)
01875  b9d6  20 07 ba          jsr bindec      Umwandlung Hexa-Dezimal -> Dezimal
01876  b9d9  a9 00             lda #0          Flag für Nullenunterdrückung setzen
01877  b9db  a2 08             ldx #8          Anzahl Ziffern
01878  b9dd  a0 03             ldy #3          Anzahl Bits/Ziffer -1
```

```
01879  b9df  20 5d ba          jsr unpack       Wert in gewünschtem Zahlensystem
                                                ausgeben
01880  b9e2
01881  b9e2  ===> Aktuellen Wert in Oktal ausgeben <===
01882  b9e2
01883  b9e2  20 b9 b8          jsr newlin       Ausgabe 'carriage  return' und
                                                Zeile löschen
01884  b9e5  a9 26             lda #'&'         Prefix für Oktal
01885  b9e7  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01886  b9ea  a9 00             lda #0           Flag für Nullenunterdrückung setzen
01887  b9ec  a2 08             ldx #8           Anzahl Ziffern
01888  b9ee  a0 02             ldy #2           Anzahl Bits/Ziffer -1
01889  b9f0  20 47 ba          jsr unpkto       Aktuellen Wert kopieren u.um eine
                                                Stelle erweitern
01890  b9f3
01891  b9f3  ===> Aktuellen Wert in Binär ausgeben <===
01892  b9f3
01893  b9f3  20 b9 b8          jsr newlin       Ausgabe 'carriage  return' und
                                                Zeile löschen
01894  b9f6  a9 25             lda #'%'         Prefix für Binär
01895  b9f8  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
01896  b9fb  a9 00             lda #0           Flag für Nullenunterdrückung setzen
01897  b9fd  a2 18             ldx #24          Anzahl Ziffern
01898  b9ff  a0 00             ldy #0           Aznahl Bits/Ziffer -1
01899  ba01  20 47 ba          jsr unpkto       Aktuellen Wert kopieren u.um eine
                                                Stelle erweitern
01900  ba04  4c 8b b0          jmp main         Monitor-Warteschleife
01901  ba07
01902  ba07
01903  ba07  ===> Umwandlung Hexa-Dezimal -> Dezimal <===
01904  ba07
01905  ba07  20 01 b9   bindec jsr totot2       Zeiger kopieren
01906  ba0a  a9 00              lda #0
01907  ba0c  a2 07              ldx #7          Anzahl Bytes im Zwischenspeicher
01908  ba0e
01909  ba0e
01910  ba0e  ===> Zwischenspeicher löschen <===
01911  ba0e
01912  ba0e  9d a0 0a   bindc1 sta hulp,x       Monitor-Hilfsbereich
01913  ba11  ca                dex
01914  ba12  10 fa             bpl bindc1      Zwischenspeicher löschen
01915  ba14  ee a7 0a          inc hulp2+3     Bitwert = 1
01916  ba17  a0 17             ldy #23         Anzahl Datenbits -1
01917  ba19  08                php             Original-Bedingungflags merken
01918  ba1a  78                sei             Interupt sperren
01919  ba1b  f8                sed             Dezimal-Modus setzen !
01920  ba1c  46 68      bindc2 lsr t2+2        Zu wandelnden Wert um
01921  ba1e  66 67             ror t2+1        1 Bit nach links schieben
01922  ba20  66 66             ror t2
01923  ba22  90 0f             bcc bindc4      Bit war Null -> weiter
01924  ba24  18                clc
01925  ba25  a2 03             ldx #$03        Aktuellen Bitwert
01926  ba27  bd a4 0a   bindc3 lda hulp2,x     auf Ergebnis
01927  ba2a  7d a0 0a          adc hulp,x      addieren
01928  ba2d  9d a0 0a          sta hulp,x
01929  ba30  ca                dex
```

```
01930  ba31  10 f4           bpl bindc3
01931  ba33  18       bindc4 clc              Bitwert mit zwei
01932  ba34  a2 03           ldx #3           multiplizieren
01933  ba36  bd a4 0a  bindc5 lda hulp2,x      Monitor-Hilfsbereich 2
01934  ba39  7d a4 0a         adc hulp2,x      Monitor-Hilfsbereich 2
01935  ba3c  9d a4 0a         sta hulp2,x      Monitor-Hilfsbereich 2
01936  ba3f  ca               dex
01937  ba40  10 f4           bpl bindc5
01938  ba42  88               dey              Noch Bits vorhanden ?
01939  ba43  10 d7           bpl bindc2       Ja -> weiter
01940  ba45  28               plp              Original-Bedingungsflags zurück
01941  ba46  60               rts              damit Dezimalmodus wieder gelöscht!
01942  ba47
01943  ba47
01944  ba47  ===> Aktuellen Wert kopieren u.um eine Stelle erweitern <===
01945  ba47
01946  ba47  48         unpkto pha
01947  ba48  a5 60            lda to           addrl gelesene Adresse für
                                               Monitor-Befehl
01948  ba4a  8d a2 0a         sta hulp+2       Monitor-Hilfsbereich
01949  ba4d  a5 61            lda to+1         addrh gelesene Adresse für
                                               Monitor-Befehl
01950  ba4f  8d a1 0a         sta hulp+1       Monitor-Hilfsbereich
01951  ba52  a5 62            lda to+2         bank gelesene Adresse für
                                               Monitor-Befehl
01952  ba54  8d a0 0a         sta hulp         Monitor-Hilfsbereich
01953  ba57  a9 00            lda #0
01954  ba59  8d a3 0a         sta hulp+3       Monitor-Hilfsbereich
01955  ba5c  68               pla
01956  ba5d
01957  ba5d
01958  ba5d  ===> Wert in gewünschtem Zahlensystem ausgeben <===
01959  ba5d
01960  ba5d  8d b4 0a   unpack sta count       Flag für Nullenunterdrückung
01961  ba60  8c b6 0a         sty shift        Anzahl Bits/Ziffer -1
01962  ba63  ac b6 0a   unpck1 ldy shift       Anzahl Bits
01963  ba66  a9 00            lda #0           Accu löschen
01964  ba68
01965  ba68
01966  ba68  ===> Wert durch zwei teilen <===
01967  ba68
01968  ba68  0e a3 0a   unpck2 asl hulp+3       Monitor-Hilfsbereich
01969  ba6b  2e a2 0a         rol hulp+2       Monitor-Hilfsbereich
01970  ba6e  2e a1 0a         rol hulp+1       Monitor-Hilfsbereich
01971  ba71  2e a0 0a         rol hulp         Monitor-Hilfsbereich
01972  ba74  2a               rol a            Bit in Accu einschieben
01973  ba75  88               dey              Anzahl bits -1
01974  ba76  10 f0            bpl unpck2       Wert durch zwei teilen
01975  ba78  a8               tay              Errechnete Ziffer in y-Register
01976  ba79  d0 09            bne unpck3       Wenn nicht Null -> ausgeben
01977  ba7b  e0 01            cpx #1           Letze auszugebende Stelle ?
01978  ba7d  f0 05            beq unpck3       Ja -> null ausgeben
01979  ba7f  ac b4 0a         ldy count        Nullen unterdrücken ?
01980  ba82  f0 08            beq unpck4       Ja -> Null nicht ausgeben
01981  ba84
01982  ba84
01983  ba84  ===> Eine Ziffer ausgeben <===
01984  ba84
```

```
01985  ba84  ee b4 0a   unpck3 inc count       Zwischenzähler für
                                               Block-Operationen
01986  ba87  09 30             ora #'0'        In ASCII wandeln
01987  ba89  20 d2 ff          jsr kbsout      Ausgabe ac üb. Vector auf akt.
                                               Kanal (Sprungv.)
01988  ba8c
01989  ba8c
01990  ba8c  ===> Test auf Ende der Ausgabe <===
01991  ba8c
01992  ba8c  ca         unpck4 dex             Ziffernzähler vermindern
01993  ba8d  d0 d4             bne unpck1      Nicht Null -> nächste Ziffer
                                               ausgeben
01994  ba8f  60                rts
01995  ba90
01996  ba90
01997  ba90  ===> Floppy-Kommando übergeben <===
01998  ba90
01999  ba90  d0 03      disk   bne disk1       Prüfen auf gültige Gerätenummer
02000  ba92  a2 08             ldx #8          Standardgerät ist 8!
02001  ba94  2c                .byte skip2
02002  ba95
02003  ba95
02004  ba95  ===> Prüfen auf gültige Gerätenummer <===
02005  ba95
02006  ba95  a6 60      disk1  ldx to          addrl gelesene Adresse für
                                               Monitor-Befehl
02007  ba97  e0 04             cpx #4          Gerät < 4 ?
02008  ba99  90 65             bcc diserr      Ja, nicht zulässig -> Fehler
02009  ba9b  e0 1f             cpx #31         Größer 30 ?
02010  ba9d  b0 61             bcs diserr      Ja, nicht zulässig -> Fehler
02011  ba9f  86 60             stx to          Gerätenummer merken
02012  baa1  a9 00             lda #0
02013  baa3  85 62             sta to+2        bank gelesene Adresse für
                                               Monitor-Befehl
02014  baa5  85 b7             sta fnlen       Länge akt. Filename
02015  baa7  aa                tax
02016  baa8  20 68 ff          jsr kstbnk      Bank für load/sav/verify setzen
                                               (Sprungv.)
02017  baab  20 e9 b8          jsr gnc         Ein Zeichen aus Eingabe-Puffer
                                               holen
02018  baae  c6 7a             dec txtptr      Zeiger in aktuellen Monitor-Befehl
02019  bab0  c9 24             cmp #'$'        Inhaltsverzeichnis gefordert ?
02020  bab2  f0 4f             beq disdir      Disketten-Inhaltsverzeichnis
                                               anzeigen
02021  bab4  a9 00             lda #0          Logische Dateinummer Null!
02022  bab6  a6 60             ldx to          Geräteadresse
02023  bab8  a0 0f             ldy #15         Sekundäradresse 15
02024  baba  20 ba ff          jsr kstlfs      la, fa, sa aus ac, xr, yr setzen
                                               (Sprungv.)
02025  babd  20 c0 ff          jsr kopen       log. File üb. Vector öffnen,
                                               Ausgabe fn (sprungv.)
02026  bac0  b0 32             bcs disdon      Fehler -> Disk-Abschlußroutine
02027  bac2  a2 00             ldx #0          Datei 'Null' als Ausgabekanal
                                               eröffnen
02028  bac4  20 c9 ff          jsr kckout      Ausgabe-Kanal üb. Vector öffnen
                                               (Sprungv.)
02029  bac7  b0 2b             bcs disdon      Disk-Abschlußroutine
02030  bac9
```

```
02031  bac9
02032  bac9  ===> Befehl an Floppy übermitteln <===
02033  bac9
02034  bac9  a6 7a      disk2  ldx txtptr      Zeiger in aktuellen Monitor-Befehl
02035  bacb  e6 7a             inc txtptr      Zeiger in aktuellen Monitor-Befehl
02036  bacd  bd 00 02          lda buf,x       Zeichen aus Puffer
02037  bad0  f0 05             beq diskst      Wenn Null, Ende ->
02038  bad2  20 d2 ff          jsr kbsout      Zeichen an Floppy senden
02039  bad5  90 f2             bcc disk2       Befehl an Floppy übermitteln
02040  bad7
02041  bad7
02042  bad7  ===> Disk-Status anzeigen <===
02043  bad7
02044  bad7  20 cc ff   diskst jsr kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                               schließen (Sprungv.)
02045  bada  20 b4 b8          jsr crlf        Ausgabe 'carriage return'
02046  badd  a2 00             ldx #0          Datei 'Null' als Eingabekanal
                                               setzen
02047  badf  20 c6 ff          jsr kchkin      Eingabekanal üb. Vector öffnen
                                               (Sprungv.)
02048  bae2  b0 10             bcs disdon      Fehler -> Disk-Abschlußroutine
02049  bae4  20 cf ff   dskst1 jsr kbasin      Zeichen üb. Vector vom akt. Kanal
                                               nach ac (Sprungv.)
02050  bae7  20 d2 ff          jsr kbsout      Ausgabe ac üb. Vector auf akt.
                                               Kanal (Sprungv.)
02051  baea  c9 0d             cmp #cr         Carriage Return ?
02052  baec  f0 06             beq disdon      Ja -> Disk-Abschlußroutine
02053  baee  a5 90             lda status      I/O-Status 'st'
02054  baf0  29 bf             and #%10111111  Fehler aufgetreten ?
02055  baf2  f0 f0             beq dskst1      Nein -> nächstes Zeichen
02056  baf4
02057  baf4
02058  baf4  ===> Disk-Abschlußroutine <===
02059  baf4
02060  baf4  20 cc ff   disdon jsr kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                               schließen (Sprungv.)
02061  baf7  a9 00             lda #0          Datei 'Null' schließen
02062  baf9  38                sec
02063  bafa  20 c3 ff          jsr kclose      log. File üb. Vector schließen
                                               (Sprungv.)
02064  bafd  4c 8b b0          jmp main        Monitor-Warteschleife
02065  bb00
02066  bb00  4c bc b0   diserr jmp error       Fehlerbehandlung Diskettenbefehle
02067  bb03
02068  bb03
02069  bb03  ===> Disketten-Inhaltsverzeichnis anzeigen <===
02070  bb03
02071  bb03  a0 ff      disdir ldy #$ff
02072  bb05  a6 7a             ldx txtptr      Zeiger in aktuellen Monitor-Befehl
02073  bb07  ca                dex
02074  bb08  c8         dsdir1 iny             Länge des Textes
02075  bb09  e8                inx             Zeiger in Monitorbefehl
02076  bb0a  bd 00 02          lda buf,x       Zeichen aus Befehl
02077  bb0d  d0 f9             bne dsdir1      Nicht Null, Zeiger weiter erhöhen
02078  bb0f  98                tya             Länge des Textes in Accu
02079  bb10  a6 7a             ldx txtptr      Low-Byte des Textes
02080  bb12  a0 02             ldy #>buf       High-Byte des Textes
```

```
02081  bb14  20 bd ff          jsr kstnam       fnlen, fnadr/+1 aus ac, xr, yr
                                                setzen (Sprungv.)
02082  bb17  a9 00             lda #0           Datei 'Null' öffnen
02083  bb19  a6 60             ldx to           Gerätenummer
02084  bb1b  a0 60             ldy #$60         Sekundäradresse 'load'
02085  bb1d  20 ba ff          jsr kstlfs       la, fa, sa aus ac, xr, yr setzen
                                                (Sprungv.)
02086  bb20  20 c0 ff          jsr kopen        log. File üb. Vector öffnen,
                                                Ausgabe fn (sprungv.)
02087  bb23  b0 cf             bcs disdon       Disk-Abschlußroutine

02088  bb25  a2 00             ldx #0           Datei 'Null' als Eingabekanal
                                                öffnen
02089  bb27  20 c6 ff          jsr kchkin       Eingabekanal üb. Vector öffnen
                                                (Sprungv.)
02090  bb2a  20 b4 b8          jsr crlf         Ausgabe 'carriage return'
02091  bb2d  a0 03             ldy #3           Zähler für Doppelbyte-Paare
02092  bb2f
02093  bb2f
02094  bb2f  ===> Einen Directory-Eintrag anzeigen <===
02095  bb2f
02096  bb2f  84 63      dsdir2 sty t1           addrl Hilfszeiger Monitor
02097  bb31  20 cf ff   dsdir3 jsr kbasin       Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
02098  bb34  85 60             sta to           Low-Byte des Wertes
02099  bb36  a5 90             lda status       I/O-Status 'st'
02100  bb38  d0 ba             bne disdon       Disk-Abschlußroutine
02101  bb3a  20 cf ff          jsr kbasin       Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
02102  bb3d  85 61             sta to+1         High-Byte des Wertes
02103  bb3f  a5 90             lda status       I/O-Status 'st'
02104  bb41  d0 b1             bne disdon       Disk-Abschlußroutine
02105  bb43  c6 63             dec t1           Zähler vermindern
02106  bb45  d0 ea             bne dsdir3       Nicht null -> weiter
02107  bb47  20 07 ba          jsr bindec       Umwandlung Hexa-Dezimal -> Dezimal
02108  bb4a  a9 00             lda #0           Flag für Nullenunterdrückung setzen
02109  bb4c  a2 08             ldx #8           Anzahl Ziffern
02110  bb4e  a0 03             ldy #3           Anzahl Bits/Ziffer - 1
02111  bb50  20 5d ba          jsr unpack       Wert in gewünschtem Zahlensystem
                                                ausgeben
02112  bb53  a9 20             lda #' '         Space
02113  bb55  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
02114  bb58
02115  bb58
02116  bb58  ===> Dateinamen anzeigen <===
02117  bb58
02118  bb58  20 cf ff   dsdir4 jsr kbasin       Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
02119  bb5b  f0 09             beq dsdir5       Neue Zeile, Test auf Stop
02120  bb5d  a6 90             ldx status       I/O-Status 'st'
02121  bb5f  d0 93             bne disdon       Disk-Abschlußroutine
02122  bb61  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt.
                                                Kanal (Sprungv.)
02123  bb64  90 f2             bcc dsdir4       Dateinamen anzeigen
02124  bb66
02125  bb66
```

```
02126  bb66  ===> Neue Zeile, Test auf Stop <===
02127  bb66
02128  bb66  20 b4 b8   dsdir5 jsr crlf         Ausgabe 'carriage return'
02129  bb69  20 e1 ff          jsr kstop        Stop-Taste üb. Vector abfragen
                                                (Sprungv.)
02130  bb6c  f0 86              beq disdon       Stop gedrückt, Abbruch
02131  bb6e  a0 02              ldy #$02         Zähler für einzelnen Eintrag
02132  bb70  d0 bd              bne dsdir2       Einen Directory-Eintrag anzeigen
02133  bb72
02134  bb72                     *=$bffe
02135  bffe
02136  bffe  00 3a              .byte $00,$3a    Prüfbytes
02137  c000
02138  c000                     .end
02139  c000
02140  c000                     .end

Fehler in Pass 1 = 00000
Fehler in Pass 2 = 00000

Assemblierung Ende
```

3.2 Verweistabelle (Crossreference)

```
addt2  = $b952:   372 ,   954 ,  1033 ,  1787#
addt2x = $b95f:  1790 ,  1792 ,  1794#
aerr   = $b579:   925 ,   927 ,   931 ,   933 ,   978#,   995
areg   =   $06:    20#,   129$
as10   = $b40b:   789#,   802
as100  = $b48f:   856 ,   868#
as110  = $b496:   871#,   996
as20   = $b413:   792#,   806
as210  = $b4b4:   883#,   902
as220  = $b4bd:   887#,   893
as230  = $b4cc:   884 ,   886 ,   894#
as240  = $b4df:   895 ,   899 ,   901#
as250  = $b4e4:   890 ,   907#
as30   = $b41f:   793 ,   795 ,   801#
as300  = $b4ea:   903 ,   909#
as310  = $b4f3:   911 ,   914#
as320  = $b511:   924 ,   929#
as340  = $b519:   928 ,   934#
as40   = $b42b:   810#,   821
as400  = $b521:   918 ,   939#,   942
as420  = $b524:   938 ,   940#
as50   = $b436:   816#,   820
as500  = $b52a:   915 ,   943#
as60   = $b442:   783 ,   823#,   834 ,   839 ,   863
as70   = $b445:   811 ,   829#
as71   = $b447:   830#,   858 ,   862
as80   = $b46b:   843 ,   846#
as90   = $b471:   845 ,   847 ,   849#
as91   = $b473:   850#,   853
as92   = $b47a:   833 ,   854#
as93   = $b47c:   831 ,   855#
assem  = $b406:   274 ,   286 ,   783#
ba     =   $c6:    36#,   660$,   704$
bank   =   $02:    16#,   113$,   138$,   171 ,   182 ,  1820$
bases  = $b88a:  1498 ,  1598#
bindc1 = $ba0e:  1912#,  1914
bindc2 = $ba1c:  1920#,  1939
bindc3 = $ba27:  1926#,  1930
bindc4 = $ba33:  1923 ,  1931#
bindc5 = $ba36:  1933#,  1937
bindec = $ba07:  1875 ,  1905#,  2107
break  = $b009:   100 ,   106#
break1 = $b019:   118#,   121
buf    = $0200:    45#,   195$,   202$,   679 ,  1706 ,  2036 ,  2076 ,  2080
call   = $b021:    99 ,   127#
char1  = $b715:   896 ,  1096 ,  1269#
char2  = $b71b:   898 ,  1098 ,  1279#
cmdchr = $b0e6:   217 ,   257#
cmdnum = $b0f5:   266#,  1488
cmdtbl = $b0fc:   241 ,   243 ,   274#
cmpare = $b13d:   318#
cmparx = $02c8:    47#,   320$,   546$
```

```
cnvrt1 = $b9c7:  1858 ,  1866#
compar = $b231:   275 ,   492#
convrt = $b9b1:   250 ,  1853#
count  = $0ab4:    63#,   791$,   830 ,   844 ,  1447 ,  1463 ,  1473$,  1519$,  1592
                  1960$,  1979 ,  1985$
cr     =   $0d:     9#,   108 ,   141 ,   160 ,   160 ,   199 ,  1025 ,  1428 ,  1643
                  1650 ,  1658 ,  2051
crlf   = $b8b4:   191 ,   444 ,   535 ,   620 ,   707 ,  1650#,  2045 ,  2090 ,  2128
cronly = $b8ad:   945 ,  1641#
dect1  = $b93c:   583 ,   646 ,   771 ,  1767#
dect2  = $b960:   575 ,  1799#
decto  = $b922:   373 ,   574 ,  1746#
direct = $0ab3:    62#,   501$,   534$,   563
dis10  = $b5a3:  1005 ,  1013#
dis100 = $b5ae:  1015 ,  1023#,  1036
dis110 = $b5ce:  1029 ,  1037#
dis20  = $b5a9:  1008 ,  1017#
dis200 = $b5d1:  1018 ,  1039#
dis300 = $b5d4:  1030 ,  1044#
dis400 = $b5dc:   951 ,  1047#
disasm = $b599:   276 ,  1005#
disdir = $bb03:  2020 ,  2071#
disdon = $baf4:  2026 ,  2029 ,  2048 ,  2052 ,  2060#,  2087 ,  2100 ,  2104 ,  2121
                  2130
diserr = $bb00:  2008 ,  2010 ,  2066#
disk   = $ba90:   285 ,  1999#
disk1  = $ba95:  1999 ,  2006#
disk2  = $bac9:  2034#,  2039
diskst = $bad7:  2037 ,  2044#
dmpas1 = $b220:   477 ,   479#
dmpas2 = $b22c:   482 ,   485#
dmpasc = $b213:   472#,   484 ,   486
dmpon1 = $b1f7:   450#,   462
dmpon2 = $b1fa:   449 ,   455#
dmpon3 = $b209:   460 ,   462#
dmpone = $b1e8:   363 ,   423 ,   444#
dsdir1 = $bb08:  2074#,  2077
dsdir2 = $bb2f:  2096#,  2132
dsdir3 = $bb31:  2097#,  2106
dsdir4 = $bb58:  2118#,  2123
dsdir5 = $bb66:  2119 ,  2128#
dset   = $b659:   874 ,  1051 ,  1130#
dskst1 = $bae4:  2049#,  2055
dsperr = $b191:   344 ,   377#
dspme1 = $b15c:   333 ,   337#
dspme2 = $b162:   336 ,   343#
dspme3 = $b16e:   347 ,   352#,   356
dspme4 = $b177:   339 ,   361#,   374
dspme5 = $b186:   366 ,   372#
dspmem = $b152:   281 ,   333#
dspreg = $b050:   158#,   282
dsprg2 = $b080:   182#,   186
eom    =   $00:     8#,   108 ,   141 ,   168 ,   227 ,   420 ,   465 ,   736 ,   984
                  1025 ,  1059 ,  1643 ,  1658
err    = $b677:  1134 ,  1136 ,  1157#
error  = $b0bc:   198 ,   225#,   377 ,   506 ,   569 ,   650 ,   689 ,   775 ,   823
                   978 ,  1039 ,  1457 ,  2066
esc    =   $1b:    10#,   168 ,   420 ,   948 ,  1025 ,  1658
```

```
eval   = $b7ce:   832 ,  1442 ,  1469#
eval1  = $b7dd:  1478#,  1486
eval2  = $b7e5:  1479 ,  1485#
eval3  = $b7eb:  1488#,  1491
eval4  = $b7f6:  1489 ,  1498#
eval5  = $b7ff:  1505#,  1575
eval6  = $b813:  1511 ,  1515#
eval7  = $b826:  1523#,  1526
eval8  = $b82e:  1521 ,  1531#
eval9  = $b831:  1536#,  1541
evala  = $b862:  1543 ,  1562#
evalc1 = $b0b4:   217#,   220
evalcm = $b0b2:   101 ,   216#
evalng = $b87c:  1517 ,  1518 ,  1539 ,  1547 ,  1557 ,  1572 ,  1574 ,  1580#
evalok = $b87e:  1480 ,  1506 ,  1509 ,  1514 ,  1586#
exit   = $b0e3:   252#,   284
fa     =   $ba:    34#,   657$,   700$
fetch  = $b11a:   293#,   455 ,   472 ,   543 ,   630 ,  1050 ,  1067 ,  1087
filerr = $b403:   755 ,   758 ,   760 ,   775#
fill   = $b3db:   277 ,   754#
fill1  = $b3ee:   766#,   772
fill2  = $b400:   769 ,   773#
fnadr  =   $bb:    35#,   665$,   667$,   684$
fnbank =   $c7:    37#,   662$
fnlen  =   $b7:    32#,   661$,   685$,  2014$
format = $0aaa:    56#,   875 ,   887 ,   894$,  1084 ,  1094$,  1161$
getfmt = $b67b:  1156 ,  1159#
glc    = $b8e7:  1444 ,  1699#
gnc    = $b8e9:   207 ,   593 ,   596 ,   605 ,   672 ,   695 ,   792 ,   855 ,  1478
                 1505 ,  1704#,  2017
gnc1   = $b8f9:  1707 ,  1709 ,  1711#
go     = $b1d6:   278 ,   429#
gosub  = $b1df:   280 ,   437#
hulp   = $0aa0:    54#,   790$,   817$,   818$,   836$,   850$,   859$,   990 ,
                 1912$,  1927 ,  1928$,  1948$,  1950$,  1952$,  1954$,  1968$,
                 1969$,  1970$,  1971$
hulp2  = $0aa4:    55#,  1915$,  1926 ,  1933 ,  1934 ,  1935$
hunt   = $b2ce:   279 ,   590#
hunt1  = $b2e3:   603#,   608
hunt2  = $b2f2:   595 ,   610#
hunt3  = $b2f8:   612#,   618
hunt4  = $b307:   606 ,   609 ,   616 ,   619#
hunt5  = $b30c:   625#,   647
hunt6  = $b30e:   630#,   635
hunt7  = $b324:   632 ,   643#
hunt8  = $b331:   644 ,   648#
hunter = $b334:   591 ,   598 ,   650#
ieven  = $b668:  1132 ,  1143#
imoncd = $032e:    49#,   211
inct2  = $b950:   578 ,   645 ,   770 ,  1782#
kbasin = $ffcf:    87#,   194 ,  2049 ,  2097 ,  2101 ,  2118
kbsout = $ffd2:    88#,   174 ,   446 ,   479 ,  1045 ,  1097 ,  1100 ,  1203 ,  1615
                 1636 ,  1651 ,  1668 ,  1671 ,  1856 ,  1861 ,  1874 ,  1885 ,  1895
                 1987 ,  2038 ,  2050 ,  2113 ,  2122
kchkin = $ffc6:    84#,  2047 ,  2089
kckout = $ffc9:    85#,  2028
kclose = $ffc3:    83#,  2063
kclrch = $ffcc:    86#,  2044 ,  2060
```

```
keyd   = $034a:   50#,   959$,   961$,   962$,   965$,   968$,   969$,   972$,
                  973$
kicmpv = $ff7a:   77#,   323 ,   557
kifetv = $ff74:   75#,   297
kistav = $ff77:   76#,   310 ,   551
kjmpfr = $ff71:   74#,   432
kjsrfr = $ff6e:   73#,   438
kloads = $ffd5:   89#,   728
kopen  = $ffc0:   82#,  2025 ,  2086
kprimm = $ff7d:   78#,   106 ,   139 ,   158 ,   225 ,   418 ,   463 ,   734 ,   946
                 1023 ,  1057 ,  1641 ,  1656
ksaves = $ffd8:   90#,   716
kstbnk = $ff68:   72#,  2016
kstlfs = $ffba:   80#,  2024 ,  2085
kstmsg = $ff90:   79#,   152
kstnam = $ffbd:   81#,  2081
kstop  = $ffe1:   91#,   361 ,   541 ,   643 ,   768 ,  1028 ,  2129
length = $0aab:   57#,   885 ,   914 ,   937 ,   952$,   953 ,  1031$,  1032 ,  1034
                 1053 ,  1082 ,  1163$
lodsav = $b337:  248 ,   656#
lodsv1 = $b34e:  672#,   675
lodsv2 = $b35d:  679#,   688
lodsv3 = $b370:  677 ,   689#,   712 ,   726 ,   733
lodsv4 = $b373:  683 ,   694#
lodsv5 = $b3a8:  717#,   731
lodsv6 = $b3ab:  673 ,   680 ,   696 ,   698 ,   702 ,   722#,   749
lodsv7 = $b3b7:  724 ,   728#
lodsv8 = $b3d1:  706 ,   745#
main   = $b08b:  191#,   208 ,   229 ,   375 ,   395 ,   424 ,   439 ,   585 ,   648
                 717 ,   740 ,   773 ,   796 ,   976 ,  1037 ,  1900 ,  2064
main01 = $b092:  194#,   200
main02 = $b0a6:  207#,   210
main1  = $b0c5:  218 ,   234#
main11 = $b0db:  235 ,   247#
main12 = $b0e0:  237 ,   250#
makhex = $b8d2:  172 ,   964 ,   967 ,   971 ,  1613 ,  1667 ,  1677#,  1859
makhx2 = $b8dc: 1678 ,  1689#
makhx3 = $b8e4: 1691 ,  1693#
mmucr  = $ff00:   70#,   128$
mnem1  = $b721:  880 ,  1187 ,  1295#
mnemr  = $b761:  878 ,  1189 ,  1363#
mnndx1 = $b692: 1171#,  1180
mnndx2 = $b696: 1174#,  1177
mnndx3 = $b69d: 1170 ,  1172 ,  1179#
mode   =   $d7:   39#,   364 ,   365 ,   412 ,   459 ,   481
monbrk = $b003:  100#
moncal = $b000:   99#,   133 ,   134
moncmd = $b006:  101#
msal   = $0aac:   58#,   803$,   812
ndx    =   $d0:   38#,   975$
newlin = $b8b9: 1656#,  1854 ,  1872 ,  1883 ,  1893
nmode  = $b6c3: 1145 ,  1215#
nmode2 = $b707: 1160 ,  1251#
number = $0ab5:   64#,  1515$,  1516 ,  1563
pargot = $b7a5:  611 ,  1437#,  1853
parse  = $b7a7:  245 ,   335 ,   388 ,   407 ,   504 ,   615 ,   697 ,   701 ,   705
                 759 ,  1007 ,  1442#,  1828
parse1 = $b7ba: 1445 ,  1451#
```

```
parse2 = $b7c2:  1443 ,  1455#
parse3 = $b7c7:  1449 ,  1459#
parse4 = $b7c9:  1448 ,  1452 ,  1454 ,  1462#
pcadj3 = $b64d:  1108 ,  1118#
pcadj4 = $b653:  1120 ,  1122#
pchi   =   $03:    17#,   119$,   136$,   175 ,  1819$
pclo   =   $04:    18#,   135$,  1818$
pcrts  = $b658:  1123 ,  1125#
pradr1 = $b60f:  1080#,  1102
pradr2 = $b618:  1084#,  1092
pradr3 = $b62a:  1081 ,  1083 ,  1094#
pradr4 = $b63d:  1095 ,  1099 ,  1101#
pradrl = $b5fc:  1056 ,  1067#
pradrm = $b602:  1062 ,  1069#
pradro = $b5ef:  1055#,  1071
prmn1  = $b6ac:  1195#,  1205
prmn2  = $b6b0:  1197#,  1201
prmne  = $b6a1:  1074 ,  1186#
ptr2   =   $9f:    31#,   872$,   910 ,   989 ,   998$
puthex = $b8c2:   176 ,   456 ,  1090 ,  1624 ,  1630 ,  1666#
puthxs = $b8a5:   183 ,  1068 ,  1630#
putspc = $b8a8:   450 ,   561 ,   562 ,   637 ,   638 ,  1046 ,  1048 ,  1206 ,
                  1635#
putt2  = $b892:   447 ,   560 ,   636 ,  1047 ,  1612#
putwrd = $b89f:  1113 ,  1622#,  1868
range  = $b983:   502 ,   590 ,   754 ,  1826#
ranger = $b9af:  1826 ,  1829 ,  1843 ,  1847#
regk   = $b7a1:  1428#
relad1 = $b641:  1089 ,  1108#
relad2 = $b64a:  1111 ,  1113#
rtmode = $b673:  1146 ,  1155#
sa     =   $b9:    33#,   658$,   714$,   748$
setme1 = $b1b2:   407#,   415 ,   417
setme2 = $b1c5:   413 ,   416#
setme3 = $b1c9:   400 ,   408 ,   418#
setmem = $b1ab:   287 ,   400#
setreg = $b194:   288 ,   382#
setrg1 = $b199:   388#,   394
setrg2 = $b1a8:   389 ,   395#
shift  = $0ab6:    65#,   841 ,  1500$,  1531 ,  1587$,  1961$,  1962
shiftb = $b88e:  1499 ,  1606#
shifts = $b18e:   362 ,   375#
skip1  =   $24:    11#,  1460 ,  1581 ,  1845
skip2  =   $2c:    12#,  2001
sreg   =   $05:    19#,   132$,   391$
stack  = $0100:    43#,   610$
start  = $b046:   122 ,   148#
stash  = $b12a:   305#,   410 ,   767 ,   940 ,   944
stashx = $02b9:    46#,   307$,   545$
status =   $90:    28#,   663$,   729 ,  2053 ,  2099 ,  2103 ,  2120
stkey  =   $91:    29#
stkptr =   $09:    23#,   150$,   430
subom2 = $b90e:   343 ,  1017 ,  1731#,  1836
subto  = $b924:  1035 ,  1751#
sxreg  = $0aaf:    59#,   988$,   999 ,  1666$,  1670 ,  1704$,  1713 ,  1751$,  1754
sysvec = $0a00:    52#,   252
```

```
t1      =   $63:   25#,  519 ,  522 ,  525 ,  868$,  876$,  909 ,  916 ,
                 1188$, 1190$, 1197$, 1198$, 1768 , 1770$, 1771 , 1773$,
                 1774 , 1776$, 1838$, 1840$, 1842$, 2096$, 2105$
t2      =   $66:   26#,  294 ,  295 ,  306 ,  308 ,  319 ,  321 ,  514 ,
                  516 ,  530$,  715 ,  745 ,  746 ,  756 ,  920 ,  923 ,
                  963 ,  966 ,  970 , 1118 , 1122 , 1612 , 1616 , 1617 ,
                 1721$, 1723$, 1725$, 1733 , 1736 , 1739 , 1788 , 1789$,
                 1791$, 1793$, 1800 , 1802$, 1803 , 1805$, 1806 , 1808$,
                 1920$, 1921$, 1922$
temps   = $0ab7:   66#,  529 ,  757 , 1524$, 1544$, 1545$, 1546$, 1548 ,
                 1551 , 1554 , 1831$, 1833$, 1835$
to      =   $60:   24#,  338$,  352$,  353$,  354$,  390 ,  409 ,  515 ,
                  517 ,  520 ,  521$,  523 ,  524$,  526 ,  527$,  544 ,
                  547 ,  556 ,  565$,  567$,  612 ,  699 ,  708 ,  709 ,
                  766 ,  838 ,  846 ,  919 ,  922 ,  939 , 1014$, 1470$,
                 1471$, 1472$, 1523 , 1536$, 1537$, 1538$, 1549 , 1550$,
                 1552 , 1553$, 1555 , 1556$, 1564 , 1565$, 1567 , 1568$,
                 1570 , 1571$, 1720 , 1722 , 1724 , 1732 , 1734$, 1735 ,
                 1737$, 1738 , 1740$, 1753 , 1755$, 1756 , 1758$, 1759 ,
                 1761$, 1815 , 1816 , 1817 , 1830 , 1832 , 1834 , 1837 ,
                 1839 , 1841 , 1857 , 1866 , 1867 , 1947 , 1949 , 1951 ,
                 2006 , 2011$, 2013$, 2022 , 2083 , 2098$, 2102$
totopc  = $b974:   382 ,  429 ,  437 , 1814#
totopx  = $b982:  1814 , 1821#
totot2  = $b901:   334 ,  401 ,  703 , 1006 , 1720#, 1827 , 1905
trns01  = $b247:   503 ,  506#
trns02  = $b24a:   505 ,  511#
trns03  = $b26d:   529#,  532
trns04  = $b27a:   512 ,  518 ,  535#
trns05  = $b27f:   541#,  584
trns06  = $b28f:   547#
trns07  = $b299:   550 ,  556#
trns08  = $b2aa:   559 ,  563#
trns09  = $b2ba:   564 ,  574#
trns10  = $b2c3:   566 ,  568 ,  578#
trns11  = $b2c6:   576 ,  583#
trns12  = $b2cb:   542 ,  585#
trnsfr  = $b234:   283 ,  498#
tst10   = $b593:   991 ,  998#
tst2    = $b57c:   891 ,  907 ,  908 ,  983#
tsto5   = $b58b:   912 ,  994#
tstrx   = $b57f:   879 ,  881 ,  897 ,  900 ,  983 ,  988#
txtptr  =   $7a:    27#,  193$,  678 ,  694$,  854$, 1437$, 1446$, 1493$,
                 1699$
                 1705 , 1712$, 2018$, 2034 , 2035$, 2072 , 2079
unpack  = $ba5d:  1879 , 1960#, 2111
unpck1  = $ba63:  1962#, 1993
unpck2  = $ba68:  1968#, 1974
unpck3  = $ba84:  1976 , 1978 , 1985#
unpck4  = $ba8c:  1980 , 1992#
unpkto  = $ba47:  1889 , 1899 , 1946#
verck   =   $93:    30#,  247$,  499$,  511 ,  549 ,  619$,  634 ,  710 ,  722
                  732
wrap    = $0ab1:    60#,  870$,  873 ,  943 ,  994$
xcnt    = $0a80:    53#,  603$,  613$,  631 ,  664 ,  666
xreg    =   $07:    21#,  130$
xsave   = $0ab2:    61#,  293$,  299 ,  305$,  312 ,  318$,  326
yreg    =   $08:    22#,  131$
```

4 Editor und Kernal des C-128

4.1 Kommentiertes Assemblerlisting

4.1.1 Editor-/Operating-System-Labels

```
 Pass 1


00005  0000                    .lib c128.lab
00507  0000                    .lib c128.cpu
00563  0000                    .lib c128.vic
00693  0000                    .lib c128.sid
00795  0000                    .lib c128.mmu
00958  0000                    .lib c128.vdc
01125  0000                    .lib c128.cia1
01318  0000                    .lib c128.cia2
01500  0000                    .lib c128.screen1
01956  c234                    .lib c128.screen2
02644  c55d                    .lib c128.keyboard1
03453  c854                    .lib c128.keyboard2
04440  cc27                    .lib c128.editor4
05102  d000                    .lib c128.init
06106  e33b                    .lib c128.floppy
06745  e5ff                    .lib c128.rs232
07636  e8d0                    .lib c128.tape1
08326  ec1f                    .lib c128.tape2
08868  eeeb                    .lib c128.kernal1
09571  f188                    .lib c128.kernal2
10552  f5b5                    .lib c128.kernal3
11499  f9fb                    .lib c128.kernal4
11705  fc3b                    .lib c128.keyboard3
12017  fe71                    .lib c128.kernal5

 Pass 2


00001  0000
00002  0000
00003  0000
00004  0000
00005  0000                    .lib c128.lab
00006  0000
00008  0000
00009  0000
00010  0000  ===> Deklaration von Codes, Token <====
00011  0000
00012  0000            eom     = 0             END-OF-MESSAGE
00013  0000            cr      = $0d           Carriage Return
00014  0000            esc     = $1b           Escape
```

```
00015  0000            sftcr  = $8d          Shift Carriage Return
00016  0000            sftspc = $a0          Shift Space
00017  0000            pi     = $de          ASC für PI
00018  0000            pitk   = $ff          BASIC-Token für PI
00019  0000
00020  0000            skip1  = $24          Opcode zum Überspringen von 1-Byte
                                             Befehlen
00021  0000            skip2  = $2c          Opcode zum Überspringen von 2-Byte
                                             Befehlen
00022  0000
00023  0000            aigu   = $af          DIN-Tastaturcode Accent aigu
00024  0000            grave  = $c0          DIN-Tastaturcode Accent grave
00025  0000            crcflx = $bf          DIN-Tastaturcode Accent circonflexe
00026  0000
00027  0000
00028  0000  ===> Definitionen im COMMON RAM <===
00029  0000
00030  0000            bank   = $02          Banknummer
00031  0000            pchi   = $03          8502 CPU PC hi
00032  0000            pclo   = $04          8502 CPU PC lo
00033  0000            sreg   = $05          8502 CPU sr
00034  0000            areg   = $06          8502 CPU ac
00035  0000            xreg   = $07          8502 CPU xr
00036  0000            yreg   = $08          8502 CPU yr
00037  0000            stkptr = $09          8502 CPU sp
00038  0000
00039  0000            status = $90          I/O-Status 'st'
00040  0000
00041  0000  7.......  1 = Dev.not pres. oder Band-Ende
00042  0000  .6......  1 = End of File
00043  0000  ..5.....  1 = Prüfsummenfehler
00044  0000  ...4....  1 = Fatal Error
00045  0000  ....3...  1 = Block zu lang (Kass.)
00046  0000  .....2..  1 = Block zu kurz (Kass.)
00047  0000  ......1.  1 = Timeout lesen
00048  0000  .......0  1 = Timeout schreiben
00049  0000
00050  0000  00000000  Status ok
00051  0000
00052  0000            stkey  = $91          Flag für Stopp- und RVS-Taste
00053  0000
00054  0000  01111111 ;Stopp-Taste gedrückt
00055  0000  11111111 ;CBM-Taste gedrückt
00056  0000
00057  0000            svxt   = $92          Kassette Timing-Konstante
00058  0000
00059  0000            verck  = $93          Flag für load/verify
00060  0000
00061  0000  00000000  load
00062  0000  00000001  verify
00063  0000
00064  0000            c3po   = $94          IEC-Puffer-Flag
00065  0000            bsour  = $95          Zeichen für IEC-Ausgabe
00066  0000            syno   = $96          Kassette Synchronisations-Wert
00067  0000            xsav   = $97
00068  0000            ldtnd  = $98          Anzahl offene Files
00069  0000            dfltn  = $99          Vorgabe Eingabe-Gerätenummer (0)
00070  0000            dflto  = $9a          Vorgabe Ausgabe-Gerätenummer (3)
```

```
00071  0000            prty   = $9b          Paritäts-Bit Kassettenoperation
00072  0000            dpsw   = $9c          Flag für Byte-Ausgabe Kassette
00073  0000            msgflg = $9d          Flag disable Ausgabe OPSYS-Meldungen
00074  0000            ptr1   = $9e
00075  0000            ptr2   = $9f
00076  0000            time   = $a0          Uhrzeit in 1/60 s (jiffies)
00077  0000            pcntr  = $a3
00078  0000            bsour1 = $a4
00079  0000            cntdn  = $a5
00080  0000            bufpt  = $a6          Zeiger in Kassettenpuffer
00081  0000            inbit  = $a7
00082  0000            bitci  = $a8          RS 232 Bitzähler
00083  0000            rinone = $a9          RS 232 Flag für Startbit
00084  0000            ridata = $aa          RS 232 Eingabe Byte
00085  0000            riprty = $ab          RS 232 Eingabe-Parität
00086  0000            sal    = $ac          addrl für load/save
00087  0000            sah    = $ad          addrh für load/save
00088  0000            eal    = $ae          addrl Ende load/save
00089  0000            eah    = $af          addrh Ende load/save
00090  0000            cmpo   = $b0          Timing-Wert für Kassette
00091  0000            temp   = $b1
00092  0000            tape1  = $b2          addrl Zeiger Kassettenpuffer
00093  0000
00094  0000            snsw1  = $b4          RS 232 Ausgabe Bitzähler
00095  0000
00096  0000  11111111  1 Stopp-Bit
00097  0000  11111110  2 Stopp-Bits
00098  0000  00000101  5 Datenbits
00099  0000  00000110  6 Datenbits
00100  0000  00000111  7 Datenbits
00101  0000  00001000  8 Datanbits
00102  0000
00103  0000            nxtbit = $b5          RS 232 nächstes Ausgabe-Bit
00104  0000            rodata = $b6          RS 232 Ausgabe Byte-Puffer
00105  0000            fnlen  = $b7          Länge akt. Filename
00106  0000            la     = $b8          akt. log. Filenummer
00107  0000            sa     = $b9          akt. Sekundäradresse
00108  0000            fa     = $ba          akt. Gerätenummer
00109  0000            fnadr  = $bb          addrl akt. Filename
00110  0000            ochar  = $bd          RS 232 Ausgabe-Parität
00111  0000            fsblk  = $be          Blockzähler Kassette
00112  0000            mych   = $bf          Puffer für serielles Zeichen
00113  0000            cas1   = $c0          Flag für Schalter Kass.taste
00114  0000            stal   = $c1          addrl Startadresse save
00115  0000            stah   = $c2          addrh Startadresse save
00116  0000            tmp2   = $c3
00117  0000            data   = $c5          Kassetten Lese- und Schreibdaten
00118  0000            ba     = $c6          akt. Bank für Speicher-Operationen
00119  0000            fnbank = $c7          Bank des akt. Filenamens
00120  0000            ribuf  = $c8          RS 232 addrl Eingabe-Puffer
00121  0000            robuf  = $ca          RS 232 addrl Ausgabepuffer
00122  0000            keytab = $cc          addrl Zeiger in Tast.-Dekodier-Tab.
00123  0000            imparm = $ce          Hilfsindex Ausgabe Text aus PGM-Code
00124  0000            ndx    = $d0          Bytes im Tast.puffer
00125  0000            kyndx  = $d1          Bytes im Fkt.tasten-Puffer
00126  0000            keyidx = $d2          Zeiger in Text akt. Fkt.taste
00127  0000
00128  0000            shflag = $d3          Flag für Shift/CTRL/CBM-Taste
```

```
00129  0000
00130  0000   7.......  Flag für Spaltenabfrage Tastatur mit CIA 1/VIC
00131  0000   1xxxxxxx  Tastaturabfrage VIC
00132  0000   0xxxxxxx  Tastaturabfrage CIA 1
00133  0000
00134  0000   .65.....  nicht benutzt
00135  0000
00136  0000   ...4....  Flag für CAPS LOCK bzw. ASCII/DIN
00137  0000   xxx1xxxx  Taste gedrückt
00138  0000   xxx0xxxx  Taste nicht gedrückt
00139  0000
00140  0000   ....3210  Flags für reserv. Tasten
00141  0000   xxxx1xxx  ALT
00142  0000   xxxxx1xx  CTRL
00143  0000   xxxxxx1x  CBM-Taste
00144  0000   xxxxxxx1  Shift li./re.
00145  0000
00146  0000
00147  0000              sfdx   = $d4          Nummer der akt. Taste
00148  0000              lstx   = $d5          letzte gedrückte Taste
00149  0000              crsw   = $d6          Flag Warten Tast.eingabe
00150  0000              mode   = $d7          BS-Modus 40/80-Zeichen
00151  0000
00152  0000   00000000  40-Zeichen BS
00153  0000   10000000  80-Zeichen BS
00154  0000
00155  0000              graphm = $d8          Flag Text/Graphik/Split-BS
00156  0000
00157  0000   00000000  Text: graph 0
00158  0000   00100000  HI-RES: graph 1
00159  0000   01100000  geteilter BS mit HI-RES/Text: graph 2
00160  0000   10100000  HI-RES Multi-Color-Modus: graph 3
00161  0000   11100000  geteilter BS mit HI-RES-Multi-Color/Text: graph 4
00162  0000
00163  0000              charen = $d9          RAM/ROM-Flag für VIC
00164  0000              sedsal = $da          addrl Zeiger auf Startzeile
00165  0000              sedsah = $db          addrh Zeiger auf Startzeile
00166  0000              sedeal = $dc          addrl Zeiger auf Endezeile
00167  0000              sedeah = $dd          addrh Zeiger auf Endezeile
00168  0000              keybnk = $de
00169  0000              keytmp = $df
00170  0000              pnt    = $e0          addrl Zeiger auf akt. BS-Zeile
00171  0000              user   = $e2          addrl Zeiger auf akt. Attributzeile
00172  0000              scbot  = $e4          BS-Fenster: unterste Zeile
00173  0000              sctop  = $e5          BS-Fenster: oberste Zeile
00174  0000              sclf   = $e6          BS-Fenster: Spalte linker Rand
00175  0000              scrt   = $e7          BS-Fenster: Spalte rechter Rand
00176  0000              lsxp   = $e8          Cursor-Zeile vor input
00177  0000              lstp   = $e9          Cursor-Spalte vor input
00178  0000              indx   = $ea          Spalte letztes Zeichen in Zeile
00179  0000              tblx   = $eb          akt. Cursor-Zeile
00180  0000              pntr   = $ec          akt. Cursor-Spalte
00181  0000              lines  = $ed          höchste BS-Zeile akt. Modus
00182  0000              colums = $ee          höchste BS-Spalte akt. Modus
00183  0000              datax  = $ef          akt. Ausgabezeichen
00184  0000              lstchr = $f0          letztes Ausgabezeichen
00185  0000              color  = $f1          akt. Attribut
00186  0000
```

```
00187  0000  im 80-Zeichen-Modus (VDC)
00188  0000
00189  0000  7.......  1 = Groß/Graphik, 0 = Klein/Groß
00190  0000  ..5.....  1 = unterstreichen ein, 0 = aus
00191  0000  ...4....  1 = Blinken ein, 0 = aus
00192  0000  ....3...  1 = Rot
00193  0000  .....2..  1 = Grün
00194  0000  ......1.  1 = Blau
00195  0000  .......0  1 = Intensität
00196  0000
00197  0000            tcolor = $f2          temporäres Register für Attribut
00198  0000            rvs    = $f3          Flag für REVERS-Modus
00199  0000
00200  0000  00000000  REVERS-Modus aus
00201  0000  10000000  REVERS-Modus ein
00202  0000
00203  0000            qtsw   = $f4          Flag für QUOTE-Modus
00204  0000
00205  0000  00000000  QUOTE-Modus aus
00206  0000  00000001  QUOTE-Modus ein
00207  0000
00208  0000            insrt  = $f5          Zähler für einzufügende Zeichen
00209  0000            insflg = $f6          AUTO-INSERT-Flag
00210  0000
00211  0000  11111111  AUTO-INSERT-Modus ein
00212  0000  00000000  AUTO-INSERT-Modus aus
00213  0000
00214  0000            locks  = $f7          Flag enable/disable Shift/CBM-Taste
00215  0000
00216  0000  7.......  1 = shift-Commodore-Taste zugelassen, 0 = gesperrt
00217  0000  .6......  1 = CTRL-s/NO SCROLL gesperrt, 0 = zugelassen
00218  0000
00219  0000            scroll = $f8          scroll-disable-Flag
00220  0000
00221  0000  7.......  1 = BS Scroll aus
00222  0000  .6......  1 = BS-Doppelzeilen ausgeschaltet
00223  0000
00224  0000            beeper = $f9          Flag enable/disable beep
00225  0000
00226  0000  10000000  Beeper aus
00227  0000  00000000  Beeper ein
00228  0000
00229  0000            dummy  = $ff          Dummy-Adr. für ind. RAM-Routinen
00230  0000
00231  0000            stack  = $0100        8502 CPU-Stapel
00232  0000
00233  0000            buf    = $0200        Kassettenpuffer
00234  0000
00235  0000            fetch  = $02a2        LDA (LDAVEC),y, Bank xr
00236  0000            fetchx = $02aa        LDAVEC
00237  0000            stash  = $02af        STA (STAVEC),y, Bank xr
00238  0000            stashx = $02b9        STAVEC
00239  0000            cmpare = $02be        CMP (CMPVEC),y, Bank xr
00240  0000            jsrfar = $02cd        JSR, Bank xr
00241  0000            jmpfar = $02e3        JMP, Bank xr
00242  0000
00243  0000            iirq   = $0314        Hardware-IRQ ($fa65)
00244  0000            ibrk   = $0316        BREAK Instruction ($b003)
```

```
00245  0000            inmi   = $0318        Hardware NMI ($fa40)
00246  0000            iopen  = $031a        log. File öffnen ($efbd)
00247  0000            iclose = $031c        log. File schliessen ($f188)
00248  0000            ichkin = $031e        Eingabe-Kanal öffnen ($f106)
00249  0000            ickout = $0320        Ausgabe-Kanal öffnen ($f14c)
00250  0000            iclrch = $0322        Ein-/Ausg.-Kanal schließen ($f226)
00251  0000            ibasin = $0324        Zeichen akt. Kanal nach ac ($ef06)
00252  0000            ibsout = $0326        Ausgabe ac auf akt. Kanal ($ef79)
00253  0000            istop  = $0328        Stopp-Taste lesen ($f66e)
00254  0000            igetin = $032a        Zeichen akt. Kanal nach ac ($eeeb)
00255  0000            iclall = $032c        off. Files =0, Vorgabe-I/O ($f222)
00256  0000            iload  = $0330        Einlesen vom log. File ($f26c)
00257  0000            isave  = $0332        Abspeichern auf log. File ($f54e)
00258  0000            ctlvec = $0334        Bearbeitung CTRL-Tasten ($c7b9)
00259  0000            shfvec = $0336        Bearb. geshiftete Tasten ($c805)
00260  0000            escvec = $0338        Bearbeitung ESC-Fnkt. ($c9c1)
00261  0000            keyvec = $033a        Bearbeitung Fkt.Tasten ($c6ad)
00262  0000            keychk = $033c        Zeichen-Bearbeitung
00263  0000            nrmtab = $033e        addrl akt. Tast.tab. normal
00264  0000            keyd   = $034a        Tast.puffer
00265  0000            tabmap = $0354        Tab. der akt. Tabulator-Pos.
00266  0000            aktcol = $035d        akt. Farbe für Zeichenausgabe
00267  0000            bitabl = $035e        Tab. BS-Doppelzeilen
00268  0000            lat    = $0362        Tab. der log. Filenummern
00269  0000            fat    = $036c        Tab. der Gerätenummern
00270  0000            sat    = $0376        Tab. der Sekundär-Adr.
00271  0000            dodma  = $03f0        DMA-Register setzen
00272  0000
00273  0000            linz1  = $0400        Bildschirmzeile 1
00274  0000            linz2  = $0428        Bildschirmzeile 2
00275  0000            linz3  = $0450        Bildschirmzeile 3
00276  0000            linz4  = $0478        Bildschirmzeile 4
00277  0000            linz5  = $04a0        Bildschirmzeile 5
00278  0000            linz6  = $04c8        Bildschirmzeile 6
00279  0000            linz7  = $04f0        Bildschirmzeile 7
00280  0000            linz8  = $0518        Bildschirmzeile 8
00281  0000            linz9  = $0540        Bildschirmzeile 9
00282  0000            linz10 = $0568        Bildschirmzeile 10
00283  0000            linz11 = $0590        Bildschirmzeile 11
00284  0000            linz12 = $05b8        Bildschirmzeile 12
00285  0000            linz13 = $05e0        Bildschirmzeile 13
00286  0000            linz14 = $0608        Bildschirmzeile 14
00287  0000            linz15 = $0630        Bildschirmzeile 15
00288  0000            linz16 = $0658        Bildschirmzeile 16
00289  0000            linz17 = $0680        Bildschirmzeile 17
00290  0000            linz18 = $06a8        Bildschirmzeile 18
00291  0000            linz19 = $06d0        Bildschirmzeile 19
00292  0000            linz20 = $06f8        Bildschirmzeile 20
00293  0000            linz21 = $0720        Bildschirmzeile 21
00294  0000            linz22 = $0748        Bildschirmzeile 22
00295  0000            linz23 = $0770        Bildschirmzeile 23
00296  0000            linz24 = $0798        Bildschirmzeile 24
00297  0000            linz25 = $07c0        Bildschirmzeile 25
00298  0000
00299  0000            sysvec = $0a00        Vector (BASIC-) Warmstart ($4003)
00300  0000            dejavu = $0a02        Flag (KERNAL-) Warm/Kalt-Start
00301  0000
00302  0000            palnts = $0a03        Flag PAL/NTSC
```

```
00303  0000
00304  0000  11111111  PAL
00305  0000  00000000  NTSC
00306  0000
00307  0000            initst = $0a04          Flag Status nach RESET, BASIC-IRQ
00308  0000
00309  0000  7.......  1 = VDC initialisiert, Zeichengenerator in VDC geladen
00310  0000  .6......  1 = Adreßtabelle Tastaturdekodierung/Funktionstasten init.
00311  0000  .......0  1 = Interrupt-Anforderung (IRQ) von BASIC-Programm
00312  0000
00313  0000            memstr = $0a05          addrl Anfang des Benutzerspeichers
00314  0000            memsiz = $0a07          addrl Ende des Benutzerspeichers
00315  0000            irqtmp = $0a09          Puffer für addrl IRQ Kassette
00316  0000            caston = $0a0b          Zeitvergleich für Kass.-Operat.
00317  0000            kika26 = $0a0c          temporäres Register Kassette
00318  0000            stupid = $0a0d          temporäres Register Kassette
00319  0000            timout = $0a0e          IEC-Timeout enable Flag
00320  0000
00321  0000  IEC-Timeout wird vom System nicht bearbeitet
00322  0000
00323  0000            enabl  = $0a0f          CIA2 NMI-Flag
00324  0000
00325  0000  ...4....  1 = FLAG IRQ (RS Empfang)
00326  0000  ......1.  1 = TIMER B UNDERFLOW (RS Empfang)
00327  0000  .......0  1 = TIMER A UNDERFLOW (RS Senden)
00328  0000
00329  0000            m51ctr = $0a10          RS 232 CTRL-Register
00330  0000
00331  0000  7.......  Anzahl Stopp-Bits:
00332  0000  0xxxxxxx  1 Stopp-Bit
00333  0000  1xxxxxxx  2 Stopp-Bits
00334  0000
00335  0000  .65.....  Anzahl Datenbits
00336  0000  x00xxxxx  8 Datenbits
00337  0000  x01xxxxx  7 Datenbits
00338  0000  x10xxxxx  6 Datenbits
00339  0000  x11xxxxx  5 Datenbits
00340  0000
00341  0000  ...4....  unbenutzt
00342  0000
00343  0000  ....3210  Baud-Rate-Schlüssel
00344  0000  xxxx0000  USER-Baud-Rate
00345  0000  xxxx0001     50   Baud
00346  0000  xxxx0010     75   Baud
00347  0000  xxxx0011    110   Baud
00348  0000  xxxx0100    134.5 Baud
00349  0000  xxxx0101    150   Baud
00350  0000  xxxx0110    300   Baud
00351  0000  xxxx0111    600   Baud
00352  0000  xxxx1000   1200   Baud
00353  0000  xxxx1001   2400   Baud
00354  0000
00355  0000            m51cdr = $0a11          RS 232 Command-Register
00356  0000
00357  0000  765.....  PARITY
00358  0000  xx0xxxxx  keine PARITY-Prüfung, weder Senden noch Empfangen
00359  0000  001xxxxx  ungerade PARITY Senden/Empfangen
00360  0000  011xxxxx  gerade PARITY Senden/Empfangen
```

```
00361  0000   101xxxxx  keine PARITY-Prüfung, PARITY MARK Senden
00362  0000   111xxxxx  keine PARITY-Prüfung, PARITY SPACE Senden
00363  0000
00364  0000   ...4....  DUPLEX
00365  0000   xxx0xxxx  FULL DUPLEX
00366  0000   xxx1xxxx  HALF DUPLEX
00367  0000
00368  0000   ....321.  unbenutzt
00369  0000
00370  0000   .......0  HANDSHAKE
00371  0000   xxxxxxx0  3-Draht
00372  0000   xxxxxxx1  1-X Draht
00373  0000
00374  0000              m51ajb = $0a12          RS 232 BAUD-Rate Konst. lo
00375  0000              rsstat = $0a14          RS 232 Status-Byte
00376  0000
00377  0000   7.......  1= BREAK Detected Bit
00378  0000   .6......  1= DSR fehlt
00379  0000   ..5.....     nicht benutzt
00380  0000   ...4....  1= CTS fehlt
00381  0000   ....3...  1= Empfänger-Puffer leer
00382  0000   .....2..  1= Empfänger-Puffer übergelaufen
00383  0000   ......1.  1= FRAMING Fehler
00384  0000   .......0  1= PARITY Fehler
00385  0000
00386  0000              bitnum = $0a15          RS 232 Anzahl Datenbits
00387  0000              baudof = $0a16          RS 232 Timer-Konst. lo BAUD-Rate
00388  0000              ridbe  = $0a18          RS 232 Zeiger auf eingeles. Byte
00389  0000              ridbs  = $0a19          RS 232 Zeiger auf auszules. Byte
00390  0000              rodbs  = $0a1a          RS 232 Zeiger auf einzules. Byte
00391  0000              rodbe  = $0a1b          RS 232 Zeiger auf auszugeb. Byte
00392  0000              serial = $0a1c          Flag für Fast-Serial-Device
00393  0000              timer  = $0a1d          Speicher für Echtzeituhr
00394  0000              xmax   = $0a20          max. Länge Tast.puffer
00395  0000
00396  0000              pause  = $0a21          Flag für CTRL s/ NO SCROLL
00397  0000
00398  0000   00000000  BS-Ausgabe möglich
00399  0000   <> 0      BS-Ausgabe halt (OPSYS benutzt für <>0 den Wert $0d)
00400  0000
00401  0000              rptflg = $0a22          Flag für REPEAT, alle Tasten = $80
00402  0000
00403  0000   7.......  1 = REPEAT alle Tasten, 0 = Cursortasten
00404  0000   .6......  1 = REPEAT aus, 0 = ein
00405  0000
00406  0000              kount  = $0a23          Zähler für REPEAT-Geschwindigkeit
00407  0000              delay  = $0a24          Zeit bis zum Einsetzen des REPEAT
00408  0000              lstshf = $0a25
00409  0000
00410  0000              blnon  = $0a26          Cursor-Modus 40-Zeichen
00411  0000
00412  0000   7.......  1 = Cursor REVERS, 0 = Cursor aus (Blinkstatus)
00413  0000   .6......  1 = fest, 0 = blinken
00414  0000
00415  0000              blnsw  = $0a27          VIC Flag für Cursor-Blinken
00416  0000
00417  0000   00000000  Cursor 40-Zeichen BS ein
00418  0000   <>0       Cursor aus
```

```
00419  0000
00420  0000             curcnt = $0a28          VIC Zähler für Cursor-Blinken
00421  0000             gdbln  = $0a29          VIC Cursorzeichen vor Blinken
00422  0000             gdcol  = $0a2a          VIC Cursorfarbe vor Blinken
00423  0000
00424  0000             curmod = $0a2b          VDC Cursormodus
00425  0000
00426  0000  .65.....   Cursor-Modus:
00427  0000  x00xxxxx   Cursor blinkt nicht
00428  0000  x01xxxxx   Cursor ausgeschaltet
00429  0000  x10xxxxx   Cursor blinkt schnell (1/16 Bildwiederholfrequenz)
00430  0000  x11xxxxx   Cursor blinkt langsam (1/32 Bildwiederholfrequenz)
00431  0000  ...43210   Cursor-Anfangszeile im Zeichen (hier 0-7)
00432  0000
00433  0000             vm1    = $0a2c          VIC Text-Basiszeiger
00434  0000             vm2    = $0a2d          VIC Bit-Map Basiszeiger
00435  0000             vm3    = $0a2e          VDC Basisadresse BS-RAM
00436  0000             vm4    = $0a2f          VDC Basisadresse Attribut-RAM
00437  0000             lintmp = $0a30          Puffer für BS-Zeilennummer
00438  0000             sav80a = $0a31          VDC temporäres Register
00439  0000             sav80b = $0a32          VDC temporäres Register
00440  0000             curcol = $0a33          VDC Cursorfarbe vor dem Blinken
00441  0000             split  = $0a34          VIC Rasterwert geteilter BS
00442  0000             fnadrx = $0a35          Speicher während Bank-Operation
00443  0000             palcnt = $0a36          Korrekturwert für Systemuhr 50 Hz
00444  0000             speed  = $0a37          Speicher für akt. CPU-Takt
00445  0000             sprits = $0a38          Speicher Einschaltzustand Sprites
00446  0000
00447  0000             blnkng = $0a39          Flag-Bit-Speicher VIC Reg. 17
00448  0000
00449  0000  00000000  VIC Blank Screen (Ganzer BS Randfarbe)
00450  0000  00010000  VIC normaler Schirm
00451  0000
00452  0000             holdof = $0a3a          Flag 'Sprites abschalten'
00453  0000
00454  0000  7.......  1 = Sprites während Floppy/Kasseten I/O
00455  0000                nicht abschalten, 0 = abschalten
00456  0000
00457  0000             ldtbsa = $0a3b          VIC Basisadresse BS-RAM
00458  0000             clreal = $0a3c          clear end address lo-byte
00459  0000             clreah = $0a3d          clear end address hi-byte
00460  0000             swpout = $0a40          Swap-Bereich Editor-Werte
00461  0000             swpmap = $0a60          Swap-Bereich Editor-Werte
00462  0000             curbnk = $0ac0          akt. Bank Fkt.tasten
00463  0000             pat    = $0ac1          Tab. der Flags init. ROM-Module
00464  0000
00465  0000             dinflg = $0ac5          Flag Akzente & ASCII/DIN-Modus
00466  0000
00467  0000  7.......  Umschaltung durch CAPS-LOCK-Taste:
00468  0000  0xxxxxxx  ASCII / DIN-Tastatur
00469  0000  1xxxxxxx  normal ASCII / CAPS-LOCK ASCII
00470  0000             oder DIN / DIN ohne Akzente
00471  0000
00472  0000  .....210  Flag letzte gedrückte Taste:
00473  0000  xxxxx000  kein Akzent
00474  0000  xxxxx001  Accent aigu
00475  0000  xxxxx010  Accent grave
00476  0000  xxxxx100  Accent circonflexe
```

```
00477  0000   xxxxx101  Reserve (0)
00478  0000   xxxxx110  Reserve (0)
00479  0000
00480  0000             tbuffr = $0b00          Kassettenpuffer
00481  0000             rsibuf = $0c00          RS 232 Eingabe-Puffer
00482  0000             rsobuf = $0d00          RS 232 Ausgabe-Puffer
00483  0000
00484  0000             pkybuf = $1000          Längentab. Fkt.tasten-Texte
00485  0000             pkydef = $100a          Puffer für Fkt.tasten-Befehle
00486  0000
00487  0000             rambot = $1c00          Anfang Speicher BASIC-Programm
00488  0000
00489  0000             coldvc = $4000          Vector (BASIC-) Kaltstart
00490  0000             birqvc = $4006          BASIC-Interrupt bearbeiten
00491  0000
00492  0000
00493  0000             moncal = $b000          Monitor-Einsprung über CALL
00494  0000             monbrk = $b003          Monitor-Einsprung über BREAK
00495  0000             moncmd = $b006          Monitor-Kommando ausführen
00496  0000
00497  0000             chrrom = $d000          Basisadresse Zeichengenerator
00498  0000             chips  = $d000          Basisadresse der Chips
00499  0000
00500  0000             colram = $d800          VIC Farb-RAM
00501  0000
00502  0000             dmacmd = $df01
00503  0000
00504  0000             locker = $fff5          CBM-ID
00505  0000
00506  0000                    .end
00507  0000                    .lib c128.cpu
00508  0000
```

4.1.2 CPU 8502 I/O-Port

```
00510  0000
00511  0000
00512  0000  ===> CPU 8502 I/O-Port <===
00513  0000
00514  0000  ---- CPU 8502 Port Datenrichtungs Register ----
00515  0000
00516  0000             d8502  = $00           CPU 8502 Port Datenrichtungs
                                                Register
00517  0000
00518  0000   x0101111  0 = Input, 1 = Output (s. Daten Reg.)
00519  0000
00520  0000  ---- CPU 8502 Port Daten Register ----
00521  0000
00522  0000             r8502  = $01           CPU 8502 Port Daten Register
00523  0000
00524  0000   7.......  unbenutzt
00525  0000
00526  0000   .6......  Status CAPS LOCK/DIN Taste
00527  0000
00528  0000   x0xxxxxx  Taste gedrückt
00529  0000   x1xxxxxx  Taste nicht gedrückt
```

```
00530  0000
00531  0000   ..5.....  Steuerung Kassettenmotor
00532  0000
00533  0000   xx0xxxxx  Motor ein
00534  0000   xx1xxxxx  Motor aus
00535  0000
00536  0000   ...4....  Eingang Taste Kassettenrek.
00537  0000
00538  0000   xxx0xxxx  Taste an Rekorder gedrückt
00539  0000   xxx1xxxx  Taste an Rekorder nicht gedrückt
00540  0000
00541  0000   ....3...  Kassette Schreibleitung
00542  0000
00543  0000   xxxx0xxx  serielle Schreibleitung 0
00544  0000   xxxx1xxx  serielle Schreibleitung 1
00545  0000
00546  0000   .....2..  Auswahl I/O / ROM $d000-$dfff (C 64)
00547  0000
00548  0000   xxxxx0xx  Character Generator ROM ein $d000-$dfff
00549  0000   xxxxx1xx  I/O-Chips C 64 ein $d000-$dfff
00550  0000
00551  0000   ......1.  Auswahl KERNAL ROM / RAM $e000-$ffff (C 64)
00552  0000
00553  0000   xxxxxx0x  8 KByte RAM ein $e000-$ffff
00554  0000   xxxxxx1x  KERNAL ROM C 64 ein $d000-$dfff
00555  0000
00556  0000   .......0  Auswahl BASIC ROM / RAM $a000-$bfff (C 64)
00557  0000
00558  0000   xxxxxxx0  8 KByte RAM ein $a000-$bfff
00559  0000   xxxxxxx1  BASIC ROM C 64 ein $a000-$bfff
00560  0000
00561  0000
00562  0000                    .end
00563  0000                    .lib c128.vic
00564  0000
```

4.1.3 VIC 8564 Video Controller

```
00566  0000
00567  0000
00568  0000  ===> VIC 8564 Video Controller <===
00569  0000
00570  0000   unbenutzte Bits erscheinen beim Lesen immer als 1.
00571  0000
00572  0000  Reg.    Bits
00573  0000  ==============
00574  0000
00575  0000             vic    = $d000          Basisadresse VIC 8564
00576  0000
00577  0000    0   76543210  Sprite0, lo-Byte x-Wert (Abszisse)
00578  0000    1   76543210  Sprite0, lo-Byte y-Wert (Ordinate)
00579  0000
00580  0000    2
00581  0000    .   Register 2-15 enthalten entsprechend die
00582  0000    .   lo-Bytes der Koordinaten von Sprite 1-7.
00583  0000   15
```

```
00584  0000
00585  0000   16   76543210  MSB (=Bit8) Abszisse Sprite7-0
00586  0000
00587  0000            viccr1 = $d011          VIC 8564 Control-Register 1
00588  0000
00589  0000   17   7.......  MSB (=Bit8) Raster-Wert Reg. 18
00590  0000        .6......  Extendet Color Text (0=aus, 1=ein)
00591  0000        ..5.....  Bit Map Mode (0=aus, 1=ein)
00592  0000        ...4....  Blank Screen to Color Border (0=Blank)
00593  0000        ....3...  24/25 Zeilen BS (0=24, 1=25)
00594  0000        .....210  Soft-Scroll vertikal, Dot-Bits 0-7
00595  0000
00596  0000            vicrwr = $d012          VIC 8564 read/write Raster
00597  0000
00598  0000   18   76543210  lo-Byte Raster-Vorgabe für Raster IRQ (WRITE)
00599  0000                   lo-Byte akt. Raster-Wert (READ)
00600  0000                   MSB (=Bit8) in Reg. 17, Bit7
00601  0000
00602  0000   19   76543210  Light-Pen, x-Wert (Abszisse)
00603  0000   20   76543210  Light-Pen, y-Wert (Ordinate)
00604  0000
00605  0000            vicmoe = $d015          VIC 8564 Sprite Enable Register
00606  0000
00607  0000   21   76543210  Sprite7-0 (1=ein)
00608  0000
00609  0000            viccr2 = $d016          VIC 8564 Control-Register 2
00610  0000
00611  0000   22   76......  unbenutzt
00612  0000        ..5.....  immer 0 setzen (reserviert für Weiterentw.)
00613  0000        ...4....  Priorität (0=aus, 1=ein)
00614  0000        ....3...  38/40 Spalten/Zeile (0=38, 1=40)
00615  0000        .....210  Soft-Scroll horizontal, Dot-Bits 0-7
00616  0000
00617  0000   23   76543210  Sprite7-0 vert. verdoppeln (1=ein)
00618  0000
00619  0000            vicmcr = $d018          VIC 8564 Memory-Control Register
00620  0000
00621  0000   24   7654....  Adreßbits 13-10 Basisadresse BS RAM
00622  0000        ....321.  Adreßbits 13-11 Basisadresse Chargen. ROM
00623  0000        .......0  unbenutzt
00624  0000
00625  0000            vicifr = $d019          VIC 8564 Interrupt Flag-Register
00626  0000
00627  0000   25   7.......  Interrupt-Flag (1=VIC IRQ)
00628  0000        .654....  unbenutzt
00629  0000        ....3...  Light-Pen IRQ (1=IRQ)
00630  0000        .....2..  Sprite/Sprite Kollision (1=IRQ)
00631  0000        ......1.  Sprite/Hintergrund Kollision (1=IRQ)
00632  0000        .......0  Raster-Vergleich IRQ (1=IRQ)
00633  0000
00634  0000            vicier = $d01a          VIC 8564 Interrupt Enable Register
00635  0000
00636  0000   26   7654....  unbenutzt
00637  0000        ....3...  Light-Pen IRQ (1=IRQ enabled)
00638  0000        .....2..  Sprite/Sprite Kollision (1=IRQ enabled)
00639  0000        ......1.  Sprite/Hintergrund Kollision (1=IRQ enabled)
00640  0000        .......0  Raster-Vergleich IRQ (1=IRQ enabled)
00641  0000
```

```
00642  0000   27   76543210  Priorität für Sprite7-0 (1=ein)
00643  0000
00644  0000   28   76543210  Multi-Color-Modus für Sprite7-0 (1=ein)
00645  0000
00646  0000   29   76543210  Sprite7-0 hor. verdoppeln (1=ein)
00647  0000
00648  0000   30   76543210  Sprite-Kollision mit Sprite7-0 (1=ein)
00649  0000
00650  0000   31   76543210  Hintergrund-Kollision mit Sprite7-0 (1=ein)
00651  0000
00652  0000   32   7654....  unbenutzt
00653  0000        ....3210  Rahmenfarbe (0-15)
00654  0000
00655  0000   33   7654....  unbenutzt
00656  0000        ....3210  Hintergrundfarbe #0 (0-15)
00657  0000
00658  0000   34   7654....  unbenutzt
00659  0000        ....3210  Hintergrundfarbe #1 (0-15)
00660  0000
00661  0000   35   7654....  unbenutzt
00662  0000        ....3210  Hintergrundfarbe #2 (0-15)
00663  0000
00664  0000   36   7654....  unbenutzt
00665  0000        ....3210  Hintergrundfarbe #3 (0-15)
00666  0000
00667  0000   37   7654....  unbenutzt
00668  0000        ....3210  Sprite Multicolor #0 (0-15)
00669  0000
00670  0000   38   7654....  unbenutzt
00671  0000        ....3210  Sprite Multicolor #1 (0-15)
00672  0000
00673  0000   39   7654....  unbenutzt
00674  0000        ....3210  Farbe Sprite0 (0-15)
00675  0000
00676  0000   40
00677  0000    .   Register 40-46 enthalten entsprechend die
00678  0000    .   Farben (0-15) der Sprites 1-7
00679  0000   46
00680  0000
00681  0000             vicpen = $d02f          VIC 8564 Port-Enable
00682  0000
00683  0000   47   7654....  unbenutzt
00684  0000        ....3210  VIC Tastatur-Port (K3-K0)
00685  0000
00686  0000             vicspd = $d030          VIC 8564 Speed-Register
00687  0000
00688  0000   48   7654321.  unbenutzt
00689  0000        .......0  Flag CPU-Takt (0 = 1 MHz, 1 = 2 MHz)
00690  0000
00691  0000
00692  0000                       .end
00693  0000                       .lib c128.sid
00694  0000
```

4.1.4 SID 6581 Sound Interface Device

```
00696  0000
00697  0000
00698  0000  ===> SID 6581 Sound Interface Device <===
00699  0000
00700  0000            sid    = $d400          Basisadresse SID 6581
00701  0000
00702  0000  ---- Stimme 1, Frequenz ----
00703  0000
00704  0000            sidf1l = $d400          SID 6581 Frequenz 1 low
00705  0000
00706  0000   76543210  lo-Byte Frequenz Stimme 1
00707  0000
00708  0000            sidf2h = $d401          SID 6581 Frequenz 1 high
00709  0000
00710  0000   76543210  hi-Byte Frequenz Stimme 1
00711  0000
00712  0000  ---- Stimme 1, Pulsbreite ----
00713  0000
00714  0000            sidt1l = $d402          SID 6581 Pulsbreite 1 low
00715  0000
00716  0000   76543210  lo-Byte Pulsbreite Stimme 1
00717  0000
00718  0000            sidt1h = $d403          SID 6581 Pulsbreite 1 high
00719  0000
00720  0000   7654....  unbenutzt
00721  0000   ....3210  hi-Byte Nybble Pulsbreite Stimme 1
00722  0000
00723  0000  ---- Stimme 1, Control Register ----
00724  0000
00725  0000            sidct1 = $d404          SID 6581 Control-Register 1
00726  0000
00727  0000   7.......  1 = Zufallsgenerator ein
00728  0000   .6......  1 = Rechteckschwingung ein
00729  0000   ..5.....  1 = Sägezahnschwingung ein
00730  0000   ...4....  1 = Dreieckschwingung ein
00731  0000   ....3...  1 = Oszillator 1 aus
00732  0000   .....2..  1 = Stimme 1 mit 3 ring-modulieren
00733  0000   ......1.  1 = Stimme 1 mit 3 synchronisieren
00734  0000   .......0  1 = Start Att/Dec/Sus, 0 = Start Rel.
00735  0000
00736  0000  ---- Stimme 1, Att.-Dec. ----
00737  0000
00738  0000            sidad1 = $d405          SID 6581 Attack/Decay-Register 1
00739  0000
00740  0000   7654....  Attack - Zeit (0-15)
00741  0000   ....3210  Decay  - Zeit (0-15)
00742  0000
00743  0000  ---- Stimme 1, Sus.-Rel. ----
00744  0000
00745  0000            sidsr1 = $d406          SID 6581 Sustain/Release-Register 1
00746  0000
00747  0000   7654....  Sustain - Zeit (0-15)
00748  0000   ....3210  Release - Zeit (0-15)
00749  0000
00750  0000
```

```
00751  0000
00752  0000  ---- Stimme 2 ----
00753  0000
00754  0000  Mit den Registern 7-13 ($d407-$d40d) wird Stimme 2 analog
00755  0000  gehandhabt, dabei gilt für Register 11 (Control-Register):
00756  0000
00757  0000   .....2..  1 = Stimme 2 mit 1 ring-modulieren
00758  0000   ......1.  1 = Stimme 2 mit 1 synchronisieren
00759  0000
00760  0000  ---- Stimme 3 ----
00761  0000
00762  0000  Mit den Registern 14-20 ($d40e-$d414) wird Stimme 3 analog
00763  0000  gehandhabt, dabei gilt für Register 18 (Control-Register):
00764  0000
00765  0000   .....2..  1 = Stimme 3 mit 2 ring-modulieren
00766  0000   ......1.  1 = Stimme 3 mit 2 synchronisieren
00767  0000
00768  0000  ---- Filter-Modus und Lautstärke ----
00769  0000
00770  0000            sidvol = $d418          SID 6581 Modus/Lautstärke-Register
00771  0000
00772  0000   7.......  1 = Stimme 3 abgeschaltet
00773  0000   .6......  1 = Filter Hochpaß ein
00774  0000   ..5.....  1 = Filter Bandpaß ein
00775  0000   ...4....  1 = Filter Tiefpaß ein
00776  0000   ....3210  Lautstärke (0-15)
00777  0000
00778  0000  ---- A/D Wandler 1 ----
00779  0000
00780  0000            sidptx = $d419          SID 6581 A/D-Wandler 1
00781  0000
00782  0000  ---- A/D Wandler 2 ----
00783  0000
00784  0000            sidpty = $d41a          SID 6581 A/D-Wandler 2
00785  0000
00786  0000  ---- Rauschgenerator Stimme 3 ----
00787  0000
00788  0000            sidrnd = $d41b          SID 6581 Rauschgenerator Stimme 3
00789  0000
00790  0000  ---- Hüllkurvengenerator Stimme 3 ----
00791  0000
00792  0000            sidenv = $d41c          SID 6581 Hüllkurvengen. Stimme 3
00793  0000
00794  0000
00795  0000                  .end
00796  0000                  .lib c128.mmu
00797  0000
```

4.1.5 MMU 8722 Memory Management Unit

```
00799  0000
00800  0000
00801  0000  ===> MMU 8722 Memory Management Unit <===
00802  0000
00803  0000  ---- Configuration Register ----
00804  0000
```

```
00805  0000            mmucrl = $d500          MMU 8722 Configuration Register
00806  0000            mmucr  = $ff00          MMU 8722 Config. Register (Kopie)
00807  0000
00808  0000   76......  Auswahl der 64 K RAM-Bank:
00809  0000
00810  0000   00xxxxxx  RAM Bank 0
00811  0000   01xxxxxx  RAM Bank 1
00812  0000   10xxxxxx  RAM Bank 2 (wenn nicht vorhanden: Bank 0)
00813  0000   11xxxxxx  RAM Bank 3 (wenn nicht vorhanden: Bank 1)
00814  0000
00815  0000   ..54....  Auswahl RAM/ROM Adreßbereich $c000 - $ffff
00816  0000
00817  0000   xx00xxxx  Kernal ROM
00818  0000   xx01xxxx  Internal Function ROM
00819  0000   xx10xxxx  External Function ROM
00820  0000   xx11xxxx  RAM Bank, gewählt mit Bit7/6
00821  0000
00822  0000   ....32..  Auswahl RAM/ROM Adreßbereich $8000 - $bfff
00823  0000
00824  0000   xxxx00xx  Teil BASIC ROM
00825  0000   xxxx01xx  Internal Function ROM
00826  0000   xxxx10xx  External Function ROM
00827  0000   xxxx11xx  RAM Bank, gewählt mit Bit7/6
00828  0000
00829  0000   ......1.  Auswahl RAM/ROM Adreßbereich $4000 - $7fff
00830  0000
00831  0000   xxxxxx0x  Teil BASIC ROM
00832  0000   xxxxxx1x  RAM Bank, gewählt mit Bit7/6
00833  0000
00834  0000   .......0  Auswahl RAM/ROM-I/O Adreßbereich $d000 - $dfff
00835  0000
00836  0000   xxxxxxx0  I/O-Chips
00837  0000   xxxxxxx1  RAM/ROM/Char.gen., gewählt mit Bit5/4
00838  0000
00839  0000
00840  0000  ---- Preconfiguration- / Load Configuration Register ----
00841  0000
00842  0000            pcra   = $d501          Preconfiguration Register A
00843  0000            lcra   = $ff01          Load Configuration Register A
00844  0000
00845  0000            pcrb   = $d502          Preconfiguration Register B
00846  0000            lcrb   = $ff02          Load Configuration Register B
00847  0000
00848  0000            pcrc   = $d503          Preconfiguration Register C
00849  0000            lcrc   = $ff03          Load Configuration Register C
00850  0000
00851  0000            pcrd   = $d504          Preconfiguration Register D
00852  0000            lcrd   = $ff04          Load Configuration Register D
00853  0000
00854  0000  In die Preconfigurations Register A-D wird ein Bitmuster geschrieben,
00855  0000  das durch Schreiben eines beliebigen Wertes in das entsprechende Load
00856  0000  Configuration Register ins Configurationsregister übernommen wird
00857  0000
00858  0000
00859  0000  ---- MMU 8722 Modus Configurations Register ----
00860  0000
00861  0000            mmumcr = $d505          MMU 8722 Modus Configurations
                                               Register
```

```
00862  0000
00863  0000   7.......  Flag für 40/80-Zeichen-BS - Taste:
00864  0000
00865  0000   0xxxxxxx  80-Zeichen BS - Taste nicht gedrückt
00866  0000   1xxxxxxx  40-Zeichen BS - Taste gedrückt
00867  0000
00868  0000   .6......  Flag für C 128 / C 64 - Modus:
00869  0000
00870  0000   x0xxxxxx  C 128 - Modus
00871  0000   x1xxxxxx  C 64 - Modus
00872  0000
00873  0000   ..5.....  Flag für Steckmodul GAME (C 64):
00874  0000
00875  0000   xx0xxxxx  GAME belegt
00876  0000   xx1xxxxx  GAME nicht belegt
00877  0000
00878  0000   ..5.....  Flag für Umschaltung Farbspeicher (C 128):
00879  0000
00880  0000   xx0xxxxx  unbekannt
00881  0000   xx1xxxxx  unbekannt
00882  0000
00883  0000   ...4....  Flag für Steckmodul EXROMIN (C 64):
00884  0000
00885  0000   xxx0xxxx  EXROMIN belegt
00886  0000   xxx1xxxx  EXROMIN nicht belegt
00887  0000
00888  0000   ....3...  FSDIR-Control Bit (FAST SERIAL DEVICE)
00889  0000   xxxx0xxx  FSD Input
00890  0000   xxxx1xxx  FSD Output
00891  0000
00892  0000   .....21.  noch nicht verwendet
00893  0000
00894  0000   .......0  Auswahl des aktiven Prozessors:
00895  0000
00896  0000   xxxxxxx0  Z 80
00897  0000   xxxxxxx1  8502
00898  0000
00899  0000
00900  0000  ---- RAM Configuration Register ----
00901  0000
00902  0000            mmurcr = $d506          RAM Configuration Register
00903  0000
00904  0000   76......  Auswahl RAM-Bank für VIC-Chip
00905  0000
00906  0000   00xxxxxx  RAM Bank 0
00907  0000   01xxxxxx  RAM Bank 1
00908  0000   10xxxxxx  RAM Bank 2 (wenn nicht vorhanden: Bank 0)
00909  0000   11xxxxxx  RAM Bank 3 (wenn nicht vorhanden: Bank 1)
00910  0000
00911  0000   ..54....  Auswahl 256K-Block (spätere 1MB-Version):
00912  0000
00913  0000   xx00xxxx  256K-Block 0
00914  0000   xx01xxxx  256K-Block 1
00915  0000   xx10xxxx  256K-Block 2
00916  0000   xx11xxxx  256K-Block 3
00917  0000
00918  0000   ....32..  Definition COMMON RAM AREA:
00919  0000
```

```
00920  0000   xxxx00xx  keine  COMMON AREA, unabhängig von Bits1/0
00921  0000   xxxx01xx  untere COMMON AREA
00922  0000   xx10xxxx  obere  COMMON AREA
00923  0000   xx11xxxx  untere und obere COMMON AREA
00924  0000
00925  0000   ......10  Definition Größe der COMMON RAM AREA:
00926  0000
00927  0000   xxxxxx00   1 K
00928  0000   xxxxxx01   4 K
00929  0000   xxxxxx10   8 K
00930  0000   xxxxxx11  16 K
00931  0000
00932  0000
00933  0000  ---- Zeropage Zeiger ----
00934  0000
00935  0000            mmup0l = $d507          Page-0-Pointer lo
00936  0000            mmup0h = $d508          Page-0-Pointer hi
00937  0000
00938  0000  ---- Stapel Zeiger ----
00939  0000
00940  0000            mmup1l = $d509          Page-1-Pointer lo
00941  0000            mmup1h = $d50a          Page-1-Pointer hi
00942  0000
00943  0000  ---- Versions Register ----
00944  0000
00945  0000            mmuver = $d50b          Versions-Register
00946  0000
00947  0000   7654....  Bank-Version
00948  0000
00949  0000   0010xxxx   2 64K Banks (kommentierter Rechner)
00950  0000   bis
00951  0000   1111xxxx  15 64K Banks (zukünftige Versionen)
00952  0000
00953  0000   ....3210  MMU - Version
00954  0000
00955  0000   xxxx0000  der kommentierte Rechner besitzt MMU V 0
00956  0000
00957  0000
00958  0000                    .end
00959  0000                    .lib c128.vdc
00960  0000
```

4.1.6 VDC 8563 Video Display Controller

```
00962  0000
00963  0000
00964  0000  ===> VDC 8563 Video Display Controller <===
00965  0000
00966  0000  Der Video-Controller VDC 8563 besitzt intern 16 KByte RAM für
00967  0000  Zeichengenrator und Attribute.
00968  0000
00969  0000            vdcscn = $0000          Basisadresse VDC BS-RAM
00970  0000            vdccol = $0800          Basisadresse VDC Attribut-RAM
00971  0000            vdcchr = $2000          Basisadresse VDC Zeichengenerator
00972  0000
00973  0000            vdcadr = $d600          VDC 8563 Adreßregister
```

```
00974 0000
00975 0000  WRITE: ..543210  Nummer gewähltes Register (0-36)
00976 0000
00977 0000  READ : 7.......  Statusbit des VDC
00978 0000         0xxxxxxx  letzter VDC-Job noch in Arbeit
00979 0000         1xxxxxxx  VDC bereit für nächsten Job
00980 0000
00981 0000  Da die Firma COMMODORE mit Informationen über von ihr entwickelte
00982 0000  Chips äußerst sparsam umgeht, sind über die weiteren Statusbits
00983 0000  nur Spekulationen möglich (Vergleich mit 6845):
00984 0000
00985 0000         .6......  Status Light-Pen Register:
00986 0000         x0xxxxxx  nach Lesen Reg. 16/17 (LPEN)
00987 0000         x1xxxxxx  Light-Pen Strobe festgestellt
00988 0000
00989 0000         ..5.....  Vertical Blanking: Kathodenstrahl
00990 0000         xx0xxxxx  nicht im Rücklauf
00991 0000         xx1xxxxx  im Rücklauf
00992 0000
00993 0000  Die Bits 4-0 werden im 6845 nicht benutzt. Im C 128 werden Bits3-0
00994 0000  des 8563 abgefragt ($e17e). Die Bedeutung ist jedoch unklar.
00995 0000
00996 0000            vdcdat = $d601          VDC 8563 Datenregister
00997 0000
00998 0000  Der Video-Controller VDC 8563 belegt nur die Adressen $d600 und $d601,
00999 0000  verwaltet jedoch intern 36 Register.
01000 0000
01001 0000  Auf die Register wird zum Schreiben/Lesen zugegriffen, indem die ge-
01002 0000  wünschte Registernummer ins Adreßregister geschrieben und das Daten-
01003 0000  register geschrieben/gelesen wird.
01004 0000
01005 0000  Die einzelnen Register haben folgende Bedeutung:
01006 0000
01007 0000  Reg.    Bits
01008 0000  ==============
01009 0000
01010 0000    0   76543210  gesamte Anzahl der Zeichen zwischen aufeinander-
01011 0000                  folgenden horizontalen Synchronimpulsen-1
01012 0000
01013 0000    1   76543210  Anzahl Zeichen pro Bildschirmzeile (BS-Zeile)
01014 0000
01015 0000    2   76543210  Position des horizontalen Synchronimpulses
01016 0000
01017 0000    3   7654....  Breite des vertikalen Synchronimpulses
01018 0000        ....3210  Breite des horizontalen Synchronimpulses
01019 0000                  (Zeichengenerator-Taktzyklen+1)
01020 0000
01021 0000    4   76543210  Gesamtzahl BS-Zeilen-1
01022 0000
01023 0000    5   ...43210  Feinabgleich (Anzahl der Bildzeilen)
01024 0000
01025 0000    6   76543210  Anzahl der dargestellten Zeilen
01026 0000
01027 0000    7   76543210  Position des vertikalen Synchronimpulses
01028 0000                  (Anzahl Zeichenzeilen)
01029 0000
01030 0000    8   ......10  Anzeigeform:
01031 0000        xxxxxxx0  kein Zeilensprung
```

```
01032  0000        xxxxxx01  Zeilensprung
01033  0000        xxxxxx11  Zeilensprung doppelte Dichte
01034  0000
01035  0000   9    ...43210  Anzahl der Bildzeilen-1 pro BS-Zeile
01036  0000
01037  0000   10   .65.....  Cursor-Modus:
01038  0000        x00xxxxx  Cursor blinkt nicht
01039  0000        x01xxxxx  Cursor ausgeschaltet
01040  0000        x10xxxxx  Cursor blinkt schnell (1/16 Bildwiederholfrequenz)
01041  0000        x11xxxxx  Cursor blinkt langsam (1/32 Bildwiederholfrequenz)
01042  0000
01043  0000        ...43210  Cursor-Anfangszeile im Zeichen (0-7)
01044  0000   11   ...43210  Cursor-Endzeile im Zeichen (0-7)
01045  0000
01046  0000   12   76543210  addrh Startadresse BS-Speicher (Video-RAM)
01047  0000   13   76543210  addrl Startadresse BS-Speicher (Video-RAM)
01048  0000
01049  0000   14   76543210  addrh akt. Cursorposition im BS-Speicher
01050  0000   15   76543210  addrl akt. Cursorposition im BS-Speicher
01051  0000
01052  0000   16   76543210  vertikale Lichtgriffel-Pos. (in Bildzeilen+1)
01053  0000   17   76543210  horizontale Lichtgriffel-Pos. (in Bildspalten+1)
01054  0000
01055  0000   18   76543210  addrh akt. Schreib-/Leseposition im BS-Speicher
01056  0000   19   76543210  addrl akt. Schreib-/Leseposition im BS-Speicher
01057  0000
01058  0000   20   76543210  addrh Startadresse Attribut-Speicher
01059  0000   21   76543210  addrl Startadresse Attribut-Speicher
01060  0000
01061  0000   22   7654....  Zeichenbreite in Pixel-1 (einschl. Zwischenraum)
01062  0000        ....3210  Zeichenbreite in Pixel-1 (ohne Zwischenraum)
01063  0000
01064  0000   23   ...43210  Anzahl Bildzeilen/BS-Zeile (ohne Zwischenraum)
01065  0000
01066  0000   24   7.......  Block Copy
01067  0000        0xxxxxxx  Block wird geschr. nach vorausgeg. Schreiben Reg. 30
01068  0000        1xxxxxxx  Block wird kopiert nach vorausgeg. Schreiben Reg. 30
01069  0000
01070  0000        .6......
01071  0000        x0xxxxxx  Normaldarstellung
01072  0000        x1000000  REVERS-Modus
01073  0000
01074  0000        ..5.....  Blinkfrequenz der Bildschirmzeichen
01075  0000        xx0xxxxx  schnell (1/16 Bildwiederholfrequenz)
01076  0000        xx1xxxxx  langsam (1/32 Bildwiederholfrequenz)
01077  0000
01078  0000        ...43210  Anzahl der BS-Zeilen, die BS nach oben kopiert wird
01079  0000
01080  0000   25   7.......  Betriebsart
01081  0000        0xxxxxxx  Text-Modus
01082  0000        1xxxxxxx  Bit-Map-Modus
01083  0000
01084  0000        .6......  Attribute
01085  0000        x0xxxxxx  kein Attribut, Vordergrundfarbe lt. Reg. 26
01086  0000        x1xxxxxx  Vordergrundfarbe und Attribut lt. Reg. 26
01087  0000
01088  0000        ..5.....  Semigraphik-Modus
01089  0000        xx0xxxxx  horiz. Zwischenraum zweier Zeichen in Hintergrundfarbe
```

```
01090  0000          xx1xxxxx  horiz. Zwischenraum wie letztes normal dargest. Zeichen
01091  0000
01092  0000          ...4....  Pixelgröße
01093  0000          xxx0xxxx  = 1 Dot-Takt
01094  0000          xxx1xxxx  = 2 Dot-Takte
01095  0000
01096  0000          ....3210  Anzahl Pixel, um die BS nach links geschoben wird
01097  0000
01098  0000    26    7654....  Vordergrundfarbe, wenn Bit6 von Reg. 25 = 0
01099  0000          ....3210  Hintergrundfarbe für alle Zeichen und Rahmenfarbe
01100  0000
01101  0000    27    76543210  Anzahl Zeichen/BS-Zeile
01102  0000
01103  0000    28    765.....  Adreßbits 15-13 des Zeichengenerators
01104  0000
01105  0000          ...4....  DRAM-Type
01106  0000          xxx0xxxx  4416
01107  0000          xxx1xxxx  4164
01108  0000
01109  0000    29    ...43210  BS-Zeilen Zähler für Unterstreichen
01110  0000
01111  0000    30    76543210  Anzahl Block-Write- bzw. Block-Copy-Zyklen
01112  0000
01113  0000    31    76543210  Daten für Block-Write bzw. normales Write
01114  0000
01115  0000    32    76543210  addrh Quelladresse Block-Copy
01116  0000    33    76543210  addrl Quelladresse Block-Copy
01117  0000
01118  0000    34    76543210  Zeichen von Anfang akt. BS-Zeile bis pos. Flanke,
01119  0000    35    76543210  Zeichen von Anfang akt. BS-Zeile bis neg. Flanke
01120  0000                    des Display-Enable-Pins
01121  0000
01122  0000    36    ....3210  Anzahl der Refresh-Zyklen pro BS-Zeile
01123  0000
01124  0000
01125  0000                      .end
01126  0000                      .lib c128.cia1
01127  0000
```

4.1.7 CIA 6526 Complex Interface Adapter

```
01129  0000
01130  0000  ===> CIA 6526 Complex Interface Adapter  -1- <===
01131  0000
01132  0000            cia1   = $dc00          Basisadresse CIA 6526 -1-
01133  0000
01134  0000  ---- Peripheral Data Register A (PRA) ----
01135  0000
01136  0000            d1pra  = $dc00          Schreib-/Lese-Register Port A
01137  0000
01138  0000   76543210  Spalten Tastaturmatrix (OUTPUTS)
01139  0000
01140  0000  außerdem:
01141  0000   76......  Auswahl Paddle-Set A/B
01142  0000   ...43210  Joystick 0
01143  0000
```

```
01144  0000  ---- Peripheral Data Register B (PRB) ----
01145  0000
01146  0000            rows   = $dc01          Schreib-/Leseregister Port B
01147  0000
01148  0000   76543210  Reihen Tastaturmatrix
01149  0000
01150  0000  außerdem:
01151  0000   ...43210  Joystick 1
01152  0000
01153  0000  ---- Data Direction Register A (DDRA) ----
01154  0000
01155  0000            d1ddra = $dc02          CIA1 Datenrichtungs-Reg. a
01156  0000
01157  0000   11111111  alle Bits als Ausgänge geschaltet
01158  0000            zum Scannen der Tastatur
01159  0000
01160  0000  ---- Data Direction Register B (DDRB) ----
01161  0000
01162  0000            d1ddrb = $dc03          CIA1 Datenrichtungs-Reg. b
01163  0000
01164  0000   00000000  alle Bits als Eingänge geschaltet
01165  0000            zum Scannen der Tastatur
01166  0000
01167  0000  ---- Timer A Low / High Register (TALO/TAHI) ----
01168  0000
01169  0000            d1t1l  = $dc04          benutzt für Serial Device
01170  0000            d1t1h  = $dc05          und Kassettenrekorder
01171  0000
01172  0000  ---- Timer B Low / High Register (TBLO/TBHI) ----
01173  0000
01174  0000            d1t2l  = $dc06          CIA1 TIMER B low
01175  0000            d1t2h  = $dc07          CIA1 TIMER B high
01176  0000
01177  0000  ---- Time of Day (TOD) Zehntelsekunden ----
01178  0000
01179  0000            tod10  = $dc08
01180  0000
01181  0000   7654....  unbenutzt
01182  0000   ....3210  Zehntelsekunden (BCD)
01183  0000
01184  0000
01185  0000  ---- Time of Day (TOD) Sekunden ----
01186  0000
01187  0000            todsec = $dc09
01188  0000
01189  0000   7654....  Sekunden hi (BCD)
01190  0000   ....3210  Sekunden lo (BCD)
01191  0000
01192  0000  ---- Time of Day (TOD) Minuten ----
01193  0000
01194  0000            todmin = $dc0a
01195  0000
01196  0000   7654....  Minuten hi (BCD)
01197  0000   ....3210  Minuten lo (BCD)
01198  0000
01199  0000  ---- Time of Day (TOD) Stunden ----
01200  0000
01201  0000            todhr  = $dc0b
```

```
01202  0000
01203  0000   7.......  PM
01204  0000   0xxxxxxx  AM
01205  0000   1xxxxxxx  PM
01206  0000
01207  0000   .65.....  unbenutzt
01208  0000   ...4....  Stunden hi
01209  0000   ....3210  Stunden lo (BCD)
01210  0000
01211  0000
01212  0000  ---- Serial Data Register (SDR) ----
01213  0000
01214  0000             d1sdr  = $dc0c          CIA1 Serial Data Register
01215  0000
01216  0000   76543210  Serial Data
01217  0000
01218  0000  ---- Interrupt Control Register (ICR) ----
01219  0000
01220  0000             d1icr  = $dc0d          CIA1 Interrupt Control-Register
01221  0000
01222  0000  READ:   7.......  IR IRQ-Flag (1=IRQ)
01223  0000          .65.....  unbenutzt (immer 0)
01224  0000
01225  0000          ...43210  IRQ-Flags:  1=IRQ
01226  0000                                0=NO IRQ
01227  0000          ...4....  FLAG-IRQ (Flag-Input =1)
01228  0000          ....3...  SERIAL PORT (full/empty)
01229  0000          .....2..  ALARM (Alarmzeit erreicht)
01230  0000          ......1.  TIMER B (underflow)
01231  0000          .......0  TIMER A (underflow)
01232  0000
01233  0000  WRITE:  7.......  SET/CLEAR Bit
01234  0000          0xxxxxxx  CLEAR alle Maskenbit =1
01235  0000          1xxxxxxx  SET   alle Maskenbit =1
01236  0000
01237  0000          .65.....  unbenutzt
01238  0000
01239  0000          ...43210  Maskenbits: 1=SET/CLEAR
01240  0000                                0=UNAFFECTED
01241  0000          ...4....  Maskenbit FLAG IRQ
01242  0000          ....3...  Maskenbit SERIAL PORT
01243  0000          .....2..  Maskenbit ALARM
01244  0000          ......1.  Maskenbit TIMER B
01245  0000          .......0  Maskenbit TIMER A
01246  0000
01247  0000  ---- Control Register A (CRA) ----
01248  0000
01249  0000             d1cra  = $dc0e          CIA1 Control-Register a
01250  0000
01251  0000   7.......  TODIN
01252  0000   0xxxxxxx  60 Hz an TOD-Pin
01253  0000   1xxxxxxx  50 Hz an TOD-Pin
01254  0000
01255  0000   .6......  SPMODE
01256  0000   x0xxxxxx  SERIAL PORT INPUT (ext. Shift-Takt erforderl.)
01257  0000   x1xxxxxx  SERIAL PORT OUTPUT (CNT als Shift-Takt)
01258  0000
01259  0000   ..5.....  INMODE
```

```
01260  0000   xx0xxxxx  TIMER A zählt PHI2 Pulse
01261  0000   xx1xxxxx  TIMER A zählt positive CNT-Übergänge
01262  0000
01263  0000   ...4....  LOAD
01264  0000   xxx0xxxx  wirkungslos
01265  0000   xxx1xxxx  FORCE LOAD
01266  0000
01267  0000   ....3...  RUNMODE
01268  0000   xxxx0xxx  CONTINOUS
01269  0000   xxxx1xxx  ONE-SHOT
01270  0000
01271  0000   .....2..  OUTMODE
01272  0000   xxxxx0xx  PULSE
01273  0000   xxxxx1xx  TOGGLE
01274  0000
01275  0000   ......1.  PBON Steuerung PB6
01276  0000   xxxxxx0x  PB6 (Bit6 Port B) normal Operation
01277  0000   xxxxxx1x  PB6 ist TIMER A output
01278  0000
01279  0000   .......0  START TIMER A
01280  0000   xxxxxxx0  Stop TIMER A
01281  0000   xxxxxxx1  Start TIMER A
01282  0000
01283  0000
01284  0000  ---- Control Register B (CRB) ----
01285  0000
01286  0000            d1crb  = $dc0f          CIA1 Control-Register b
01287  0000
01288  0000   7.......  ALARM
01289  0000   0xxxxxxx  Write TOD-Register setzt Uhr
01290  0000   1xxxxxxx  Write TOD-Register setzt Alarm
01291  0000
01292  0000   .65.....  INMODE
01293  0000   x00xxxxx  TIMER B zählt PHI2-Pulse
01294  0000   x01xxxxx  TIMER B zählt pos. CNT-Übergänge
01295  0000   x10xxxxx  TIMER B zählt TIMER A underflow-Pulse
01296  0000   x11xxxxx  TIMER B zählt TIMER A underflow-Pulse wenn CNT hi
01297  0000
01298  0000   ...4....  LOAD
01299  0000   xxx0xxxx  wirkungslos
01300  0000   xxx1xxxx  FORCE LOAD
01301  0000
01302  0000   ....3...  RUNMODE
01303  0000   xxxx0xxx  CONTINOUS
01304  0000   xxxx1xxx  ONE-SHOT
01305  0000
01306  0000   .....2..  OUTMODE
01307  0000   xxxxx0xx  PULSE
01308  0000   xxxxx1xx  TOGGLE
01309  0000
01310  0000   ......1.  PBON Steuerung PB7
01311  0000   xxxxxx0x  PB7 (Bit7 Port B) normal Operation
01312  0000   xxxxxx1x  PB7 ist TIMER B output
01313  0000
01314  0000   .......0  START TIMER B
01315  0000   xxxxxxx0  Stop TIMER B
01316  0000   xxxxxxx1  Start TIMER B
01317  0000
```

```
01318  0000
01319  0000                    .end
01320  0000                    .lib c128.cia2
01321  0000
01323  0000
01324  0000  ===> CIA 6526 Complex Interface Adapter  -2- <===
01325  0000
01326  0000             cia2   = $dd00          Basisadresse CIA 6526 -2-
01327  0000
01328  0000  ---- Peripheral Data Register A (PRA) ----
01329  0000
01330  0000             d2pra  = $dd00          Schreib-/Lese-Register Port A
01331  0000
01332  0000   7.......  DATA Input Serial Device
01333  0000   .6......  CLOCK Input Serial Device
01334  0000   ..5.....  DATA Output Serial Device
01335  0000   ...4....  CLOCK Output Serial Device
01336  0000   ....3...  ATN Output Serial Device
01337  0000   .....2..  TXD Transmit Date (RS 232 Kassette erforderl.)
01338  0000   ......10  Adressbits 15/14 BS-RAM
01339  0000
01340  0000  ---- Peripheral Data Register B (PRB) ----
01341  0000
01342  0000             d2prb  = $dd01          Schreib-/Lese-Register Port B
01343  0000
01344  0000   7.......  DSR Data Set Ready
01345  0000   .6......  CTS Clear to Send
01346  0000   ..5.....  DCD Data Carrier Detect
01347  0000   ...4....
01348  0000   ....3...  RI  Ring Indicator
01349  0000   .....2..  DTR Data Terminal Ready
01350  0000   ......1.  RTS Request to Send
01351  0000   .......0  RXD Received Data
01352  0000
01353  0000  ---- Data Direction Register A (DDRA) ----
01354  0000
01355  0000             d2ddra = $dd02          CIA2 Datenrichtungs-Reg. A
01356  0000
01357  0000   0xxxxxxx  INPUT  DATA Serial Device
01358  0000   x0xxxxxx  INPUT  CLOCK Serial Device
01359  0000   xx1xxxxx  OUTPUT DATA Serial Device
01360  0000   xxx1xxxx  OUTPUT CLOCK Serial Device
01361  0000   xxxx1xxx  OUTPUT ATN Serial Device
01362  0000   xxxxx1xx  OUTPUT TXD Transmit Data
01363  0000   xxxxxx11  OUTPUT Adressbits 15/14 BS-RAM
01364  0000
01365  0000
01366  0000  ---- Data Direction Register B (DDRB) ----
01367  0000
01368  0000             d2ddrb = $dd03          CIA2 Datenrichtungs-Reg. B
01369  0000
01370  0000   0xxxxxxx  INPUT  DSR Data Set Ready
01371  0000   x0xxxxxx  INPUT  CTS Clear to Send
01372  0000   xx0xxxxx  INPUT  DCD Data Carrier Detect
01373  0000   xxx0xxxx  INPUT  unbenutzt
01374  0000   xxxx0xxx  INPUT  RI  Ring Indicator
01375  0000   xxxxx1xx  OUTPUT DTR Data Terminal Ready
01376  0000   xxxxxx1x  OUTPUT RTS Request to Send
```

```
01377  0000   xxxxxxx0  INPUT  RXD Received Data
01378  0000
01379  0000
01380  0000
01381  0000  ---- Timer A Low / High Register (TALO/TAHI) ----
01382  0000
01383  0000            d2t1l  = $dd04          benutzt für Serial Device
01384  0000            d2t1h  = $dd05          und Kassettenrekorder
01385  0000
01386  0000  ---- Timer B Low / High Register (TBLO/TBHI) ----
01387  0000
01388  0000            d2t2l  = $dd06          CIA2 TIMER B low
01389  0000            d2t2h  = $dd07          CIA2 TIMER B high
01390  0000
01391  0000  ----  TIME OF DAY ----
01392  0000
01393  0000  Register $dd08 bis $dd0b wie CIA1
01394  0000  (vom System nicht benutzt)
01395  0000
01396  0000  ----  Serial Data Register ----
01397  0000
01398  0000  Register $dd0c wie CIA1
01399  0000  (vom System nicht benutzt)
01400  0000
01401  0000  ---- Interrupt Control Register (ICR) ----
01402  0000
01403  0000            d2icr  = $dd0d          CIA2 Interrupt Control-Register
01404  0000
01405  0000  READ:   7.......  IR IRQ-Flag (1=IRQ)
01406  0000         .65.....  unbenutzt (immer 0)
01407  0000
01408  0000         ...43210  IRQ-Flags:  1=IRQ
01409  0000                                0=NO IRQ
01410  0000         ...4....  FLAG-IRQ (Flag-Input =1)
01411  0000         ....3...  SERIAL PORT (full/empty)
01412  0000         .....2..  ALARM (Alarmzeit erreicht)
01413  0000         ......1.  TIMER B (underflow)
01414  0000         .......0  TIMER A (underflow)
01415  0000
01416  0000  WRITE:  7.......  SET/CLEAR Bit
01417  0000         0xxxxxxx  CLEAR alle Maskenbit =1
01418  0000         1xxxxxxx  SET   alle Maskenbit =1
01419  0000
01420  0000         .65.....  unbenutzt
01421  0000
01422  0000         ...43210  Maskenbits: 1=SET/CLEAR
01423  0000                                0=UNAFFECTED
01424  0000         ...4....  Maskenbit FLAG IRQ
01425  0000         ....3...  Maskenbit SERIAL PORT
01426  0000         .....2..  Maskenbit ALARM
01427  0000         ......1.  Maskenbit TIMER B
01428  0000         .......0  Maskenbit TIMER A
01429  0000
01430  0000  ---- Control Register A (CRA) ----
01431  0000
01432  0000            d2cra  = $dd0e          CIA2 Control-Register A
01433  0000
01434  0000   7.......  TODIN
```

```
01435  0000   0xxxxxxx  60 Hz an TOD-Pin
01436  0000   1xxxxxxx  50 Hz an TOD-Pin
01437  0000
01438  0000   .6......  SPMODE
01439  0000   x0xxxxxx  SERIAL PORT INPUT (ext. Shift-Takt erforderl.)
01440  0000   x1xxxxxx  SERIAL PORT OUTPUT (CNT als Shift-Takt)
01441  0000
01442  0000   ..5.....  INMODE
01443  0000   xx0xxxxx  TIMER A zählt PHI2 Pulse
01444  0000   xx1xxxxx  TIMER A zählt positive CNT-Übergänge
01445  0000
01446  0000   ...4....  LOAD
01447  0000   xxx0xxxx  wirkungslos
01448  0000   xxx1xxxx  FORCE LOAD
01449  0000
01450  0000   ....3...  RUNMODE
01451  0000   xxxx0xxx  CONTINOUS
01452  0000   xxxx1xxx  ONE-SHOT
01453  0000
01454  0000   .....2..  OUTMODE
01455  0000   xxxxx0xx  PULSE
01456  0000   xxxxx1xx  TOGGLE
01457  0000
01458  0000   ......1.  PBON Steuerung PB6
01459  0000   xxxxxx0x  PB6 (Bit6 Port B) normal Operation
01460  0000   xxxxxx1x  PB6 ist TIMER A output
01461  0000
01462  0000   .......0  START TIMER A
01463  0000   xxxxxxx0  Stop TIMER A
01464  0000   xxxxxxx1  Start TIMER A
01465  0000
01466  0000
01467  0000  ---- Control Register B (CRB) ----
01468  0000
01469  0000            d2crb  = $dd0f          CIA2 Control-Register B
01470  0000
01471  0000   7.......  ALARM
01472  0000   0xxxxxxx  Write TOD-Register setzt Uhr
01473  0000   1xxxxxxx  Write TOD-Register setzt Alarm
01474  0000
01475  0000   .65.....  INMODE
01476  0000   x00xxxxx  TIMER B zählt PHI2-Pulse
01477  0000   x01xxxxx  TIMER B zählt pos. CNT-Übergänge
01478  0000   x10xxxxx  TIMER B zählt TIMER A underflow-Pulse
01479  0000   x11xxxxx  TIMER B zählt TIMER A underflow-Pulse wenn CNT hi
01480  0000
01481  0000   ...4....  LOAD
01482  0000   xxx0xxxx  wirkungslos
01483  0000   xxx1xxxx  FORCE LOAD
01484  0000
01485  0000   ....3...  RUNMODE
01486  0000   xxxx0xxx  CONTINOUS
01487  0000   xxxx1xxx  ONE-SHOT
01488  0000
01489  0000   .....2..  OUTMODE
01490  0000   xxxxx0xx  PULSE
01491  0000   xxxxx1xx  TOGGLE
01492  0000
```

```
01493  0000   ......1.  PBON Steuerung PB7
01494  0000   xxxxxx0x  PB7 (Bit7 Port B) normal Operation
01495  0000   xxxxxx1x  PB7 ist TIMER B output
01496  0000
01497  0000   .......0  START TIMER B
01498  0000   xxxxxxx0  Stop TIMER B
01499  0000   xxxxxxx1  Start TIMER B
01500  0000
01501  0000
01502  0000                    .end
01503  0000                    .lib c128.screen1
01504  0000
```

4.1.8 Bildschirm-Editor

```
01506  0000
01507  0000
01508  0000                    * = $c000
01509  c000
01510  c000   ===> Sprungverteiler EDITOR <===
01511  c000
01512  c000   4c 7b c0   jpcint jmp cint        Video-Chips/Editor init.
01513  c003   4c 34 cc   jdsply jmp disply      Ausgabe ac auf BS, Attribut in xr
01514  c006   4c 34 c2   jlp2   jmp lp2         Zeichen Tast.puffer nach ac
01515  c009   4c 9b c2   jloop5 jmp loop5       Zeichen von BS nach ac
01516  c00c   4c 2d c7   jprint jmp print       BS-Ausgabe ac
01517  c00f   4c 5b cc   jscorg jmp scrorg      BS-Organisation lesen
01518  c012   4c 87 fc   jkey   jmp scnfky      Tastatur-Abfrage
01519  c015   4c 51 c6   jrep   jmp repeat      Code aus Tast.-Tab. holen,
                                                REPEAT-Test
01520  c018   4c 6a cc   jplot  jmp plot        Cursor-Pos. lesen/schreiben
01521  c01b   4c 57 cd   jcurs  jmp cursor      Cursorpos. im 80-Zeichen BS setzen
01522  c01e   4c c1 c9   jesc   jmp escape      Bearbeitung der ESC-Funktionen
01523  c021   4c a2 cc   jkyset jmp keyset      Funktionstaste belegen
01524  c024   4c 94 c1   jrsirq jmp rstirq      Raster-Interrupt bearbeiten
01525  c027   4c 0c ce   jint80 jmp init80      Charg.gen. in VDC laden
01526  c02a   4c 2e cd   jswap  jmp swappr      Umschalten 40/80-Zeichen-BS
01527  c02d   4c 1b ca   jwind  jmp window      BS-Fenster definieren
01528  c030
01529  c030
01530  c030   ff ff ff           .byte $ff,$ff,$ff     Erweiterung möglich
01531  c033
01532  c033
01533  c033   ===> Tab. Anfangsadr. der BS-Zeilen lo <===
01534  c033
01535  c033   00         ldtb2  .byte <linz1    Bildschirmzeile 1
01536  c034   28                .byte <linz2    Bildschirmzeile 2
01537  c035   50                .byte <linz3    Bildschirmzeile 3
01538  c036   78                .byte <linz4    Bildschirmzeile 4
01539  c037   a0                .byte <linz5    Bildschirmzeile 5
01540  c038   c8                .byte <linz6    Bildschirmzeile 6
01541  c039   f0                .byte <linz7    Bildschirmzeile 7
01542  c03a   18                .byte <linz8    Bildschirmzeile 8
01543  c03b   40                .byte <linz9    Bildschirmzeile 9
01544  c03c   68                .byte <linz10   Bildschirmzeile 10
01545  c03d   90                .byte <linz11   Bildschirmzeile 11
```

```
01546  c03e  b8              .byte <linz12    Bildschirmzeile 12
01547  c03f  e0              .byte <linz13    Bildschirmzeile 13
01548  c040  08              .byte <linz14    Bildschirmzeile 14
01549  c041  30              .byte <linz15    Bildschirmzeile 15
01550  c042  58              .byte <linz16    Bildschirmzeile 16
01551  c043  80              .byte <linz17    Bildschirmzeile 17
01552  c044  a8              .byte <linz18    Bildschirmzeile 18
01553  c045  d0              .byte <linz19    Bildschirmzeile 19
01554  c046  f8              .byte <linz20    Bildschirmzeile 20
01555  c047  20              .byte <linz21    Bildschirmzeile 21
01556  c048  48              .byte <linz22    Bildschirmzeile 22
01557  c049  70              .byte <linz23    Bildschirmzeile 23
01558  c04a  98              .byte <linz24    Bildschirmzeile 24
01559  c04b  c0              .byte <linz25    Bildschirmzeile 25
01560  c04c
01561  c04c
01562  c04c
01563  c04c  ===> Tab. Anfangsadr. der BS-Zeilen hi <===
01564  c04c
01565  c04c  04        ldtb1 .byte >linz1     Bildschirmzeile 1
01566  c04d  04              .byte >linz2     Bildschirmzeile 2
01567  c04e  04              .byte >linz3     Bildschirmzeile 3
01568  c04f  04              .byte >linz4     Bildschirmzeile 4
01569  c050  04              .byte >linz5     Bildschirmzeile 5
01570  c051  04              .byte >linz6     Bildschirmzeile 6
01571  c052  04              .byte >linz7     Bildschirmzeile 7
01572  c053  05              .byte >linz8     Bildschirmzeile 8
01573  c054  05              .byte >linz9     Bildschirmzeile 9
01574  c055  05              .byte >linz10    Bildschirmzeile 10
01575  c056  05              .byte >linz11    Bildschirmzeile 11
01576  c057  05              .byte >linz12    Bildschirmzeile 12
01577  c058  05              .byte >linz13    Bildschirmzeile 13
01578  c059  06              .byte >linz14    Bildschirmzeile 14
01579  c05a  06              .byte >linz15    Bildschirmzeile 15
01580  c05b  06              .byte >linz16    Bildschirmzeile 16
01581  c05c  06              .byte >linz17    Bildschirmzeile 17
01582  c05d  06              .byte >linz18    Bildschirmzeile 18
01583  c05e  06              .byte >linz19    Bildschirmzeile 19
01584  c05f  06              .byte >linz20    Bildschirmzeile 20
01585  c060  07              .byte >linz21    Bildschirmzeile 21
01586  c061  07              .byte >linz22    Bildschirmzeile 22
01587  c062  07              .byte >linz23    Bildschirmzeile 23
01588  c063  07              .byte >linz24    Bildschirmzeile 24
01589  c064  07              .byte >linz25    Bildschirmzeile 25
01590  c065
01591  c065
01592  c065  ===> Tab. der Vectoren Tastaturabfragen <===
01593  c065
01594  c065  b9 c7     vectab .word contrl     CTRL-Tasten bearbeiten
01595  c067  05 c8            .word shiftd     Test auf geshiftete Steuerzeichen
01596  c069  c1 c9            .word escape     Bearbeitung der ESC-Funktionen
01597  c06b  e1 c5            .word keylog     Test auf NO SCROLL Taste
01598  c06d  ad c6            .word keyput     Bearbeitung Funktionstasten
01599  c06f
01600  c06f
```

```
01601  c06f  ===> Adreßtab. der Tast.-Decodier-Tab. <===
01602  c06f
01603  c06f  80 fa       keycod .word mode1       Tastaturtabelle ASCII normal
01604  c071  d9 fa              .word mode2       Tastaturtabelle ASCII mit Shift
01605  c073  32 fb              .word mode3       Tastaturtabelle ASCII mit CBM-taste
01606  c075  8b fb              .word mode4       Tastaturtabelle ASCII mit CTRL-taste
01607  c077  80 fa              .word mode1       Tast.tab. ALT (Ersatz ASCII norm.)
01608  c079  e4 fb              .word mode5       Tastaturtabelle ASCII CAPS LOCK
01609  c07b
01610  c07b
01611  c07b  ===> Video-Chips/Editor initialisieren <===
01612  c07b
01613  c07b  a9 03       cint    lda #%00000011
01614  c07d
01615  c07d   xxxxxx11  Adressbits 15/14 VIC BS-RAM
01616  c07d
01617  c07d  0d 00 dd            ora d2pra         CIA2 Datenport A
01618  c080  8d 00 dd            sta d2pra         Adresse VIC BS-RAM einstellen
01619  c083
01620  c083  a9 fb               lda #%11111011    Zeichengenerator einblenden
01621  c085
01622  c085   xxxxx0xx  VIC Chargen.ROM ein
01623  c085
01624  c085  25 01               and r8502         8502 CPU-Port Datenregister
01625  c087  09 02               ora #%00000010    Kernal-ROM einblenden
01626  c089
01627  c089   xxxxxx1x  VIC KERNAL ROM $e000-$ffff ein
01628  c089
01629  c089  85 01               sta r8502         8502 CPU-Port Datenregister
01630  c08b
01631  c08b  20 cc ff            jsr kclrch        Ein-/Ausgabe-Kanäle üb. Vector
                                                   schließen (Sprungv.)
01632  c08e  a9 00               lda #0            Voreinstellungen vornehmen:
01633  c090
01634  c090  20 80 fc            jsr patch         DIN-Modus mögl., Lautst. SID 6581 =
                                                   0
01635  c093  85 d8               sta graphm        Textmodus (graph 0)
01636  c095  85 d7               sta mode          BS-Modus 40 Zeichen
01637  c097  85 d0               sta ndx           Tastaturpuffer leer
01638  c099  85 d1               sta kyndx         Fkt.tasten-Puffer leer
01639  c09b  85 d6               sta crsw          cr-Taste nicht gedrückt
01640  c09d  8d 21 0a            sta pause         CTRL s Modus aus
01641  c0a0  8d 26 0a            sta blnon         Cursor Blinken ein
01642  c0a3  85 d9               sta charen        RAM/ROM-Flag für VIC
01643  c0a5  8d 2e 0a            sta vm3           VDC Basisadresse BS-RAM
01644  c0a8
01645  c0a8  a9 14               lda #%00010100
01646  c0aa  8d 2c 0a            sta vm1           VIC Text-Basiszeiger
01647  c0ad
01648  c0ad  a9 78               lda #%01111000
01649  c0af  8d 2d 0a            sta vm2           VIC Bit-Map Basiszeiger
01650  c0b2
01651  c0b2  a9 08               lda #>vdccol      Basisadresse VDC Attribut-RAM
01652  c0b4  8d 2f 0a            sta vm4           VDC Basisadresse Attribut-RAM
01653  c0b7
01654  c0b7  ad 4c c0            lda ldtb1         Tab. Anfangsadr. der BS-Zeilen hi
01655  c0ba  8d 3b 0a            sta ldtbsa        VIC Basisadresse BS-RAM
01656  c0bd
```

```
01657  c0bd  a9 0a           lda #10
01658  c0bf  8d 20 0a        sta xmax        max. Länge Tast.puffer
01659  c0c2  8d 28 0a        sta curcnt      VIC Zähler für Cursor-Blinken
01660  c0c5  8d 27 0a        sta blnsw       VIC Flag für Cursor-Blinken
01661  c0c8  8d 24 0a        sta delay       Zeit bis zum Einsetzen des REPEAT
01662  c0cb
01663  c0cb  a9 04           lda #4
01664  c0cd  8d 23 0a        sta kount       Zähler für REPEAT-Geschwindigkeit
01665  c0d0
01666  c0d0  20 83 c9        jsr stdeft      Vorgabe-Tab.marken setzen
01667  c0d3  8d 22 0a        sta rptflg      REPEAT für alle Tasten ($80)
01668  c0d6
01669  c0d6  0d 05 d5        ora mmumcr      MMU 8722 Modus Config. Reg.
01670  c0d9
01671  c0d9   1xxxxxxx  40-Zeichen BS
01672  c0d9
01673  c0d9  8d 05 d5        sta mmumcr      MMU 8722 Modus Config. Reg.
01674  c0dc
01675  c0dc  a9 60           lda #%01100000
01676  c0de
01677  c0de   x11xxxxx  Cursor blinkt langsam (1/32 Bildwiederholfreq.)
01678  c0de   xxx00000  Block-Cursor
01679  c0de
01680  c0de  8d 2b 0a        sta curmod      VDC Cursormodus
01681  c0e1
01682  c0e1  a9 d0           lda #208
01683  c0e3  8d 34 0a        sta split       lo-Byte Rastervorgabe für IRQ
01684  c0e6
01685  c0e6  ===> Editor-Variablen 40-Zeichen Bildschirm in Zeropage <===
01686  c0e6
01687  c0e6  a2 1a           ldx #26
01688  c0e8  bd 74 ce  cint01 lda loczp,x    Editor-Vorgaben für 40-Zeichen BS
01689  c0eb  95 e0           sta pnt,x       addrl Zeiger auf akt. BS-Zeile
01690  c0ed
01691  c0ed  ===> Editor-Variablen 80-Zeichen Bildschirm in Rettbereich <===
01692  c0ed
01693  c0ed  bd 8e ce        lda locabs,x    Editor-Vorgaben für 80-Zeichen-BS
01694  c0f0  9d 40 0a        sta swpout,x    Swap-Bereich Editor-Werte
01695  c0f3
01696  c0f3  ca              dex
01697  c0f4  10 f2           bpl cint01
01698  c0f6
01699  c0f6  ===> Tab. Vectoren Tastasturabfrage in RAM kopieren <===
01700  c0f6
01701  c0f6  a2 09           ldx #9
01702  c0f8  bd 65 c0  cint02 lda vectab,x   Tab. der Vectoren Tastaturabfragen
01703  c0fb  9d 34 03        sta ctlvec,x    ins RAM kopieren
01704  c0fe  ca              dex
01705  c0ff  10 f7           bpl cint02
01706  c101
01707  c101  2c 04 0a        bit initst      Flag Status nach RESET, BASIC-IRQ
01708  c104  70 1e           bvs cint05      Tastaturtab./ Fkt.Tasten
                                             initialisiert
01709  c106
01710  c106  ===> akt. Adressen Tastatur-Dekodiertabellen initialisieren <===
01711  c106
01712  c106  a2 0b           ldx #11
01713  c108  bd 6f c0  cint03 lda keycod,x    Adreßtab. der Tast.-Decodier-Tab.
```

```
01714  c10b  9d 3e 03          sta nrmtab,x     addrl akt. Tast.tab. normal
01715  c10e  ca                dex
01716  c10f  10 f7             bpl cint03
01717  c111
01718  c111  ===> Funktionstasten initialisieren <===
01719  c111
01720  c111  a2 4c             ldx #76
01721  c113  bd a8 ce   cint04 lda pkytbl,x     Längen der Standard-Texte der
                                               Funktionstasten
01722  c116  9d 00 10          sta pkybuf,x     Längentab. Fkt.tasten-Texte
01723  c119  ca                dex
01724  c11a  10 f7             bpl cint04
01725  c11c
01726  c11c  a9 40             lda #%01000000   bit6: 1=Tast.tab./Fkt.tasten init.
01727  c11e  0d 04 0a          ora initst       Flag Status nach RESET, BASIC-IRQ
01728  c121  8d 04 0a          sta initst       Flag Status nach RESET, BASIC-IRQ
01729  c124
01730  c124  ===> 80-Zeichen Bildschirm löschen / Tabulatormarken setzen <===
01731  c124
01732  c124  20 2e cd   cint05 jsr swappr       Umschalten 40/80-Zeichen-BS
01733  c127  20 83 c9          jsr stdeft       Standard-Tabulatoren setzen (ESC y)
01734  c12a  20 24 ca          jsr sreset       BS-Fenster auf volle Größe setzen
01735  c12d  20 42 c1          jsr clsr         BS löschen
01736  c130  20 2e cd          jsr swappr       Umschalten 40/80-Zeichen-BS
01737  c133
01738  c133  ===> 40-Zeichen Bildschirm löschen <===
01739  c133
01740  c133  20 24 ca          jsr sreset       BS-Fenster auf volle Größe setzen
01741  c136  20 42 c1          jsr clsr         BS löschen
01742  c139
01743  c139  2c 05 d5          bit mmumcr       MMU 8722 Modus Config. Reg.
01744  c13c  30 03             bmi cint06       40-Zeichen Bildschirm
01745  c13e
01746  c13e  20 2e cd          jsr swappr       Umschalten 40/80-Zeichen-BS
01747  c141  60         cint06 rts
01748  c142
01749  c142
01750  c142  ===> BS löschen <===
01751  c142
01752  c142  20 50 c1   clsr   jsr home         Cursor auf HOME-Pos.
01753  c145  20 5e c1   cls10  jsr scrset       pnt auf Adresse BS-Zeile xr
01754  c148  20 a5 c4          jsr clrln        akt. BS-Zeile als Normalzeile
                                               eintragen und löschen
01755  c14b
01756  c14b  e4 e4             cpx scbot        BS-Fenster: unterste Zeile
01757  c14d  e8                inx
01758  c14e  90 f5             bcc cls10
01759  c150
01760  c150
01761  c150  ===> Cursor auf HOME-Pos. <===
01762  c150
01763  c150  a6 e5      home   ldx sctop        BS-Fenster: oberste Zeile
01764  c152  86 eb             stx tblx         akt. Cursor-Zeile
01765  c154  86 e8             stx lsxp         Cursor-Zeile vor input
01766  c156
01767  c156
```

```
01768  c156  ===> Cursor an linken Rand BS-Zeile <===
01769  c156
01770  c156  a4 e6      stu10  ldy sclf         BS-Fenster: Spalte linker Rand
01771  c158  84 ec             sty pntr         akt. Cursor-Spalte
01772  c15a  84 e9             sty lstp         Cursor-Spalte vor input
01773  c15c
01774  c15c
01775  c15c  ===> pnt auf Adresse akt. BS-Zeile (Cursor) <===
01776  c15c
01777  c15c  a6 eb      stupt  ldx tblx         akt. Cursor-Zeile
01778  c15e
01779  c15e
01780  c15e  ===> pnt auf Adresse BS-Zeile xr <===
01781  c15e
01782  c15e  bd 33 c0   scrset lda ldtb2,x      Tab. Anfangsadr. der BS-Zeilen lo
01783  c161
01784  c161  24 d7             bit mode         BS-Modus 40/80-Zeichen
01785  c163  10 01             bpl scrst1
01786  c165
01787  c165  0a                asl a            Verdoppeln für 80-Zeichen BS
01788  c166  85 e0      scrst1 sta pnt          addrl Zeiger auf akt. BS-Zeile
01789  c168
01790  c168  bd 4c c0          lda ldtb1,x      Tab. Anfangsadr. der BS-Zeilen hi
01791  c16b  29 03             and #%00000011
01792  c16d
01793  c16d  24 d7             bit mode         BS-Modus 40/80-Zeichen
01794  c16f  10 06             bpl scrst2
01795  c171
01796  c171  2a                rol a            Verdoppeln mit Übertrag für
                                                 80-Zeichen BS
01797  c172  0d 2e 0a          ora vm3          VDC Basisadresse BS-RAM
01798  c175  90 03             bcc scrst3       immer
01799  c177
01800  c177
01801  c177  0d 3b 0a   scrst2 ora ldtbsa       VIC Basisadresse BS-RAM
01802  c17a  85 e1      scrst3 sta pnt+1        addrh Zeiger auf akt. BS-Zeile
01803  c17c
01804  c17c
01805  c17c  ===> Zeiger in Farb-RAM nachführen <===
01806  c17c
01807  c17c  a5 e0      scolor lda pnt          addrl Zeiger auf akt. BS-Zeile
01808  c17e  85 e2             sta user         addrl Zeiger auf akt. Attributzeile
01809  c180
01810  c180  a5 e1             lda pnt+1        addrh Zeiger auf akt. BS-Zeile
01811  c182  24 d7             bit mode         BS-Modus 40/80-Zeichen
01812  c184  10 07             bpl scrst5
01813  c186
01814  c186  29 07             and #%00000111
01815  c188  0d 2f 0a          ora vm4          VDC Basisadresse Attribut-RAM
01816  c18b  d0 04             bne scrst6
01817  c18d
01818  c18d  29 03      scrst5 and #%00000011
01819  c18f  09 d8             ora #>colram     VIC Farb-RAM
01820  c191  85 e3      scrst6 sta user+1       addrh Zeiger auf akt. Attributzeile
01821  c193  60                rts
01822  c194
01823  c194
```

```
01824  c194  ===> Raster-Interrupt bearbeiten <===
01825  c194
01826  c194  38         rstirq sec
01827  c195
01828  c195  ad 19 d0          lda vicifr       VIC 8564 Interrupt Flag-Register
01829  c198  29 01             and #%00000001   Raster-IRQ Bit testen
01830  c19a  f0 07             beq rstir1       nicht gesetzt, rts
01831  c19c
01832  c19c  ===> Raster-IRQ bearbeiten <===
01833  c19c
01834  c19c  8d 19 d0          sta vicifr       Raster-IRQ Bit löschen
01835  c19f
01836  c19f  a5 d8             lda graphm       Flag Text/Graphik/Split-BS
01837  c1a1  c9 ff             cmp #$ff
01838  c1a3  f0 6f       rstir1 beq irqrts
01839  c1a5
01840  c1a5  2c 11 d0          bit viccr1       Bit7 (=Bit8 Rasterwert-Vorgabe)
                                                testen
01841  c1a8  30 04             bmi rstir2       gesetzt
01842  c1aa
01843  c1aa  29 40             and #%01000000   Bit6 graphm testen (1=Split BS)
01844  c1ac  d0 31             bne text         gesetzt
01845  c1ae
01846  c1ae  38         rstir2 sec
01847  c1af
01848  c1af  a5 d8             lda graphm       Flag Text/Graphik/Split-BS
01849  c1b1  f0 2c             beq text         mit carry = 1
01850  c1b3
01851  c1b3  24 d8             bit graphm       Flag Text/Graphik/Split-BS
01852  c1b5  50 06             bvc rstir3       Bit6 = 0: nicht Split BS
01853  c1b7
01854  c1b7  ad 34 0a          lda split        VIC Rasterwert geteilter BS
01855  c1ba  8d 12 d0          sta vicrwr       VIC 8564 read/write Raster
01856  c1bd
01857  c1bd
01858  c1bd  ===> Parameter für Graphik vorbesetzen <===
01859  c1bd
01860  c1bd  a5 01       rstir3 lda r8502        8502 CPU-Port Datenregister
01861  c1bf  29 fd             and #%11111101   HI-RES-RAM einblenden
01862  c1c1  09 04             ora #%00000100   I/O-Chips ein-, Chargen ausblenden
01863  c1c3  48                pha
01864  c1c4
01865  c1c4  ad 2d 0a          lda vm2          VIC Bit-Map Basiszeiger
01866  c1c7  48                pha
01867  c1c8
01868  c1c8  ad 11 d0          lda viccr1       VIC 8564 Control-Register 1
01869  c1cb  29 7f             and #%01111111   MSB Raster-Register rücksetzen
01870  c1cd  09 20             ora #%00100000   Bit-Map-Mode einschalten
01871  c1cf  a8                tay
01872  c1d0
01873  c1d0  ad 16 d0          lda viccr2       VIC 8564 Control-Register 2
01874  c1d3
01875  c1d3  24 d8             bit graphm       Flag Text/Graphik/Split-BS
01876  c1d5  30 03             bmi rstir4       Bit7 = 1: Multi-Color-Modus
                                                eingeschaltet
01877  c1d7
01878  c1d7  29 ef             and #%11101111   Bit4 rücks. (Multi-Color-Modus aus)
01879  c1d9  2c                .byte skip2
```

```
01880  c1da
01881  c1da
01882  c1da  ===> Multi-Color-Modus <===
01883  c1da
01884  c1da  09 10       rstir4 ora #%00010000   Bit4 setzen (Multi-Color-Modus ein)
01885  c1dc  aa                 tax
01886  c1dd  d0 28              bne doit          immer
01887  c1df
01888  c1df
01889  c1df  ===> Parameter für Text vorbesetzen <===
01890  c1df
01891  c1df  a9 ff       text   lda #%11111111
01892  c1e1  8d 12 d0           sta vicrwr        VIC 8564 read/write Raster
01893  c1e4
01894  c1e4  a5 01              lda r8502         8502 CPU-Port Datenregister
01895  c1e6  09 02              ora #%00000010    Kernal-ROM einblenden
01896  c1e8  29 fb              and #%11111011    Chargen einblenden
01897  c1ea  05 d9              ora charen        RAM/ROM-Flag für VIC
01898  c1ec  48                 pha
01899  c1ed
01900  c1ed  ad 2c 0a           lda vm1           VIC Text-Basiszeiger
01901  c1f0  48                 pha
01902  c1f1
01903  c1f1  ad 11 d0           lda viccr1        VIC 8564 Control-Register 1
01904  c1f4  29 5f              and #%01011111
01905  c1f6
01906  c1f6  Bit5: 0=Bit-Map-Modus ausschalten, Bit7: 0=MSB Raster
01907  c1f6
01908  c1f6  a8                 tay
01909  c1f7
01910  c1f7  ad 16 d0           lda viccr2        VIC 8564 Control-Register 2
01911  c1fa  29 ef              and #%11101111    Bit4: 0=Multi-Color-Modus aus
01912  c1fc  aa                 tax
01913  c1fd  b0 08              bcs doit          bereitgestellte Parameter in
                                                  VIC-Register
01914  c1ff
01915  c1ff  ===> Warteschleife 0.03 ms <===
01916  c1ff
01917  c1ff  a2 07              ldx #7
01918  c201  ca           text1 dex
01919  c202  d0 fd              bne text1
01920  c204  ea                 nop
01921  c205  ea                 nop
01922  c206  aa                 tax
01923  c207
01924  c207
01925  c207  ===> bereitgestellte Parameter in VIC-Register <===
01926  c207
01927  c207  68           doit  pla
01928  c208  8d 18 d0           sta vicmcr        VIC 8564 Memory-Control Register
01929  c20b
01930  c20b  68                 pla
01931  c20c  85 01              sta r8502         8502 CPU-Port Datenregister
01932  c20e
01933  c20e  8c 11 d0           sty viccr1        VIC 8564 Control-Register 1
01934  c211  8e 16 d0           stx viccr2        VIC 8564 Control-Register 2
01935  c214
01936  c214  b0 13       irqrts bcs irqrt1        Text-Modus
```

```
01937  c216
01938  c216  ad 30 d0          lda vicspd       VIC 8564 Speed-Register
01939  c219  29 01             and #%00000001   CPU-Takt maskieren
01940  c21b  f0 0c             beq irqrt1       1 MHz
01941  c21d
01942  c21d  a5 d8             lda graphm       Flag Text/Graphik/Split-BS
01943  c21f  29 40             and #%01000000   Bit6: 1 = Split BS
01944  c221  f0 06             beq irqrt1       nicht Split BS
01945  c223
01946  c223  ad 11 d0          lda viccr1       Bit7: = Bit8 Rasterwert
01947  c226  10 01             bpl irqrt1       nicht gesetzt
01948  c228
01949  c228  38                sec
01950  c229  58          irqrt1 cli
01951  c22a  90 07             bcc irqrt2
01952  c22c
01953  c22c  20 87 fc          jsr scnfky       Tastatur-Abfrage (foreign key)
01954  c22f  20 e7 c6          jsr blink        Cursor ein/aus 40-Zeichen BS
01955  c232  38                sec
01956  c233  60          irqrt2 rts
01957  c234
01958  c234
01959  c234                    .end
01960  c234                    .lib c128.screen2
01961  c234
01963  c234
01964  c234
01965  c234  ===> Zeichen aus Fkt.- /Tast.puffer nach ac <===
01966  c234
01967  c234  a6 d1       lp2   ldx kyndx        Bytes im Fkt.tasten-Puffer
01968  c236  f0 0c             beq lp3          Zeichen aus Tastaturpuffer nach ac
01969  c238
01970  c238  a4 d2             ldy keyidx       Zeiger in Text akt. Fkt.taste
01971  c23a  b9 0a 10          lda pkydef,y     Puffer für Fkt.tasten-Befehle
01972  c23d
01973  c23d  c6 d1             dec kyndx        Bytes im Fkt.tasten-Puffer
01974  c23f  e6 d2             inc keyidx       Zeiger in Text akt. Fkt.taste
01975  c241
01976  c241  58                cli
01977  c242  18                clc
01978  c243  60                rts
01979  c244
01980  c244
01981  c244  ===> Zeichen aus Tastaturpuffer nach ac <===
01982  c244
01983  c244  ac 4a 03    lp3   ldy keyd         Tast.puffer
01984  c247  bd 4b 03    lp1   lda keyd+1,x     Tast.puffer + 1
01985  c24a  9d 4a 03          sta keyd,x       Tast.puffer
01986  c24d
01987  c24d  e8                inx
01988  c24e  e4 d0             cpx ndx          Bytes im Tast.puffer
01989  c250  d0 f5             bne lp1
01990  c252
01991  c252  c6 d0             dec ndx          Bytes im Tast.puffer
01992  c254  98                tya
01993  c255  58                cli
01994  c256  18                clc
01995  c257  60                rts
```

```
01996  c258
01997  c258
01998  c258  ===> BS-Ausgabe ac, Eingabe von Tastatur bis CR <===
01999  c258
02000  c258  20 2d c7   loop4  jsr print       BS-Ausgabe ac
02001  c25b
02002  c25b
02003  c25b  ===> Warten auf Eingabe von Tastatur, nach CR Zeiger setzen <===
02004  c25b
02005  c25b  20 6f cd   waitky jsr crsron      Cursor ein
02006  c25e  a5 d0      loop3a lda ndx         Bytes im Tast.puffer
02007  c260  05 d1             ora kyndx       Bytes im Fkt.tasten-Puffer
02008  c262  f0 fa             beq loop3a
02009  c264
02010  c264  20 9f cd          jsr crsrof      Cursor aus
02011  c267  20 34 c2          jsr lp2         Zeichen aus Fkt.- /Tast.puffer nach
                                                ac
02012  c26a
02013  c26a  c9 0d             cmp #cr
02014  c26c  d0 ea             bne loop4       BS-Ausgabe ac, Eingabe von Tastatur
                                                bis CR
02015  c26e
02016  c26e  85 d6             sta crsw        Flag Warten Tast.eingabe
02017  c270
02018  c270  a9 00             lda #0
02019  c272  85 f4             sta qtsw        Flag für QUOTE-Modus
02020  c274
02021  c274  20 c3 cb          jsr fndend      Cursor auf Ende (Doppel-) Zeile
02022  c277  8e 30 0a          stx lintmp      Puffer für BS-Zeilennummer
02023  c27a
02024  c27a  20 b5 cb          jsr fistrt      pnt auf Anfang (Doppel-) Zeile
02025  c27d
02026  c27d  ===> Test auf Zulässigkeit der Parameter vor input <===
02027  c27d
02028  c27d  a4 e6             ldy sclf        BS-Fenster: Spalte linker Rand
02029  c27f  a5 e8             lda lsxp        Cursor-Zeile vor input
02030  c281  30 13             bmi lp80        Zeile/Spalte vor input ignorieren
02031  c283
02032  c283  c5 eb             cmp tblx        akt. Cursor-Zeile
02033  c285  90 0f             bcc lp80        Zeile/Spalte vor input ignorieren
02034  c287
02035  c287  a4 e9             ldy lstp        Cursor-Spalte vor input
02036  c289  cd 30 0a          cmp lintmp      Puffer für BS-Zeilennummer
02037  c28c  d0 04             bne lp70
02038  c28e
02039  c28e  c4 ea             cpy indx        Spalte letztes Zeichen in Zeile
02040  c290  f0 02             beq lp75        ac=Eingabezeile, yr=Eingabespalte,
                                                Zeichen von BS
02041  c292
02042  c292  b0 11      lp70   bcs clp2
02043  c294
02044  c294
02045  c294  ===> ac=Eingabezeile, yr=Eingabespalte, Zeichen von BS <===
02046  c294
02047  c294  85 eb      lp75   sta tblx        akt. Cursor-Zeile
02048  c296
02049  c296
```

```
02050  c296  ===> yr=Eingabespalte, Zeichen von BS, Wandlung BSC/ASC <===
02051  c296
02052  c296  84 ec       lp80   sty pntr          akt. Cursor-Spalte
02053  c298
02054  c298  4c bc c2           jmp lop5          akt. Zeichen aus BS-RAM,
                                                  Codewandlung BSC/ASC
02055  c29b
02056  c29b
02057  c29b  ===> Zeichen von BS nach ac <===
02058  c29b
02059  c29b  98          loop5  tya
02060  c29c  48                 pha
02061  c29d
02062  c29d  8a                 txa
02063  c29e  48                 pha
02064  c29f
02065  c29f  a5 d6              lda crsw          Flag Warten Tast.eingabe
02066  c2a1  f0 b8              beq waitky        Warten auf Eingabe von Tastatur,
                                                  nach CR Zeiger setzen
02067  c2a3  10 17              bpl lop5          akt. Zeichen aus BS-RAM,
                                                  Codewandlung BSC/ASC
02068  c2a5
02069  c2a5  a9 00       clp2   lda #0
02070  c2a7  85 d6              sta crsw          Flag Warten Tast.eingabe
02071  c2a9
02072  c2a9  a9 0d              lda #cr
02073  c2ab
02074  c2ab  a2 03              ldx #3
02075  c2ad  e4 99              cpx dfltn         Vorgabe Eingabe-Gerätenummer (0)
02076  c2af  f0 04              beq clp21
02077  c2b1
02078  c2b1  e4 9a              cpx dflto         Vorgabe Ausgabe-Gerätenummer (3)
02079  c2b3  f0 03              beq clp22
02080  c2b5
02081  c2b5  20 2d c7    clp21  jsr print         BS-Ausgabe ac
02082  c2b8
02083  c2b8  a9 0d       clp22  lda #cr
02084  c2ba  d0 39              bne clp7
02085  c2bc
02086  c2bc
02087  c2bc  ===> akt. Zeichen aus BS-RAM, Codewandlung BSC/ASC <===
02088  c2bc
02089  c2bc  20 5c c1    lop5   jsr stupt         pnt auf Adresse akt. BS-Zeile
                                                  (Cursor)
02090  c2bf  20 58 cb           jsr get1ch        Zeichen + Attribut aus BS holen
02091  c2c2  85 ef              sta datax         akt. Ausgabezeichen
02092  c2c4
02093  c2c4  29 3f              and #%00111111
02094  c2c6  06 ef              asl datax         akt. Ausgabezeichen
02095  c2c8
02096  c2c8  24 ef              bit datax         akt. Ausgabezeichen
02097  c2ca  10 02              bpl lop54
02098  c2cc
02099  c2cc  09 80              ora #%10000000
02100  c2ce  90 04       lop54  bcc lop52
02101  c2d0
02102  c2d0  a6 f4              ldx qtsw          Flag für QUOTE-Modus
02103  c2d2  d0 04              bne lop53
```

```
02104  c2d4
02105  c2d4  70 02      lop52  bvs lop53
02106  c2d6
02107  c2d6  09 40             ora #%01000000
02108  c2d8  20 ff c2   lop53  jsr qtswc       QUOTE-Modus Flag '"' invertieren
02109  c2db
02110  c2db  a4 eb             ldy tblx        akt. Cursor-Zeile
02111  c2dd  cc 30 0a          cpy lintmp      Puffer für BS-Zeilennummer
02112  c2e0  90 0a             bcc clp00
02113  c2e2
02114  c2e2  a4 ec             ldy pntr        akt. Cursor-Spalte
02115  c2e4  c4 ea             cpy indx        Spalte letztes Zeichen in Zeile
02116  c2e6  90 04             bcc clp00
02117  c2e8
02118  c2e8  66 d6             ror crsw        Flag Warten Tast.eingabe
02119  c2ea
02120  c2ea  Bit7 setzen, Ausführung Carriage Return veranlassen
02121  c2ea
02122  c2ea  30 03             bmi clp1        immer
02123  c2ec
02124  c2ec
02125  c2ec  20 ed cb   clp00  jsr nxtchr      Cursor rechts, carry=1 neue Zeile
02126  c2ef  c9 de      clp1   cmp #pi         ASC für PI
02127  c2f1  d0 02             bne clp7
02128  c2f3
02129  c2f3  a9 ff             lda #pitk       BASIC-Token für PI
02130  c2f5  85 ef      clp7   sta datax       akt. Ausgabezeichen
02131  c2f7
02132  c2f7  68                pla
02133  c2f8  aa                tax
02134  c2f9
02135  c2f9  68                pla
02136  c2fa  a8                tay
02137  c2fb
02138  c2fb  a5 ef             lda datax       akt. Ausgabezeichen
02139  c2fd  18                clc
02140  c2fe  60                rts
02141  c2ff
02142  c2ff
02143  c2ff  ===> QUOTE-Modus Flag '"' invertieren <===
02144  c2ff
02145  c2ff  c9 22      qtswc  cmp #'"'
02146  c301  d0 08             bne qtswl
02147  c303
02148  c303  a5 f4             lda qtsw        Flag für QUOTE-Modus
02149  c305  49 01             eor #%00000001  Bit flippen
02150  c307  85 f4             sta qtsw        Flag für QUOTE-Modus
02151  c309
02152  c309  a9 22             lda #'"'
02153  c30b  60        qtswl  rts
02154  c30c
02155  c30c
02156  c30c  ===> Ende Zeichenausgabe <===
02157  c30c
02158  c30c  a5 ef      loop2  lda datax       akt. Ausgabezeichen
02159  c30e  85 f0             sta lstchr      letztes Ausgabezeichen
02160  c310  20 57 cd          jsr cursor      Cursorpos. im 80-Zeichen BS setzen
02161  c313
```

```
02162  c313  a5 f5           lda insrt       Zähler für einzufügende Zeichen
02163  c315  f0 02           beq lop2
02164  c317
02165  c317  46 f4           lsr qtsw        Flag für QUOTE-Modus
02166  c319
02167  c319  68        lop2   pla
02168  c31a  a8              tay
02169  c31b
02170  c31b  68              pla
02171  c31c  aa              tax
02172  c31d
02173  c31d  68              pla
02174  c31e  18              clc
02175  c31f  60              rts
02176  c320
02177  c320
02178  c320  ===> Ausgabe shift-Zeichen (ac) auf BS <===
02179  c320
02180  c320  09 40     nxt33  ora #%01000000
02181  c322
02182  c322  a6 f3     nxt3   ldx rvs          Flag für REVERS-Modus
02183  c324  f0 02            beq nvs          Test auf INSERT-Modus, Zeichen
                                                zählen
02184  c326
02185  c326
02186  c326  ===> REVERS-Bit setzen <===
02187  c326
02188  c326  09 80     nc3    ora #%10000000
02189  c328
02190  c328
02191  c328  ===> Test auf INSERT-Modus, Zeichen zählen <===
02192  c328
02193  c328  a6 f5     nvs    ldx insrt       Zähler für einzufügende Zeichen
02194  c32a  f0 02            beq nvsa         Test auf AUTO-INSERT-Modus
02195  c32c
02196  c32c  c6 f5            dec insrt       Zähler für einzufügende Zeichen
02197  c32e
02198  c32e
02199  c32e  ===> Test auf AUTO-INSERT-Modus <===
02200  c32e
02201  c32e  24 f6     nvsa   bit insflg      AUTO-INSERT-Flag
02202  c330  10 09            bpl nvs2
02203  c332
02204  c332  48               pha
02205  c333  20 e3 c8         jsr insert      Ausführen 'INSERT'
02206  c336
02207  c336  a2 00            ldx #0
02208  c338  86 f5            stx insrt       Zähler für einzufügende Zeichen
02209  c33a
02210  c33a  68               pla
02211  c33b
02212  c33b  20 2f cc  nvs2   jsr displc      BS-Ausgabe ac mit akt.
                                               Farbe/Attribut
02213  c33e
02214  c33e
02215  c33e  ===> Test auf Einfüge-/Doppelzeile/Scroll <===
02216  c33e
02217  c33e  c4 e7     movchr cpy scrt        BS-Fenster: Spalte rechter Rand
```

```
02218  c340  90 0a            bcc movc10      Bildschirm Scroll
02219  c342
02220  c342  a6 eb            ldx tblx        akt. Cursor-Zeile
02221  c344  e4 e4            cpx scbot       BS-Fenster: unterste Zeile
02222  c346  90 04            bcc movc10      Bildschirm Scroll
02223  c348
02224  c348  24 f8            bit scroll      Bit7: 1=Bildschirm-Scroll aus
02225  c34a  30 16            bmi movc30      kein Bildschirm-Scroll
02226  c34c
02227  c34c
02228  c34c  ===> Bildschirm Scroll <===
02229  c34c
02230  c34c  20 5c c1   movc10 jsr stupt       pnt auf Adresse akt. BS-Zeile
                                               (Cursor)
02231  c34f  20 ed cb          jsr nxtchr      Cursor rechts, carry=1 neue Zeile
02232  c352  90 0e            bcc movc30      carry=0/1: Platz/kein Platz für
                                               weiteres Zeichen
02233  c354
02234  c354  20 74 cb          jsr getbit      Test akt. Zeile (Doppelz: carry=1)
02235  c357  b0 08            bcs movc20      carry=0: Platz für weiteres Zeichen
02236  c359
02237  c359  38                sec
02238  c35a  24 f8            bit scroll      Bit6: 1=Doppelzeilen aus
02239  c35c  70 04            bvs movc30      keine Doppelzeile erlaubt
02240  c35e
02241  c35e  ===> Bildschirm scrollen für Doppelzeile <===
02242  c35e
02243  c35e  20 7c c3          jsr scrdwn      nach unten scrollen
02244  c361
02245  c361
02246  c361
02247  c361  ===> carry=0: Platz für weiteres Zeichen <===
02248  c361
02249  c361  18         movc20 clc
02250  c362
02251  c362
02252  c362  ===> carry=0/1: Platz/kein Platz für weiteres Zeichen <===
02253  c362
02254  c362  60         movc30 rts
02255  c363
02256  c363
02257  c363  ===> Cursor in nächste Zeile/Test/Scroll <===
02258  c363
02259  c363  a6 eb      nxln   ldx tblx        akt. Cursor-Zeile
02260  c365  e4 e4             cpx scbot       BS-Fenster: unterste Zeile
02261  c367  90 0e             bcc nxln1       Cursor in nächste BS-Zeile
02262  c369
02263  c369  24 f8             bit scroll      Bit7: 1=Bildschirm-Scroll aus
02264  c36b  10 06             bpl doscrl      BS-Fenster nach oben scrollen
02265  c36d
02266  c36d  ===> Cursor von unterster BS-Zeile in oberste BS-Zeile <===
02267  c36d
02268  c36d  a5 e5             lda sctop       BS-Fenster: oberste Zeile
02269  c36f  85 eb             sta tblx        akt. Cursor-Zeile
02270  c371  b0 06             bcs nowhop      immer
02271  c373
02272  c373
```

```
02273  c373  ===> BS-Fenster nach oben scrollen <===
02274  c373
02275  c373  20 a6 c3   doscrl jsr scrup        nach oben scrollen
02276  c376  18                clc
02277  c377
02278  c377
02279  c377  ===> Cursor in nächste BS-Zeile <===
02280  c377
02281  c377  e6 eb      nxln1  inc tblx         akt. Cursor-Zeile
02282  c379  4c 5c c1   nowhop jmp stupt        pnt auf Adresse akt. BS-Zeile
                                                (Cursor)
02283  c37c
02284  c37c
02285  c37c  ===> nach unten scrollen <===
02286  c37c
02287  c37c  a6 e8      scrdwn ldx lsxp         Cursor-Zeile vor input
02288  c37e  30 06             bmi scd30       Cursor-Zeile vor input ungültig
02289  c380
02290  c380  e4 eb             cpx tblx        akt. Cursor-Zeile
02291  c382  90 02             bcc scd30
02292  c384
02293  c384  e6 e8             inc lsxp        Cursor-Zeile vor input nachführen
02294  c386  a6 e4      scd30  ldx scbot        BS-Fenster: unterste Zeile
02295  c388  20 5e c1   scd10  jsr scrset       pnt auf Adresse BS-Zeile xr
02296  c38b
02297  c38b  a4 e6             ldy sclf        BS-Fenster: Spalte linker Rand
02298  c38d  e4 eb             cpx tblx        akt. Cursor-Zeile
02299  c38f  f0 0f             beq scd20
02300  c391
02301  c391
02302  c391  ===> Doppelzeile nachführen <===
02303  c391
02304  c391  ca                dex
02305  c392  20 76 cb          jsr getbt1      Test Zeile xr (Doppelz.: carry=1)
02306  c395  e8                inx
02307  c396  20 83 cb          jsr putbt1      Zeile in xr als Doppel-/Normalzeile
                                                eintragen
02308  c399
02309  c399  ca                dex
02310  c39a  20 0d c4          jsr movlin      Zeilen im BS- und Attribut-RAM
                                                verschieben
02311  c39d  4c 88 c3          jmp scd10
02312  c3a0
02313  c3a0
02314  c3a0  20 a5 c4   scd20  jsr clrln       akt. BS-Zeile als Normalzeile
                                                eintragen und löschen
02315  c3a3  4c 93 cb          jmp setbit      Zeile als Doppelzeile eintragen
02316  c3a6
02317  c3a6
02318  c3a6  ===> nach oben scrollen <===
02319  c3a6
02320  c3a6  a6 e5      scrup  ldx sctop        BS-Fenster: oberste Zeile
02321  c3a8  e8         scru00 inx
02322  c3a9  20 76 cb          jsr getbt1      Test Zeile xr (Doppelz.: carry=1)
02323  c3ac  90 0a             bcc scru15
02324  c3ae
02325  c3ae  e4 e4             cpx scbot       BS-Fenster: unterste Zeile
02326  c3b0  90 f6             bcc scru00
```

```
02327  c3b2
02328  c3b2  a6 e5           ldx sctop       BS-Fenster: oberste Zeile
02329  c3b4  e8              inx
02330  c3b5  20 85 cb        jsr clrbit      Zeile als Normalzeile eintragen
02331  c3b8  c6 eb     scru15 dec tblx        akt. Cursor-Zeile
02332  c3ba
02333  c3ba  24 e8           bit lsxp        Cursor-Zeile vor input
02334  c3bc  30 02           bmi scru20      Cursor-Zeile vor input ungültig
02335  c3be
02336  c3be  c6 e8           dec lsxp        Cursor-Zeile vor input nachführen
02337  c3c0  a6 e5     scru20 ldx sctop       BS-Fenster: oberste Zeile
02338  c3c2  e4 df           cpx keytmp
02339  c3c4  b0 02           bcs scru30
02340  c3c6  c6 df           dec keytmp
02341  c3c8
02342  c3c8  20 dc c3  scru30 jsr scr10       alle Bits für Normal-/Doppelzeilen
                                              nachführen
02343  c3cb
02344  c3cb  a6 e5           ldx sctop       BS-Fenster: oberste Zeile
02345  c3cd  20 76 cb        jsr getbt1      Test Zeile xr (Doppelz.: carry=1)
02346  c3d0
02347  c3d0  08              php
02348  c3d1  20 85 cb        jsr clrbit      Zeile als Normalzeile eintragen
02349  c3d4
02350  c3d4  28              plp
02351  c3d5  90 04           bcc scru10
02352  c3d7
02353  c3d7  24 f8           bit scroll      scroll-disable-Flag
02354  c3d9  30 cb           bmi scrup       nach oben scrollen
02355  c3db
02356  c3db  60        scru10 rts
02357  c3dc
02358  c3dc
02359  c3dc  ===> alle Bits für Normal-/Doppelzeilen nachführen <===
02360  c3dc
02361  c3dc  20 5e c1  scr10  jsr scrset      pnt auf Adresse BS-Zeile xr
02362  c3df  a4 e6           ldy sclf        BS-Fenster: Spalte linker Rand
02363  c3e1
02364  c3e1  e4 e4           cpx scbot       BS-Fenster: unterste Zeile
02365  c3e3  b0 0f           bcs scr60       Zeile löschen, Test auf
                                              Langsam-Scroll
02366  c3e5
02367  c3e5  e8              inx
02368  c3e6  20 76 cb        jsr getbt1      Test Zeile xr (Doppelz.: carry=1)
02369  c3e9
02370  c3e9  ca              dex
02371  c3ea  20 83 cb        jsr putbt1      Zeile in xr als Doppel-/Normalzeile
                                              eintragen
02372  c3ed
02373  c3ed  e8              inx
02374  c3ee  20 0d c4        jsr movlin      Zeilen im BS- und Attribut-RAM
                                              verschieben
02375  c3f1  4c dc c3        jmp scr10       alle Bits für Normal-/Doppelzeilen
                                              nachführen
02376  c3f4
02377  c3f4
02378  c3f4  ===> Zeile löschen, Test auf Langsam-Scroll <===
02379  c3f4
```

```
02380  c3f4  20 a5 c4   scr60  jsr clrln        akt. BS-Zeile als Normalzeile
                                                eintragen und löschen
02381  c3f7  a9 7f             lda #%01111111   Spalte 7 abfragen
02382  c3f9  8d 00 dc          sta d1pra
02383  c3fc
02384  c3fc  ad 01 dc          lda rows
02385  c3ff  c9 df             cmp #%11011111   Reihe 5 testen
02386  c401  d0 09             bne slorts       CBM-Taste nicht gedrückt
02387  c403
02388  c403  ===>  CBM-Taste: Warteschleife 0.5 s für langs. Scroll <===
02389  c403
02390  c403  a0 00             ldy #0
02391  c405  ea         slowmo nop
02392  c406  ca                dex
02393  c407  d0 fc             bne slowmo
02394  c409
02395  c409  88                dey
02396  c40a  d0 f9             bne slowmo
02397  c40c
02398  c40c  60         slorts rts
02399  c40d
02400  c40d
02401  c40d  ===> Zeilen im BS- und Attribut-RAM verschieben <===
02402  c40d
02403  c40d  24 d7      movlin bit mode          BS-Modus 40/80-Zeichen
02404  c40f  30 25             bmi movl80       Zeilen verschieben (80-Zeichen BS)
02405  c411
02406  c411  ===> Zeilen verschieben (40-Zeichen BS) <===
02407  c411
02408  c411  bd 33 c0          lda ldtb2,x      Tab. Anfangsadr. der BS-Zeilen lo
02409  c414  85 dc             sta sedeal       addrl Zeiger auf Endezeile
02410  c416  85 da             sta sedsal       addrl Zeiger auf Startzeile
02411  c418
02412  c418  bd 4c c0          lda ldtb1,x      Tab. Anfangsadr. der BS-Zeilen hi
02413  c41b  29 03             and #%00000011
02414  c41d  0d 3b 0a          ora ldtbsa       VIC Basisadresse BS-RAM
02415  c420  85 db             sta sedsah       addrh Zeiger auf Startzeile
02416  c422
02417  c422  29 03             and #%00000011
02418  c424  09 d8             ora #%11011000
02419  c426  85 dd             sta sedeah       addrh Zeiger auf Endezeile
02420  c428
02421  c428
02422  c428  ===> Schleife Verschieben in BS- und Farb-RAM <===
02423  c428
02424  c428  b1 da      movl10 lda (sedsal),y   addrl Zeiger auf Startzeile
02425  c42a  91 e0             sta (pnt),y      addrl Zeiger auf akt. BS-Zeile
02426  c42c
02427  c42c  b1 dc             lda (sedeal),y   addrl Zeiger auf Endezeile
02428  c42e  91 e2             sta (user),y     addrl Zeiger auf akt. Attributzeile
02429  c430
02430  c430  c4 e7             cpy scrt         BS-Fenster: Spalte rechter Rand
02431  c432  c8                iny
02432  c433  90 f3             bcc movl10       Schleife Verschieben in BS- und
                                                Farb-RAM
02433  c435
02434  c435  60                rts
02435  c436
```

```
02436  c436
02437  c436  ===> Zeilen verschieben (80-Zeichen BS) <===
02438  c436
02439  c436  8e 31 0a   movl80 stx sav80a       Zeilennummer
02440  c439  8c 32 0a          sty sav80b       Spalte
02441  c43c
02442  c43c  ===> Verschieben im BS-RAM <===
02443  c43c
02444  c43c  a2 18              ldx #24          VDC-Register 24 wählen,
02445  c43e  20 da cd           jsr vdcin        Daten vom VDC lesen, Register = xr
02446  c441
02447  c441  09 80              ora #%10000000   Bit7 = 1: Block wird kopiert nach
                                                 Schreiben Reg. 30
02448  c443  20 cc cd           jsr vdcout       ac in Register xr VDC schreiben
02449  c446
02450  c446  20 e6 cd           jsr getpnt       Textadresse im VDC setzen
02451  c449  ae 31 0a           ldx sav80a       VDC temporäres Register
02452  c44c  bd 33 c0           lda ldtb2,x      Tab. Anfangsadr. der BS-Zeilen lo
02453  c44f  0a                 asl a
02454  c450  85 da              sta sedsal       addrl Zeiger auf Startzeile
02455  c452
02456  c452  bd 4c c0           lda ldtb1,x      Tab. Anfangsadr. der BS-Zeilen hi
02457  c455  29 03              and #%00000011
02458  c457  2a                 rol a
02459  c458  0d 2e 0a           ora vm3          VDC Basisadresse BS-RAM
02460  c45b  85 db              sta sedsah       addrh Zeiger auf Startzeile
02461  c45d
02462  c45d  a2 20              ldx #32          VDC-Register 32 wählen
02463  c45f  18                 clc
02464  c460  98                 tya
02465  c461  65 da              adc sedsal       addrl Zeiger auf Startzeile
02466  c463  85 da              sta sedsal       addrl Zeiger auf Startzeile
02467  c465
02468  c465  a9 00              lda #0
02469  c467  65 db              adc sedsah       addrh Zeiger auf Startzeile
02470  c469  85 db              sta sedsah       addrh Zeiger auf Startzeile
02471  c46b
02472  c46b  20 cc cd           jsr vdcout       addrh Quelladresse BS-RAM für
                                                 Block-Copy an VDC
02473  c46e
02474  c46e  e8                 inx              VDC-Register 33 wählen
02475  c46f  a5 da              lda sedsal       addrl Zeiger auf Startzeile
02476  c471  20 cc cd           jsr vdcout       addrl Quelladresse BS-RAM für
                                                 Block-Copy an VDC
02477  c474
02478  c474  38                 sec
02479  c475  a6 e7              ldx scrt         BS-Fenster: Spalte rechter Rand
02480  c477  e8                 inx
02481  c478  8a                 txa
02482  c479  ed 32 0a           sbc sav80b       Spalte
02483  c47c  8d 32 0a           sta sav80b       Anzahl Copy-Zyklen
02484  c47f
02485  c47f  a2 1e              ldx #30          VDC-Register 30 wählen
02486  c481  20 cc cd           jsr vdcout       Anzahl Block-Copy-Zyklen an VDC,
                                                 Copy-Vorgang starten
02487  c484
02488  c484
```

```
02489  c484  ===> Verschieben im Attribut-RAM <===
02490  c484
02491  c484  a2 20          ldx #32          ,VDC-Register 32 wählen
02492  c486  a5 db          lda sedsah       addrh Zeiger auf Startzeile
02493  c488  29 07          and #%00000111   addrh Quelladresse in Attribut-RAM
                                              berechnen
02494  c48a  0d 2f 0a       ora vm4          VDC Basisadresse Attribut-RAM
02495  c48d  20 cc cd       jsr vdcout       addrh Quelladresse Attribut-RAM für
                                              Block-Copy an VDC
02496  c490
02497  c490  e8             inx
02498  c491  a5 da          lda sedsal       addrl Zeiger auf Startzeile
02499  c493  20 cc cd       jsr vdcout       addrl Quelladresse Attribut-RAM für
                                              Block-Copy an VDC
02500  c496
02501  c496  20 f9 cd       jsr getusr       Attributadresse im VDC setzen
02502  c499
02503  c499  ad 32 0a       lda sav80b       VDC temporäres Register
02504  c49c  a2 1e          ldx #30          VDC-Register 30 wählen
02505  c49e  20 cc cd       jsr vdcout       Anzahl Block-Copy-Zyklen an VDC,
                                              Copy-Vorgang starten
02506  c4a1
02507  c4a1  ae 31 0a       ldx sav80a       Zeilennummer
02508  c4a4  60             rts
02509  c4a5
02510  c4a5
02511  c4a5  ===> akt. BS-Zeile als Normalzeile eintragen und löschen <===
02512  c4a5
02513  c4a5  a4 e6     clrln ldy sclf        BS-Fenster: Spalte linker Rand
02514  c4a7  20 85 cb        jsr clrbit      Zeile als Normalzeile eintragen
02515  c4aa
02516  c4aa
02517  c4aa
02518  c4aa  ===> BS-Zeile löschen <===
02519  c4aa
02520  c4aa  20 5e c1  clrprt jsr scrset     pnt auf Adresse BS-Zeile xr
02521  c4ad  24 d7            bit mode       BS-Modus 40/80-Zeichen
02522  c4af  30 0f            bmi clr80      akt. Zeile löschen (80-Zeichen BS)
02523  c4b1
02524  c4b1  ===> akt. Zeile löschen (40-Zeichen BS) <===
02525  c4b1
02526  c4b1  88               dey
02527  c4b2  c8        clrpt1 iny
02528  c4b3  a9 20            lda #' '
02529  c4b5  91 e0            sta (pnt),y    addrl Zeiger auf akt. BS-Zeile
02530  c4b7
02531  c4b7  a5 f1            lda color      akt. Attribut
02532  c4b9  91 e2            sta (user),y   addrl Zeiger auf akt. Attributzeile
02533  c4bb
02534  c4bb  c4 e7            cpy scrt       BS-Fenster: Spalte rechter Rand
02535  c4bd  d0 f3            bne clrpt1
02536  c4bf
02537  c4bf  60               rts
02538  c4c0
02539  c4c0
02540  c4c0  ===> akt. Zeile löschen (80-Zeichen BS) <===
02541  c4c0
02542  c4c0  8e 31 0a   clr80  stx sav80a      Zeile
```

```
02543  c4c3  8c 32 0a          sty sav80b       Spalte
02544  c4c6
02545  c4c6  ===> Zeile im BS-RAM löschen <===
02546  c4c6
02547  c4c6  a2 18             ldx #24          VDC-Register 24 wählen,
02548  c4c8  20 da cd          jsr vdcin        Daten vom VDC lesen, Register = xr
02549  c4cb  29 7f             and #%01111111   Bit7 = 0: Block schreiben nach
                                                 Schreiben Reg. 30
02550  c4cd  20 cc cd          jsr vdcout       ac in Register xr VDC schreiben
02551  c4d0
02552  c4d0  a2 12             ldx #18          VDC-Register 18 wählen
02553  c4d2  18                clc
02554  c4d3  98                tya
02555  c4d4  65 e0             adc pnt          addrl Zeiger auf akt. BS-Zeile
02556  c4d6  48                pha
02557  c4d7  8d 3c 0a          sta clreal       clear end address lo-byte
02558  c4da
02559  c4da  a9 00             lda #0
02560  c4dc  65 e1             adc pnt+1        addrh Zeiger auf akt. BS-Zeile
02561  c4de  8d 3d 0a          sta clreah       clear end address hi-byte
02562  c4e1  20 cc cd          jsr vdcout       addrh akt. Schreib-/Leseposition im
                                                 BS-RAM an VDC
02563  c4e4
02564  c4e4  e8                inx
02565  c4e5  68                pla
02566  c4e6  20 cc cd          jsr vdcout       addrl akt. Schreib-/Leseposition im
                                                 BS-RAM an VDC
02567  c4e9
02568  c4e9  a9 20             lda #' '         Space zum Löschen
02569  c4eb  20 ca cd          jsr vdcput       Daten für Block-/normales Schreiben
                                                 an VDC
02570  c4ee
02571  c4ee  38                sec
02572  c4ef  a5 e7             lda scrt         BS-Fenster: Spalte rechter Rand
02573  c4f1  ed 32 0a          sbc sav80b       Spalte
02574  c4f4  48                pha
02575  c4f5  f0 14             beq clrpt3       Zeile im Attribut-RAM löschen
02576  c4f7
02577  c4f7  aa                tax
02578  c4f8  38                sec
02579  c4f9  6d 3c 0a          adc clreal       clear end address lo-byte
02580  c4fc  8d 3c 0a          sta clreal       clear end address lo-byte
02581  c4ff
02582  c4ff  a9 00             lda #0
02583  c501  6d 3d 0a          adc clreah       clear end address hi-byte
02584  c504  8d 3d 0a          sta clreah       clear end address hi-byte
02585  c507
02586  c507  8a                txa
02587  c508  20 3e c5          jsr clrpt6       ac Schreibzyklen an VDC, durchführen
                                                 bis clr-end-address
02588  c50b
02589  c50b
02590  c50b  ===> Zeile im Attribut-RAM löschen <===
02591  c50b
02592  c50b  a2 12       clrpt3 ldx #18         VDC-Register 18 wählen
02593  c50d  18                 clc
02594  c50e  98                 tya
02595  c50f  65 e2              adc user        addrl Zeiger auf akt. Attributzeile
```

```
02596  c511  48              pha
02597  c512
02598  c512  a9 00           lda #0
02599  c514  65 e3           adc user+1        addrh Zeiger auf akt. Attributzeile
02600  c516  20 cc cd        jsr vdcout        addrh akt. Schreibposition im BS-RAM
                                                an VDC
02601  c519
02602  c519  e8              inx               VDC-Register 19 wählen
02603  c51a  68              pla
02604  c51b  20 cc cd        jsr vdcout        addrl akt. Schreibposition im BS-RAM
                                                an VDC
02605  c51e
02606  c51e  ad 3d 0a        lda clreah        clear end address hi-byte
02607  c521  29 07           and #%00000111
02608  c523  0d 2f 0a        ora vm4           VDC Basisadresse Attribut-RAM
02609  c526  8d 3d 0a        sta clreah        clear end address hi-byte
02610  c529
02611  c529  a5 f1           lda color         akt. Attribut
02612  c52b  29 8f           and #%10001111    alle Attribute löschen
02613  c52d  20 ca cd        jsr vdcput        Daten für Block-/normales Schreiben
                                                an VDC
02614  c530
02615  c530  68              pla
02616  c531  f0 03           beq clrpt4
02617  c533
02618  c533  20 3e c5        jsr clrpt6        ac Schreibzyklen an VDC, durchführen
                                                bis clr-end-address
02619  c536  ae 31 0a  clrpt4 ldx sav80a        VDC temporäres Register
02620  c539  a4 e7           ldy scrt          BS-Fenster: Spalte rechter Rand
02621  c53b  60              rts
02622  c53c
02623  c53c
02624  c53c
02625  c53c  ===> 1 Schreibzyklus an VDC, durchführen bis clr-end-address <===
02626  c53c
02627  c53c  a9 01     clrpt5 lda #1
02628  c53e
02629  c53e
02630  c53e  ===> ac Schreibzyklen an VDC, durchführen bis clr-end-address <===
02631  c53e
02632  c53e  a2 1e     clrpt6 ldx #30           VDC-Register 30 wählen
02633  c540  20 cc cd         jsr vdcout       Anzahl Block-Write-/Block-Copy
                                                Zyklen an VDC
02634  c543
02635  c543  2c 00 d6  clrpt7 bit vdcadr       VDC 8563 Adress-Register
02636  c546  10 fb             bpl clrpt7
02637  c548
02638  c548  a2 12             ldx #18         VDC-Register 18 wählen
02639  c54a  20 da cd          jsr vdcin       addrh. akt. Schreib-/Lese-Position
                                                VDC nach ac
02640  c54d  cd 3d 0a          cmp clreah      clear end address hi-byte
02641  c550  90 ea             bcc clrpt5      1 Schreibzyklus an VDC, durchführen
                                                bis clr-end-address
02642  c552
02643  c552  a2 13             ldx #19         VDC-Register 19 wählen
02644  c554  20 da cd          jsr vdcin       addrl. akt. Schreib-/Lese-Position
                                                VDC nach ac
02645  c557  cd 3c 0a          cmp clreal      clear end address lo-byte
```

```
02646  c55a  90 e0           bcc clrpt5      1 Schreibzyklus an VDC, durchführen
                                             bis clr-end-address
02647  c55c
02648  c55c  60              rts
02649  c55d
02650  c55d
02651  c55d                  .end
02652  c55d                  .lib c128.keyboard1
02653  c55d
```

4.1.9 Tastatur-Abfrage

```
02655  c55d
02656  c55d
02657  c55d                ===> T A S T A T U R  -  M A T R I X <===
02658  c55d
02659  c55d
02660  c55d  !-------!--------------!-----------------------------------------!
02661  c55d  !CIA 1  !      VIC     !                  CIA 1                  !
02662  c55d  !PORT B !     PORT     !                 PORT A                  !
02663  c55d  !REIHE: !    Spalte:   !                 Spalte:                 !
02664  c55d  !------ !              !                                         !
02665  c55d  !       !  2   1   0  !  7   6   5   4   3   2   1   0         !
02666  c55d  !       !--------------!-----------------------------------------!
02667  c55d  !  0    ! ALT ESC HLP !  1  BSLH  +   9   7   5   3  DEL       !
02668  c55d  !  1    !  0   +   8  ! Pfl.  *   P   I   Y   R   W  CR        !
02669  c55d  !  2    !  .   -   5  ! CTRL  ;   L   J   G   D   A  Cri       !
02670  c55d  !  3    ! Cup  LF TAB !  2  HOME  -   0   8   6   4  f7        !
02671  c55d  !  4    ! Cdn  CR   2  ! SPC SHFr  .   M   B   C   Z  f1        !
02672  c55d  !  5    ! Cle  6    4  ! CBM   =   :   K   H   F   S  f3        !
02673  c55d  !  6    ! Cri  9    7  !  Q  Pfo.  @   O   U   T   E  f5        !
02674  c55d  !  7    ! NOSC 3    1  ! STOP  /   ,   N   V   X  SHFl Cdn     !
02675  c55d  !-------!              !-----------------------------------------!
02676  c55d          !  Reihe 1-7:  !
02677  c55d          !  Zehnerblock !
02678  c55d          !--------------!
02679  c55d
02680  c55d             matrix                  Eintrag in Verweis-Tabelle
02681  c55d
02682  c55d
02683  c55d
02684  c55d  ===> Tastatur Abfrage <===
02685  c55d
02686  c55d  a5 01       scnkey lda r8502       8502 CPU-Port Datenregister
02687  c55f  29 40              and #%01000000  Bit6 isolieren (0 = DIN-Tastatur)
02688  c561  49 40              eor #%01000000  flippen
02689  c563  4a                 lsr a           nach Bit 4 (1 = DIN-Tastatur)
02690  c564  4a                 lsr a
02691  c565  85 d3              sta shflag      Flag für Shift/CTRL/CBM-Taste
02692  c567
02693  c567  a0 58              ldy #88         Vorbesetzen mit Wert
02694  c569  84 d4              sty sfdx        für 'keine Taste gedrückt'
02695  c56b
02696  c56b  a9 00              lda #%00000000
02697  c56d  8d 00 dc           sta d1pra       alle Spalten gleichzeitig abfragen
02698  c570  8d 2f d0           sta vicpen      VIC 8564 Port-Enable
```

```
02699  c573
02700  c573  ae 01 dc          ldx rows          alle Tastatur-Zeilen abfragen
02701  c576  e0 ff             cpx #%11111111    Wert für keine Taste gedrückt
02702  c578  d0 03             bne scn10         Taste gedrückt
02703  c57a
02704  c57a  4c 97 c6          jmp ckit2         Abschluß Tastatur-Abfrage
02705  c57d
02706  c57d
02707  c57d  ===> Taste gedrückt, dekodieren <===
02708  c57d
02709  c57d  a8          scn10  tay               0 = Zeiger in keytab
02710  c57e
02711  c57e  ad 3e 03           lda nrmtab        addrl akt. Tast.tab. normal
02712  c581  85 cc              sta keytab        addrl Zeiger in Tast.-Dekodier-Tab.
02713  c583
02714  c583  ad 3f 03           lda nrmtab+1      addrh akt. Tast.tab. normal
02715  c586  85 cd              sta keytab+1      addrh Zeiger in Tast.-Dekodier-Tab.
02716  c588
02717  c588  a9 ff              lda #%11111111
02718  c58a
02719  c58a  8d 2f d0           sta vicpen        VIC 8564 Port-Enable
02720  c58d
02721  c58d  2a                 rol a             Abfrage Spalte 0 voreinstellen:
                                                  11111110
02722  c58e
02723  c58e
02724  c58e  ===> nächste Tast.spalte abfragen <===
02725  c58e
02726  c58e  24 d3       scn20  bit shflag        Flag für Shift/CTRL/CBM-Taste
02727  c590
02728  c590  7.......  Spaltenabfrage CIA 1 / VIC-Port
02729  c590
02730  c590  30 05              bmi scn21         Spaltenabfrage mit VIC-Port
02731  c592
02732  c592  ===> Spaltenabfrage mit CIA 1 Port A <===
02733  c592
02734  c592  8d 00 dc           sta d1pra         Bitmuster für Spalte in CIA 1 Port A
02735  c595  10 03              bpl scn22         immer
02736  c597
02737  c597
02738  c597  ===> Spaltenabfrage mit VIC-Port <===
02739  c597
02740  c597  8d 2f d0    scn21  sta vicpen        Bitmuster für Spalte in VIC-Port
02741  c59a
02742  c59a
02743  c59a  ===> 8 Reihen akt. Spalte abfragen <===
02744  c59a
02745  c59a  a2 08       scn22  ldx #8           Zähler für 8 Reihen
02746  c59c  48                 pha               Bitmuster für akt. Spalte
02747  c59d
02748  c59d
02749  c59d  ===> Tastatur entprellen <===
02750  c59d
02751  c59d  ad 01 dc    scn24  lda rows          CIA 1 Port B lesen: Tast. Reihen
02752  c5a0  cd 01 dc           cmp rows          Tastatur
02753  c5a3  d0 f8              bne scn24         entprellen
02754  c5a5
02755  c5a5
```

```
02756  c5a5  ===> Tast. Reihen abfragen <===
02757  c5a5
02758  c5a5  4a          scn30  lsr a
02759  c5a6  b0 17              bcs ckit          keine Taste akt. Reihe gedrückt,
                                                   weiter
02760  c5a8
02761  c5a8  48                 pha               Bitmuster gelesene Reihen auf Stapel
02762  c5a9
02763  c5a9  b1 cc              lda (keytab),y    ASCII für Taste aus Tabelle
02764  c5ab
02765  c5ab  ===> reservierte Werte der Tastaturtabelle eintragen <===
02766  c5ab
02767  c5ab  c9 08              cmp #8            Code für ALT
02768  c5ad  f0 08              beq scn35         Flag für Taste ALT in shflag setzen
02769  c5af
02770  c5af  c9 05              cmp #5
02771  c5b1  b0 09              bcs spck2         größer 4, nicht reserviert
02772  c5b3
02773  c5b3  c9 03              cmp #3
02774  c5b5  f0 05              beq spck2         nicht reserviert
02775  c5b7
02776  c5b7
02777  c5b7  ===> reserv. Tasten eintragen <===
02778  c5b7
02779  c5b7  05 d3       scn35  ora shflag        Flags für reserv. Tasten setzen
02780  c5b9
02781  c5b9   ....3210   Flags für reserv. Tasten
02782  c5b9   xxxx1xxx   ALT
02783  c5b9   xxxxx1xx   CTRL
02784  c5b9   xxxxxx1x   CBM-Taste
02785  c5b9   xxxxxxx1   Shift li./re.
02786  c5b9
02787  c5b9  85 d3              sta shflag        Flag für Shift/CTRL/CBM-Taste
02788  c5bb  2c                 .byte skip2
02789  c5bc
02790  c5bc  ===> Tastennummer speichern, weiter abfragen <===
02791  c5bc
02792  c5bc
02793  c5bc  ===> Nummer gedr. Taste speichern <===
02794  c5bc
02795  c5bc  84 d4       spck2  sty sfdx          Nummer der akt. Taste
02796  c5be
02797  c5be  68                 pla               Bitmuster gelesene Reihen vom Stapel
02798  c5bf
02799  c5bf
02800  c5bf  ===> nächste Reihe abfragen <===
02801  c5bf
02802  c5bf  c8          ckit   iny               Zeiger in Tabelle erhöhen
02803  c5c0  ca                 dex               Zähler für Reihen erniedrigen
02804  c5c1  d0 e2              bne scn30         Tast. Reihen abfragen
02805  c5c3
02806  c5c3  ===> Test auf Ende Tastaturabfrage <===
02807  c5c3
02808  c5c3  c0 59              cpy #89           Zeiger in keytab mit 89 vergleichen
02809  c5c5  b0 10              bcs ckit1         verzweigen, wenn Tastaturabfrage
                                                   fertig
02810  c5c7
```

```
02811  c5c7  ===> nächste Tastaturspalte abfragen <===
02812  c5c7
02813  c5c7  68              pla             Tastaturspalte vom Stapel
02814  c5c8  38              sec
02815  c5c9  2a              rol a           0-Bit eine Position nach links
02816  c5ca  b0 c2           bcs scn20       verzweigen, wenn 0-Bit noch im ac
02817  c5cc
02818  c5cc  ===> Umschalten Spaltenabfrage auf VIC-Port <===
02819  c5cc
02820  c5cc  8d 00 dc        sta d1pra       %11111111 ;keine Spalte CIA 1
                                             abfragen
02821  c5cf
02822  c5cf  26 d3           rol shflag      Flag für Shift/CTRL/CBM-Taste
02823  c5d1  38              sec
02824  c5d2  66 d3           ror shflag      Bit7 setzen, Flag für Abfrage
                                             VIC-Port
02825  c5d4
02826  c5d4  2a              rol a           %11111110 ;Spalte 0 VIC-Port
                                             voreinstellen
02827  c5d5  d0 b7           bne scn20       immer
02828  c5d7
02829  c5d7
02830  c5d7  06 d3     ckit1 asl shflag      Flag für Shift/CTRL/CBM-Taste
02831  c5d9  46 d3           lsr shflag      Flag für Shift/CTRL/CBM-Taste
02832  c5db
02833  c5db  68              pla             Bitmuster Spalte verwerfen
02834  c5dc
02835  c5dc  a5 d4           lda sfdx        Nummer der akt. Taste
02836  c5de  6c 3a 03        jmp (keyvec)    Vector-Vorgabe: keylog ($c5e1)
02837  c5e1
02838  c5e1
02839  c5e1  ===> Test auf NO SCROLL Taste <===
02840  c5e1
02841  c5e1  c9 57     keylog cmp #87        Nummer Taste NO SCROLL
02842  c5e3  d0 13           bne shflg1      Test auf reserv. Tasten
02843  c5e5
02844  c5e5  ===> Bearbeitung NO SCROLL Taste <===
02845  c5e5
02846  c5e5  24 f7           bit locks       Flag enable/disable Shift/CBM-Taste
02847  c5e7
02848  c5e7   x1xxxxxx  CTR s / NO SCROLL unwirksam
02849  c5e7
02850  c5e7  70 5a           bvs kylg21      Abschluß
02851  c5e9
02852  c5e9  ad 25 0a        lda lstshf
02853  c5ec  d0 55           bne kylg21
02854  c5ee
02855  c5ee  a9 0d           lda #13         Flag flippen (CTRL s = 13)
02856  c5f0  4d 21 0a        eor pause       Flag für CTRL s/ NO SCROLL
02857  c5f3  8d 21 0a        sta pause       Flag für CTRL s/ NO SCROLL
02858  c5f6
02859  c5f6  50 30           bvc shflg3      immer
02860  c5f8
02861  c5f8
02862  c5f8  ===> Test auf reserv. Tasten <===
02863  c5f8
02864  c5f8  a5 d3     shflg1 lda shflag      Flag für Shift/CTRL/CBM-Taste
02865  c5fa
```

```
02866  c5fa   ...43210  Flags für reserv. Tasten
02867  c5fa   xxx1xxxx  CAPS LOCK  DIN/ASCII
02868  c5fa   xxxx1xxx  ALT
02869  c5fa   xxxxx1xx  CTRL
02870  c5fa   xxxxxx1x  CBM-Taste
02871  c5fa   xxxxxxx1  Shift li./re.
02872  c5fa
02873  c5fa  f0 55             beq repeat       keine reserv. Taste gedrückt
02874  c5fc
02875  c5fc  ===> Test auf CAPS LOCK - ASCII/DIN - Taste <===
02876  c5fc
02877  c5fc  c9 10             cmp #%00010000   CAPS-LOCK/DIN-Modus
02878  c5fe  f0 44             beq capslk       Tastaturtabelle mode5
02879  c600
02880  c600   CAPS/LOCK Modus einstellen:
02881  c600
02882  c600   *  SHIFT CBM (Klein-/Groß)
02883  c600   *  Taste CAPS/LOCK nicht gedrückt
02884  c600   *  poke2757,128
02885  c600
02886  c600   DIN ohne Akzente einstellen:
02887  c600
02888  c600   *  SHIFT CBM (Klein-/Groß)
02889  c600   *  Taste CAPS/LOCK gedrückt
02890  c600   *  poke2757,128
02891  c600
02892  c600
02893  c600  ===> Test auf gedrückte ALT Taste <===
02894  c600
02895  c600  c9 08             cmp #%00001000
02896  c602  f0 42             beq notkat       ALT Tastatur-Tabelle (Ersatzwert
                                                Modus1)
02897  c604
02898  c604  ===> Test auf gedrückte SHIFT CBM  Taste <===
02899  c604
02900  c604  29 07             and #%00000111   CTRL, CBM, SHIFT maskieren
02901  c606  c9 03             cmp #%00000011   SHIFT CBM ?
02902  c608  d0 25             bne keylg2       nicht SHIFT CBM
02903  c60a
02904  c60a  ===> Bearbeitung SHIFT CBM Taste <===
02905  c60a
02906  c60a  a5 f7             lda locks        Flag enable/disable Shift/CBM-Taste
02907  c60c  30 43             bmi repeat       Bit7: 1 = shift-Commodore gesperrt
02908  c60e
02909  c60e  ad 25 0a          lda lstshf
02910  c611  d0 3e             bne repeat       Code aus Tastatur-Tab. lesen
02911  c613
02912  c613  ===> Umschalten Groß-/Klein- auf Groß-/Graphik-Modus <===
02913  c613
02914  c613  24 d7             bit mode         BS-Modus 40/80-Zeichen
02915  c615  10 09             bpl shflg2       40-Zeichen BS
02916  c617
02917  c617  ===> 80-Zeichen BS <===
02918  c617
02919  c617  a5 f1             lda color        akt. Attribut
02920  c619  49 80             eor #%10000000   Bit7 flippen
02921  c61b  85 f1             sta color        akt. Attribut
02922  c61d
```

```
02923  c61d  4c 28 c6          jmp shflg3       Wartezeit einstellen
02924  c620
02925  c620
02926  c620  ===> 40-Zeichen BS <===
02927  c620
02928  c620  ad 2c 0a   shflg2 lda vm1           VIC Text-Basiszeiger
02929  c623  49 02             eor #%00000010
02930  c625  8d 2c 0a          sta vm1           VIC Text-Basiszeiger
02931  c628
02932  c628
02933  c628  ===> Wartezeit einstellen <===
02934  c628
02935  c628  a9 08      shflg3 lda #8
02936  c62a  8d 25 0a          sta lstshf
02937  c62d  d0 22             bne repeat        immer
02938  c62f
02939  c62f
02940  c62f
02941  c62f  ===> Test auf CTRL Taste <===
02942  c62f
02943  c62f  0a         keylg2 asl a
02944  c630  c9 08             cmp #%00001000    Bitmuster für CTRL
02945  c632  90 12             bcc notkat        CTRL nicht gedrückt
02946  c634
02947  c634  ===> Bearbeitung CTRL Taste <===
02948  c634
02949  c634  a9 06             lda #6            Zeiger auf Adresse CTRL Tast.tab.
02950  c636
02951  c636  ===> Test auf CTRL s <===
02952  c636
02953  c636  a6 d4             ldx sfdx          Nummer der akt. Taste
02954  c638  e0 0d             cpx #13           Nummer der Taste 's'
02955  c63a  d0 0a             bne notkat        nicht CTRL s
02956  c63c
02957  c63c  ===> Bearbeitung CTRL s <===
02958  c63c
02959  c63c  24 f7             bit locks         Flag enable/disable Shift/CBM-Taste
02960  c63e  70 06             bvs notkat        Adresse Tast.-Tab. (ac) eintragen
02961  c640
02962  c640  8e 21 0a          stx pause         Tast.nr 13: Flag Ausgabe Stopp
02963  c643  60         kylg21 rts
02964  c644
02965  c644
02966  c644  ===> CAPS-LOCK-Modus <===
02967  c644
02968  c644  a9 0a      capslk lda #10           Zeiger auf Adresse CAPS Tast.tab.
02969  c646
02970  c646
02971  c646  ===> Adresse Tast.-Tab. (ac) eintragen <===
02972  c646
02973  c646  aa         notkat tax
02974  c647  bd 3e 03          lda nrmtab,x      addrl akt. Tast.tab. normal
02975  c64a  85 cc             sta keytab        addrl Zeiger in Tast.-Dekodier-Tab.
02976  c64c
02977  c64c  bd 3f 03          lda nrmtab+1,x    addrh akt. Tast.tab. normal
02978  c64f  85 cd             sta keytab+1      addrh Zeiger in Tast.-Dekodier-Tab.
02979  c651
02980  c651
```

```
02981  c651  ===> Code aus Tastatur-Tab. lesen <===
02982  c651
02983  c651  a4 d4      repeat ldy sfdx          Nummer der akt. Taste
02984  c653  b1 cc             lda (keytab),y    addrl Zeiger in Tast.-Dekodier-Tab.
02985  c655  aa               tax                Tastencode
02986  c656  c4 d5             cpy lstx          letzte gedrückte Taste
02987  c658  f0 07             beq rpt10         Test auf REPEAT für alle Tasten
02988  c65a
02989  c65a  ===> Wartezeit vor REPEAT-Einsatz initialisieren <===
02990  c65a
02991  c65a  a0 10             ldy #16
02992  c65c  8c 24 0a          sty delay         Zeit bis zum Einsetzen des REPEAT
02993  c65f  d0 36             bne ckit2
02994  c661
02995  c661
02996  c661
02997  c661  ===> Test auf REPEAT für alle Tasten <===
02998  c661
02999  c661  29 7f      rpt10  and #%01111111
03000  c663
03001  c663  2c 22 0a          bit rptflg        Flag für REPEAT, alle Tasten = $80
03002  c666  30 16             bmi rpt20         REPEAT für alle Tasten
03003  c668  70 5a             bvs scnrts        REPEAT abgeschaltet
03004  c66a
03005  c66a  ===> Steuertasten erkennen <===
03006  c66a
03007  c66a  c9 7f             cmp #%01111111    NO SCROLL ($ff)
03008  c66c  f0 29             beq ckit2
03009  c66e
03010  c66e  c9 14             cmp #$14          delete/insert
03011  c670  f0 0c             beq rpt20         Wartezeit zw. Tastenwiederholung
03012  c672
03013  c672  c9 20             cmp #' '          space/shift-space
03014  c674  f0 08             beq rpt20         Wartezeit zw. Tastenwiederholung
03015  c676
03016  c676  c9 1d             cmp #$1d          Cursor rechts/links
03017  c678  f0 04             beq rpt20         Wartezeit zw. Tastenwiederholung
03018  c67a
03019  c67a  c9 11             cmp #$11          Cursor up/down
03020  c67c  d0 46             bne scnrts        Abschluß Tastatur-Abfrage
03021  c67e
03022  c67e
03023  c67e  ===> Wartezeit zw. Tastenwiederholung <===
03024  c67e
03025  c67e  ac 24 0a   rpt20  ldy delay         Zeit bis zum Einsetzen des REPEAT
03026  c681  f0 05             beq rpt40
03027  c683
03028  c683  ce 24 0a          dec delay         Zeit bis zum Einsetzen des REPEAT
03029  c686  d0 3c             bne scnrts        Abschluß Tastatur-Abfrage
03030  c688
03031  c688  ce 23 0a   rpt40  dec kount         Zähler für REPEAT-Geschwindigkeit
03032  c68b  d0 37             bne scnrts        Abschluß Tastatur-Abfrage
03033  c68d
03034  c68d  ===> Wartezeit zw. Wiederholung initialisieren <===
03035  c68d
03036  c68d  a0 04             ldy #4
03037  c68f  8c 23 0a          sty kount         Zähler für REPEAT-Geschwindigkeit
03038  c692
```

```
03039  c692  a4 d0            ldy ndx          Bytes im Tast.puffer
03040  c694  88               dey
03041  c695  10 2d            bpl scnrts       Abschluß Tastatur-Abfrage
03042  c697
03043  c697  4e 25 0a  ckit2  lsr lstshf
03044  c69a  a4 d4            ldy sfdx         Nummer der akt. Taste
03045  c69c  84 d5            sty lstx         letzte gedrückte Taste
03046  c69e
03047  c69e  e0 ff            cpx #$ff         Tastencode mit NO SCROLL vergleichen
03048  c6a0  f0 22            beq scnrts       Abschluß Tastatur-Abfrage
03049  c6a2
03050  c6a2  ===> jede andere Taste setzt CTRL s/NO SCROLL zurück <===
03051  c6a2
03052  c6a2  a9 00            lda #0
03053  c6a4  8d 21 0a         sta pause        Flag für CTRL s/ NO SCROLL
03054  c6a7
03055  c6a7  8a               txa              Tastencode
03056  c6a8  a6 d3            ldx shflag       Flag für Shift/CTRL/CBM-Taste
03057  c6aa
03058  c6aa  im USA-Standard C128 folgt jmp (keychk) auf keyput
03059  c6aa
03060  c6aa  4c c6 fc         jmp chkdin       Test ASCII/DIN, jmp (keychk)
03061  c6ad
03062  c6ad
03063  c6ad  ===> Bearbeitung Funktionstasten <===
03064  c6ad
03065  c6ad  a2 09      keyput ldx #9
03066  c6af  dd dd c6   funts1 cmp funtab,x    ASCII-Codes der Funktionstasten
03067  c6b2  f0 16            beq dopfky       Länge/Zeiger auf akt.
                                               Funktionstastentext holen
03068  c6b4
03069  c6b4  ca               dex
03070  c6b5  10 f8            bpl funts1
03071  c6b7
03072  c6b7  ===> Test auf Tastaturpuffer voll <===
03073  c6b7
03074  c6b7  a6 d0            ldx ndx          Bytes im Tast.puffer
03075  c6b9  ec 20 0a         cpx xmax         max. Länge Tast.puffer
03076  c6bc  b0 06            bcs scnrts       Abschluß Tastatur-Abfrage
03077  c6be
03078  c6be  ===> ac in Tastaturpuffer abspeichern <===
03079  c6be
03080  c6be  9d 4a 03         sta keyd,x       Tast.puffer
03081  c6c1  e8               inx
03082  c6c2  86 d0            stx ndx          Bytes im Tast.puffer
03083  c6c4
03084  c6c4
03085  c6c4  ===> Abschluß Tastatur-Abfrage <===
03086  c6c4
03087  c6c4  a9 7f      scnrts lda #%01111111  Spalte 7 abfr. (Stopp-Taste)
03088  c6c6  8d 00 dc         sta d1pra
03089  c6c9
03090  c6c9  60               rts
03091  c6ca
03092  c6ca
```

```
03093  c6ca  ===> Länge/Zeiger auf akt. Funktionstastentext holen <===
03094  c6ca
03095  c6ca  bd 00 10   dopfky lda pkybuf,x     Länge des Texts Fkt.taste (xr)
03096  c6cd  85 d1             sta kyndx       Bytes im Fkt.tasten-Puffer
03097  c6cf
03098  c6cf  ===> Zeiger auf Funtiontasten-Text berechnen <===
03099  c6cf
03100  c6cf  a9 00             lda #0
03101  c6d1  ca         fungt1 dex
03102  c6d2  30 06             bmi fungt3
03103  c6d4
03104  c6d4  18                clc
03105  c6d5  7d 00 10          adc pkybuf,x    Länge des Texts Fkt.taste (xr)
03106  c6d8  90 f7             bcc fungt1
03107  c6da
03108  c6da  85 d2      fungt3 sta keyidx      Zeiger in Text akt. Fkt.taste
03109  c6dc
03110  c6dc  60                rts
03111  c6dd
03112  c6dd
03113  c6dd
03114  c6dd  ===> ASCII-Codes der Funktionstasten <===
03115  c6dd
03116  c6dd  85         funtab .byte $85        Code Fkt.taste f1 (graphic)
03117  c6de  89                .byte $89        Code Fkt.taste f2 (dload")
03118  c6df  86                .byte $86        Code Fkt.taste f3 (directory)
03119  c6e0  8a                .byte $8a        Code Fkt.taste f4 (scnclr)
03120  c6e1  87                .byte $87        Code Fkt.taste f5 (dsave")
03121  c6e2  8b                .byte $8b        Code Fkt.taste f6 (run)
03122  c6e3  88                .byte $88        Code Fkt.taste f7 (list)
03123  c6e4  8c                .byte $8c        Code Fkt.taste f8 (monitor)
03124  c6e5
03125  c6e5  83                .byte $83        Code Taste (shift) RUN
03126  c6e6  84                .byte $84        Code Taste HELP
03127  c6e7
03128  c6e7
03129  c6e7  ===> Cursor ein/aus 40-Zeichen BS <===
03130  c6e7
03131  c6e7  24 d7      blink  bit mode         BS-Modus 40/80-Zeichen
03132  c6e9  30 41             bmi setc42      nur 40-Zeichen BS
03133  c6eb
03134  c6eb  ad 27 0a          lda blnsw       VIC Flag für Cursor-Blinken
03135  c6ee  d0 3c             bne setc42
03136  c6f0
03137  c6f0  ce 28 0a          dec curcnt      VIC Zähler für Cursor-Blinken
03138  c6f3  d0 37             bne setc42
03139  c6f5
03140  c6f5  ad 26 0a          lda blnon       Cursor-Modus 40-Zeichen
03141  c6f8
03142  c6f8   1xxxxxxx  Cursor RVS
03143  c6f8   x1xxxxxx  Cursor blinkt nicht
03144  c6f8
03145  c6f8  29 c0             and #%11000000
03146  c6fa  c9 c0             cmp #%11000000
03147  c6fc  f0 2e             beq setc42      fester Cursor bereits eingeschaltet
03148  c6fe
```

```
03149  c6fe  ===> Rückwärtszähler für BLINK einrichten <===
03150  c6fe
03151  c6fe  a9 14           lda #20
03152  c700  8d 28 0a        sta curcnt      VIC Zähler für Cursor-Blinken
03153  c703
03154  c703  a4 ec           ldy pntr        akt. Cursor-Spalte
03155  c705  ae 2a 0a        ldx gdcol       VIC Cursorfarbe vor Blinken
03156  c708  b1 e0           lda (pnt),y     addrl Zeiger auf akt. BS-Zeile
03157  c70a
03158  c70a  2c 26 0a        bit blnon       Cursor-Modus 40-Zeichen
03159  c70d  30 10           bmi setc41      REVERS-Bit (7) in BS-RAM flippen
03160  c70f
03161  c70f  ===> Zeichen unter Cursor RVS/normal umschalten <===
03162  c70f
03163  c70f  8d 29 0a        sta gdbln       VIC Cursorzeichen vor Blinken
03164  c712
03165  c712  20 7c c1        jsr scolor      Zeiger in Farb-RAM nachführen
03166  c715  b1 e2           lda (user),y    addrl Zeiger auf akt. Attributzeile
03167  c717  8d 2a 0a        sta gdcol       VIC Cursorfarbe vor Blinken
03168  c71a
03169  c71a  a6 f1           ldx color       akt. Attribut
03170  c71c  ad 29 0a        lda gdbln       VIC Cursorzeichen vor Blinken
03171  c71f
03172  c71f
03173  c71f  ===> REVERS-Bit (7) in BS-RAM flippen <===
03174  c71f
03175  c71f  49 80     setc41 eor #%10000000
03176  c721  20 40 cc        jsr displ       BS-Ausgabe ac, Farbe/Attribut xr,
                                             Spalte yr
03177  c724
03178  c724  ad 26 0a        lda blnon       Cursor-Modus 40-Zeichen
03179  c727  49 80           eor #%10000000
03180  c729  8d 26 0a        sta blnon       Cursor-Modus 40-Zeichen
03181  c72c
03182  c72c  60        setc42 rts
03183  c72d
03184  c72d
03185  c72d  ===> BS-Ausgabe ac <===
03186  c72d
03187  c72d  85 ef     print  sta datax       akt. Ausgabezeichen
03188  c72f  48               pha
03189  c730
03190  c730  8a               txa
03191  c731  48               pha
03192  c732
03193  c732  98               tya
03194  c733  48               pha
03195  c734
03196  c734
03197  c734  ===> Warteschleife CTRL s/NO SCROLL <===
03198  c734
03199  c734  ad 21 0a  prtx1  lda pause       Flag für CTRL s/ NO SCROLL
03200  c737  d0 fb            bne prtx1       Warteschleife CTRL s/NO SCROLL
03201  c739
03202  c739  85 d6            sta crsw        Flag Warten Tast.eingabe
03203  c73b  a9 c3           lda #>loop2-1   Ende Zeichenausgabe
03204  c73d  48               pha
03205  c73e  a9 0b           lda #<loop2-1   Ende Zeichenausgabe
```

```
03206  c740  48              pha
03207  c741
03208  c741  a4 ec           ldy pntr        akt. Cursor-Spalte
03209  c743  a5 ef           lda datax       akt. Ausgabezeichen
03210  c745
03211  c745  c9 0d           cmp #cr         Carriage Return
03212  c747  f0 26           beq prtcr       Carriage Return ($0d oder $8d)
03213  c749
03214  c749  c9 8d           cmp #sftcr      Shift Carriage Return
03215  c74b  f0 22           beq prtcr       Carriage Return ($0d oder $8d)
03216  c74d
03217  c74d  a6 f0           ldx lstchr      letztes Ausgabezeichen
03218  c74f  e0 1b           cpx #esc        escape
03219  c751  d0 03           bne prtx2       Bearbeitung nicht-/shift Tasten
03220  c753
03221  c753  ===> ESC-Funktion bearbeiten <===
03222  c753
03223  c753  4c be c9        jmp escchr      Bearbeitung ESC-Funktion (Vector)
03224  c756
03225  c756
03226  c756  ===> Bearbeitung nicht-/shift Tasten <===
03227  c756
03228  c756  aa        prtx2 tax             akt. Ausgabezeichen
03229  c757  10 03           bpl prtx3       nicht geshiftet
03230  c759
03231  c759  4c 02 c8        jmp shfchr      Bearbeitung geshifteter Tasten
                                             (Sprungverteiler)
03232  c75c
03233  c75c
03234  c75c  c9 20     prtx3 cmp #$20
03235  c75e  90 56           bcc ctlchr      CTRL-Tasten bearbeiten (Sprung über
                                             Vector)
03236  c760
03237  c760  c9 60           cmp #$60
03238  c762  90 03           bcc njt10
03239  c764
03240  c764  29 df           and #%11011111
03241  c766  2c              .byte skip2
03242  c767  29 3f     njt10 and #%00111111
03243  c769  20 ff c2        jsr qtswc       QUOTE-Modus Flag '"' invertieren
03244  c76c  4c 22 c3        jmp nxt3
03245  c76f
03246  c76f
03247  c76f  ===> Carriage Return ($0d oder $8d) <===
03248  c76f
03249  c76f  20 c3 cb  prtcr jsr fndend      Cursor auf Ende (Doppel-) Zeile
03250  c772  e8              inx             nächste BS-Zeile
03251  c773  20 85 cb        jsr clrbit      Zeile als Normalzeile eintragen
03252  c776
03253  c776  a4 e6           ldy sclf        BS-Fenster: Spalte linker Rand
03254  c778  84 ec           sty pntr        akt. Cursor-Spalte
03255  c77a
03256  c77a  20 63 c3        jsr nxln        Cursor in nächste Zeile/Test/Scroll
03257  c77d
03258  c77d
03259  c77d  ===> INS-, QUOTE-, RVS-, BLINK-, Unterstreich-Modi aus (ESC o) <===
03260  c77d
03261  c77d  a5 f1     toqm  lda color       akt. Attribut
```

```
03262  c77f  29 cf           and #%11001111   Unterstreichen-, BLINK-Modi aus
03263  c781  85 f1           sta color        akt. Attribut
03264  c783
03265  c783  a9 00           lda #0
03266  c785  85 f5           sta insrt        Zähler für einzufügende Zeichen
03267  c787  85 f3           sta rvs          Flag für REVERS-Modus
03268  c789  85 f4           sta qtsw         Flag für QUOTE-Modus
03269  c78b
03270  c78b  60              rts
03271  c78c
03272  c78c
03273  c78c  ===> Tab. der gültigen CTRL-Tasten <===
03274  c78c
03275  c78c  02        ctltbs .byte 2         CTRL b
03276  c78d  07               .byte 7         CTRL g
03277  c78e  09               .byte 9         CTRL i
03278  c78f  0a               .byte 10        CTRL j
03279  c790  0b               .byte 11        CTRL k
03280  c791  0c               .byte 12        CTRL l
03281  c792  0e               .byte 14        CTRL n
03282  c793  0f               .byte 15        CTRL o
03283  c794  11               .byte 17        CTRL q
03284  c795  12               .byte 18        CTRL r
03285  c796  13               .byte 19        CTRL s
03286  c797  14               .byte 20        CTRL t
03287  c798  18               .byte 24        CTRL x
03288  c799  1d               .byte $1d       CTRL Cursor rechts
03289  c79a
03290  c79a
03291  c79a  ===> Adreßtab. der CTRL-Routinen <===
03292  c79a
03293  c79a  c6 c8     ctltbe .word undron-1  Unterstreichen ein (CTRL b)
03294  c79c  8d c9            .word beep-1    Glocke anschlagen (CTRL g)
03295  c79e  4e c9            .word dotab-1   Tabulator-Taste ausführen (CTRL i)
03296  c7a0  b0 c9            .word lfeed-1   Line-Feed ausführen (CTRL j)
03297  c7a2  a5 c8            .word lock-1    shift-CBM-Taste zulassen (CTRL k)
03298  c7a4  ab c8            .word unlock-1  shift-CBM-Taste sperren (CTRL l)
03299  c7a6  7f c8            .word lower-1   Klein-/Groß-Modus ein (CTRL n)
03300  c7a8  d4 c8            .word flshon-1  Blink-Modus ein (CTRL o)
03301  c7aa  59 c8            .word crsrdn-1  Cursor abwärts (CTRL q)
03302  c7ac  c1 c8            .word rvson-1   REVERS-Modus ein (CTRL r)
03303  c7ae  b2 c8            .word homtst-1  Cursor HOME, BS-Fenster rücksetzen
03304  c7b0  1a c9            .word delete-1  Zeichen löschen (CTRL t)
03305  c7b2  60 c9            .word settab-1  Tabulator setzen/löschen (CTRL x)
03306  c7b4  53 c8            .word crsrrt-1  Cursor rechts (CTRL Cursor rechts)
03307  c7b6
03308  c7b6
03309  c7b6  6c 34 03  ctlchr jmp (ctlvec)    Standard: Sprung nach contrl ($c7b9)
03310  c7b9
03311  c7b9
03312  c7b9  ===> CTRL-Tasten bearbeiten <===
03313  c7b9
03314  c7b9  c9 1b     contrl cmp #esc
03315  c7bb  f0 38            beq ctlrts      Ausgabe Code für ESC
03316  c7bd
03317  c7bd  a6 f5            ldx insrt       Zähler für einzufügende Zeichen
03318  c7bf  d0 08            bne dctrl1      Steuerzeichen REVERS bei
                                              INSERT-/QUOTE-Mod.
```

```
03319  c7c1
03320  c7c1  c9 14           cmp #$14          CTRL t
03321  c7c3  f0 0b           beq dctrl2        Tabulator setzen/löschen (auch im
                                                QUOTE-Modus)
03322  c7c5
03323  c7c5  a6 f4           ldx qtsw          Flag für QUOTE-Modus
03324  c7c7  f0 07           beq dctrl2        Test auf gültige CTRL-Tasten
03325  c7c9
03326  c7c9
03327  c7c9  ===> Steuerzeichen REVERS bei INSERT-/QUOTE-Mod. <===
03328  c7c9
03329  c7c9  a2 00     dctrl1 ldx #0
03330  c7cb  86 ef            stx datax        akt. Ausgabezeichen
03331  c7cd
03332  c7cd  4c 26 c3         jmp nc3          REVERS-Bit setzen
03333  c7d0
03334  c7d0
03335  c7d0  ===> Test auf gültige CTRL-Tasten <===
03336  c7d0
03337  c7d0  a2 0d     dctrl2 ldx #13
03338  c7d2  dd 8c c7  dctrl3 cmp ctltbs,x     Tab. der gültigen CTRL-Tasten
03339  c7d5  f0 1f            beq ctlexe       CTRL-Routine über Stapel aufrufen
03340  c7d7
03341  c7d7  ca               dex
03342  c7d8  10 f8            bpl dctrl3
03343  c7da
03344  c7da
03345  c7da  ===> Test auf Farb-Umschaltung (CTRL Ziffer) <===
03346  c7da
03347  c7da  a2 0f     colchk ldx #15
03348  c7dc  dd 4c ce  dctrl5 cmp coltab,x     Tab. der Farbtasten
03349  c7df  f0 04            beq dctrl6       Farb-Umschaltung durchführen
03350  c7e1
03351  c7e1  ca               dex
03352  c7e2  10 f8            bpl dctrl5
03353  c7e4
03354  c7e4  60               rts
03355  c7e5
03356  c7e5
03357  c7e5  ===> Farb-Umschaltung durchführen <===
03358  c7e5
03359  c7e5  24 d7     dctrl6 bit mode          BS-Modus 40/80-Zeichen
03360  c7e7  30 03            bmi dctrl7       80-Zeichen BS
03361  c7e9
03362  c7e9  ===> 40-Zeichen BS <===
03363  c7e9
03364  c7e9  86 f1            stx color        akt. Attribut
03365  c7eb  60               rts
03366  c7ec
03367  c7ec
03368  c7ec  ===> 80-Zeichen BS <===
03369  c7ec
03370  c7ec  a5 f1     dctrl7 lda color        akt. Attribut
03371  c7ee  29 f0            and #%11110000
03372  c7f0  1d 5c ce         ora coladj,x     Tab. der Farbwerte 80-Zeichen BS
03373  c7f3  85 f1            sta color        akt. Attribut
03374  c7f5
03375  c7f5  60        ctlrts rts
```

```
03376  c7f6
03377  c7f6
03378  c7f6  ===> CTRL-Routine über Stapel aufrufen <===
03379  c7f6
03380  c7f6  8a          ctlexe txa
03381  c7f7  0a                 asl a
03382  c7f8  aa                 tax
03383  c7f9  bd 9b c7           lda ctltbe+1,x   Adreßtab. der CTRL-Routinen
03384  c7fc  48                 pha
03385  c7fd
03386  c7fd  bd 9a c7           lda ctltbe,x     Adreßtab. der CTRL-Routinen
03387  c800  48                 pha
03388  c801
03389  c801  60                 rts
03390  c802
03391  c802
03392  c802  ===> Bearbeitung geshifteter Tasten (üb. Vector) <===
03393  c802
03394  c802  6c 36 03   shfchr jmp (shfvec)     Standard: Sprung nach shiftd ($c805)
03395  c805
03396  c805
03397  c805  ===> Test auf geshiftete Steuerzeichen <===
03398  c805
03399  c805  29 7f      shiftd and #%01111111   Bit 7 löschen
03400  c807  c9 20             cmp #$20
03401  c809  90 09             bcc nxtx2       shift-Steuerzeichen
03402  c80b
03403  c80b  c9 7f             cmp #$7f        vergleichen mit Token für 'pi' ($ff)
03404  c80d  d0 02             bne nxtx1
03405  c80f
03406  c80f  a9 5e             lda #$5e        ASC für 'pi'
03407  c811  4c 20 c3   nxtx1  jmp nxt33       Ausgabe shift-Zeichen (ac) auf BS
03408  c814
03409  c814
03410  c814  a6 f4      nxtx2  ldx qtsw        Flag für QUOTE-Modus
03411  c816  f0 05             beq cnc3x       QUOTE-Modus aus
03412  c818
03413  c818  09 40      nxtx3  ora #%01000000
03414  c81a  4c 26 c3           jmp nc3         REVERS-Bit setzen
03415  c81d
03416  c81d
03417  c81d  c9 14      cnc3x  cmp #$14        Test auf Steuerzeichen für 'insert'
03418  c81f  d0 03             bne ntcn1       Test suf weitere Steuerzeichen
03419  c821
03420  c821  4c e3 c8           jmp insert      Ausführen 'INSERT'
03421  c824
03422  c824
03423  c824  a6 f5      ntcn1  ldx insrt        Zähler für einzufügende Zeichen
03424  c826  d0 f0             bne nxtx3       INSERT-Modus, Steuerzeichen RVS
03425  c828
03426  c828  c9 11             cmp #$11        Test auf 'cursor up'
03427  c82a  f0 3b             beq crsrup      Cursor aufwärts
03428  c82c
03429  c82c  c9 1d             cmp #$1d        Test auf 'cursor left'
03430  c82e  f0 45             beq crsrlf      Cursor nach links
03431  c830
03432  c830  c9 0e             cmp #$0e        Test auf 'Graphik-Modus ein'
03433  c832  f0 5e             beq upper       Groß-/Graphik-Modus ein
```

```
03434  c834
03435  c834  c9 12           cmp #$12        Test auf 'revers off'
03436  c836  d0 03           bne nxtx4       weitere Steuerzeichen
03437  c838
03438  c838  4c bf c8        jmp rvsoff      REVERS-Modus aus
03439  c83b
03440  c83b
03441  c83b  c9 02     nxtx4  cmp #$02        Test auf 'underline off'
03442  c83d  d0 03           bne nxtx5       weitere Steuerzeichen
03443  c83f
03444  c83f  4c ce c8        jmp undrof      Unterstreichen aus
03445  c842
03446  c842
03447  c842  c9 0f     nxtx5  cmp #$0f        Test auf 'blink off'
03448  c844  d0 03           bne nxtx6       weitere Steuerzeichen
03449  c846
03450  c846  4c dc c8        jmp flshof      Blink-Modus aus
03451  c849
03452  c849
03453  c849  c9 13     nxtx6  cmp #$13        Test auf 'screen clear'
03454  c84b  d0 03           bne nxtx7       weitere Steuerzeichen
03455  c84d
03456  c84d  4c 42 c1        jmp clsr        BS löschen
03457  c850
03458  c850
03459  c850  09 80     nxtx7  ora #%10000000  Bit 7 wieder setzen
03460  c852  d0 86           bne colchk      immer Test auf Farbumschaltung
03461  c854
03462  c854
03463  c854                  .end
03464  c854                  .lib c128.keyboard2
03465  c854
03467  c854
03468  c854
03469  c854  ===> Cursor rechts (CTRL Cursor rechts) <===
03470  c854
03471  c854  20 ed cb  crsrrt jsr nxtchr      Cursor rechts, carry=1 neue Zeile
03472  c857  b0 04           bcs crsrw1
03473  c859
03474  c859  60               rts
03475  c85a
03476  c85a
03477  c85a  ===> Cursor abwärts (CTRL q) <===
03478  c85a
03479  c85a  20 63 c3  crsrdn jsr nxln        Cursor in nächste Zeile/Test/Scroll
03480  c85d  20 74 cb  crsrw1 jsr getbit      Test akt. Zeile (Doppelz: carry=1)
03481  c860  b0 03           bcs crsrok
03482  c862
03483  c862  38               sec
03484  c863  66 e8            ror lsxp        Cursor-Zeile vor input
03485  c865  18        crsrok clc
03486  c866  60        crsrts rts
03487  c867
03488  c867
03489  c867  ===> Cursor aufwärts <===
03490  c867
03491  c867  a6 e5     crsrup ldx sctop       BS-Fenster: oberste Zeile
03492  c869  e4 eb            cpx tblx        akt. Cursor-Zeile
```

```
03493  c86b  b0 f9            bcs crsrts
03494  c86d
03495  c86d  20 5d c8  crsrw2 jsr crsrw1
03496  c870  c6 eb            dec tblx          akt. Cursor-Zeile
03497  c872  4c 5c c1          jmp stupt         pnt auf Adresse akt. BS-Zeile
                                                 (Cursor)
03498  c875
03499  c875
03500  c875  ===> Cursor nach links <===
03501  c875
03502  c875  20 00 cc  crsrlf jsr bakchr        Cursor links, carry=1 nicht möglich
03503  c878  b0 ec            bcs crsrts
03504  c87a  d0 e9            bne crsrok
03505  c87c
03506  c87c  e6 eb            inc tblx          akt. Cursor-Zeile
03507  c87e  d0 ed            bne crsrw2
03508  c880
03509  c880
03510  c880  ===> Klein-/Groß-Modus ein (CTRL n) <===
03511  c880
03512  c880  24 d7      lower bit mode          BS-Modus 40/80-Zeichen
03513  c882  30 07            bmi low80         80-Zeichen BS
03514  c884
03515  c884  ===> 40-Zeichen BS <===
03516  c884
03517  c884  ad 2c 0a         lda vm1           VIC Text-Basiszeiger
03518  c887  09 02            ora #%00000010
03519  c889  d0 10            bne uprts
03520  c88b
03521  c88b
03522  c88b
03523  c88b  ===> 80-Zeichen BS <===
03524  c88b
03525  c88b  a5 f1      low80 lda color         akt. Attribut
03526  c88d  09 80            ora #%10000000
03527  c88f  85 f1            sta color         akt. Attribut
03528  c891
03529  c891  60               rts
03530  c892
03531  c892
03532  c892  ===> Groß-/Graphik-Modus ein <===
03533  c892
03534  c892  24 d7      upper bit mode          BS-Modus 40/80-Zeichen
03535  c894  30 09            bmi upp80         80-Zeichen BS
03536  c896
03537  c896  ===> 40-Zeichen BS <===
03538  c896
03539  c896  ad 2c 0a         lda vm1           VIC Text-Basiszeiger
03540  c899  29 fd            and #%11111101
03541  c89b  8d 2c 0a  uprts  sta vm1           VIC Text-Basiszeiger
03542  c89e
03543  c89e  60               rts
03544  c89f
03545  c89f
03546  c89f  ===> 80-Zeichen BS <===
03547  c89f
03548  c89f  a5 f1      upp80 lda color         akt. Attribut
03549  c8a1  29 7f            and #%01111111
```

```
03550  c8a3  85 f1          sta color       akt. Attribut
03551  c8a5
03552  c8a5  60            rts
03553  c8a6
03554  c8a6
03555  c8a6  ===> shift-CBM-Taste zulassen (CTRL k) <===
03556  c8a6
03557  c8a6  a9 80     lock  lda #%10000000
03558  c8a8  05 f7           ora locks       Flag enable/disable Shift/CBM-Taste
03559  c8aa  30 04           bmi unlrts      immer
03560  c8ac
03561  c8ac
03562  c8ac  ===> shift-CBM-Taste sperren (CTRL l) <===
03563  c8ac
03564  c8ac  a9 7f     unlock lda #%01111111
03565  c8ae  25 f7           and locks       Flag enable/disable Shift/CBM-Taste
03566  c8b0  85 f7     unlrts sta locks       Flag enable/disable Shift/CBM-Taste
03567  c8b2
03568  c8b2  60            rts
03569  c8b3
03570  c8b3
03571  c8b3  ===> Cursor HOME, BS-Fenster rücksetzen <===
03572  c8b3
03573  c8b3  a5 f0     homtst lda lstchr      letztes Ausgabezeichen
03574  c8b5  c9 13           cmp #$13        Code für 'HOME'
03575  c8b7  d0 03           bne home1
03576  c8b9
03577  c8b9  ===> HOME-HOME: BS-Fenster volle Größe <===
03578  c8b9
03579  c8b9  20 24 ca        jsr sreset      BS-Fenster auf volle Größe setzen
03580  c8bc  4c 50 c1  home1 jmp home        Cursor auf HOME-Pos.
03581  c8bf
03582  c8bf
03583  c8bf  ===> REVERS-Modus aus <===
03584  c8bf
03585  c8bf  a9 00     rvsoff lda #%00000000
03586  c8c1  2c              .byte skip2
03587  c8c2
03588  c8c2
03589  c8c2  ===> REVERS-Modus ein (CTRL r) <===
03590  c8c2
03591  c8c2  a9 80     rvson  lda #%10000000
03592  c8c4  85 f3           sta rvs         Flag für REVERS-Modus
03593  c8c6
03594  c8c6  60            rts
03595  c8c7
03596  c8c7
03597  c8c7  ===> Unterstreichen ein (CTRL b) <===
03598  c8c7
03599  c8c7  a5 f1     undron lda color      akt. Attribut
03600  c8c9  09 20           ora #%00100000
03601  c8cb  85 f1           sta color       akt. Attribut
03602  c8cd
03603  c8cd  60            rts
03604  c8ce
03605  c8ce
```

```
03606  c8ce  ===> Unterstreichen aus <===
03607  c8ce
03608  c8ce  a5 f1      undrof lda color        akt. Attribut
03609  c8d0  29 df             and #%11011111
03610  c8d2  85 f1             sta color        akt. Attribut
03611  c8d4
03612  c8d4  60                rts
03613  c8d5
03614  c8d5
03615  c8d5  ===> Blink-Modus ein (CTRL o) <===
03616  c8d5
03617  c8d5  a5 f1      flshon lda color        akt. Attribut
03618  c8d7  09 10             ora #%00010000
03619  c8d9  85 f1             sta color        akt. Attribut
03620  c8db
03621  c8db  60                rts
03622  c8dc
03623  c8dc
03624  c8dc  ===> Blink-Modus aus <===
03625  c8dc
03626  c8dc  a5 f1      flshof lda color        akt. Attribut
03627  c8de  29 ef             and #%11101111
03628  c8e0  85 f1             sta color        akt. Attribut
03629  c8e2
03630  c8e2  60                rts
03631  c8e3
03632  c8e3
03633  c8e3  ===> Ausführen 'INSERT' <===
03634  c8e3
03635  c8e3  20 1e cc   insert jsr savpos       Cursorpos. nach keybnk/keytmp
03636  c8e6  20 c3 cb          jsr fndend       Cursor auf Ende (Doppel-) Zeile
03637  c8e9  e4 df             cpx keytmp
03638  c8eb  d0 02             bne ins10
03639  c8ed
03640  c8ed  c4 de             cpy keybnk
03641  c8ef  90 21      ins10  bcc ins50
03642  c8f1
03643  c8f1  20 3e c3          jsr movchr       Test auf Einfüge-/Doppelzeile/Scroll
03644  c8f4  b0 22             bcs insout
03645  c8f6
03646  c8f6  20 00 cc   ins30  jsr bakchr       Cursor links, carry=1 nicht möglich
03647  c8f9  20 58 cb          jsr get1ch       Zeichen + Attribut aus BS holen
03648  c8fc  20 ed cb          jsr nxtchr       Cursor rechts, carry=1 neue Zeile
03649  c8ff  20 32 cc          jsr displt       BS-Ausg. ac temp. Farbe/Attribut
03650  c902  20 00 cc          jsr bakchr       Cursor links, carry=1 nicht möglich
03651  c905
03652  c905  a6 eb             ldx tblx         akt. Cursor-Zeile
03653  c907  e4 df             cpx keytmp       Zeile vor INS
03654  c909  d0 eb             bne ins30
03655  c90b
03656  c90b  c4 de             cpy keybnk       Spalte vor INS
03657  c90d  d0 e7             bne ins30
03658  c90f
03659  c90f  20 27 cc          jsr blank        BS-Ausgabe Space, Attribute rücks.
03660  c912  e6 f5      ins50  inc insrt        Zähler für einzufügende Zeichen
03661  c914  d0 02             bne insout
03662  c916
03663  c916  255 einzuschiebende Zeichen nicht überschreiten
```

```
03664  c916
03665  c916  c6 f5           dec insrt       Zähler für einzufügende Zeichen
03666  c918  4c 32 c9  insout jmp delout      Cursorpos. auf keybnk/keytmp
03667  c91b
03668  c91b
03669  c91b  ===> Zeichen löschen (CTRL t) <===
03670  c91b
03671  c91b  20 75 c8  delete jsr crsrlf      Cursor nach links
03672  c91e  20 1e cc         jsr savpos      Cursorpos. nach keybnk/keytmp
03673  c921  b0 0f            bcs delout      Cursorpos. auf keybnk/keytmp
03674  c923
03675  c923  c4 e7      deloop cpy scrt       BS-Fenster: Spalte rechter Rand
03676  c925  90 16            bcc delchr      Zeichen in BS-Zeile löschen
03677  c927
03678  c927  a6 eb            ldx tblx        akt. Cursor-Zeile
03679  c929  e8               inx
03680  c92a  20 76 cb         jsr getbt1      Test Zeile xr (Doppelz.: carry=1)
03681  c92d  b0 0e            bcs delchr      Zeichen in BS-Zeile löschen
03682  c92f
03683  c92f  20 27 cc         jsr blank       BS-Ausgabe Space, Attribute rücks.
03684  c932
03685  c932
03686  c932  ===> Cursorpos. auf keybnk/keytmp <===
03687  c932
03688  c932  a5 de      delout lda keybnk
03689  c934  85 ec            sta pntr        akt. Cursor-Spalte
03690  c936
03691  c936  a5 df            lda keytmp
03692  c938  85 eb            sta tblx        akt. Cursor-Zeile
03693  c93a
03694  c93a  4c 5c c1         jmp stupt       pnt auf Adresse akt. BS-Zeile
                                              (Cursor)
03695  c93d
03696  c93d
03697  c93d
03698  c93d  ===> Zeichen in BS-Zeile löschen <===
03699  c93d
03700  c93d  20 ed cb  delchr jsr nxtchr      Cursor rechts, carry=1 neue Zeile
03701  c940  20 58 cb         jsr get1ch      Zeichen + Attribut aus BS holen
03702  c943  20 00 cc         jsr bakchr      Cursor links, carry=1 nicht möglich
03703  c946  20 32 cc         jsr displt      BS-Ausg. ac temp. Farbe/Attribut
03704  c949  20 ed cb         jsr nxtchr      Cursor rechts, carry=1 neue Zeile
03705  c94c  4c 23 c9         jmp deloop
03706  c94f
03707  c94f
03708  c94f  ===> Tabulator-Taste ausführen (CTRL i) <===
03709  c94f
03710  c94f  a4 ec      dotab  ldy pntr        akt. Cursor-Spalte
03711  c951
03712  c951
03713  c951  ===> Spalte mit TAB-Marke suchen <===
03714  c951
03715  c951  c8        dotab1 iny
03716  c952  c4 e7            cpy scrt        BS-Fenster: Spalte rechter Rand
03717  c954  b0 06            bcs dotab2      Cursor auf rechten BS-Rand
03718  c956
03719  c956  20 6c c9         jsr tabget      Test auf Tabulatormarke
03720  c959  f0 f6            beq dotab1      Spalte mit TAB-Marke suchen
```

```
03721  c95b
03722  c95b  ===> Cursor auf Spalte mit TAB.marke <===
03723  c95b
03724  c95b  2c              .byte skip2
03725  c95c
03726  c95c
03727  c95c  ===> Cursor auf rechten BS-Rand <===
03728  c95c
03729  c95c  a4 e7      dotab2 ldy scrt         BS-Fenster: Spalte rechter Rand
03730  c95e  84 ec             sty pntr         akt. Cursor-Spalte
03731  c960
03732  c960  60                rts
03733  c961
03734  c961
03735  c961  ===> Tabulator setzen/löschen (CTRL x) <===
03736  c961
03737  c961  a4 ec      settab ldy pntr         akt. Cursor-Spalte
03738  c963  20 6c c9          jsr tabget       Test auf Tabulatormarke
03739  c966  45 da             eor sedsal       Tabulator-Bit flippen
03740  c968  9d 54 03          sta tabmap,x     Tab. der akt. Tabulator-Pos.
03741  c96b
03742  c96b  60                rts
03743  c96c
03744  c96c  Die Tabulator Bit-Tabelle besteht aus 10 Bytes zu 8 Bit. Damit
03745  c96c  kann für jede der max. 80 Bildschirmspalten einer Zeile durch
03746  c96c  Setzen ihres Bits eine Tabulatormarke gesetzt werden. Das Bit
03747  c96c  für Spalte 25 ist das 25. Bit, also Bit 7 des 4. Bytes:
03748  c96c
03749  c96c         00000000 00000000 00000000 10000000 .......
03750  c96c
03751  c96c
03752  c96c
03753  c96c
03754  c96c  ===> Test auf Tabulatormarke <===
03755  c96c
03756  c96c  98          tabget tya
03757  c96d  29 07              and #%00000111   Bit-Position 0-7
03758  c96f  aa                 tax
03759  c970  bd 6c ce           lda bits,x       Tabelle der Bit-Masken
03760  c973  85 da              sta sedsal       Bitmaske
03761  c975
03762  c975  98                 tya
03763  c976  4a                 lsr a
03764  c977  4a                 lsr a
03765  c978  4a                 lsr a
03766  c979  aa                 tax              Zeiger auf Tabulator-Byte
03767  c97a  bd 54 03           lda tabmap,x     berechnetes Tabulator-Byte
03768  c97d  24 da              bit sedsal       Bit-Test gegen Bitmaske
03769  c97f  60                 rts
03770  c980
03771  c980
03772  c980  ===> alle Tabulatoren löschen (ESC z) <===
03773  c980
03774  c980  a9 00       tabclr lda #%00000000
03775  c982  2c                 .byte skip2
03776  c983
03777  c983
```

```
03778  c983  ===> Standard-Tabulatoren setzen (ESC y) <===
03779  c983
03780  c983  a9 80      stdeft lda #%10000000   Bit für jede 8. Spalte setzen
03781  c985
03782  c985  a2 09             ldx #9
03783  c987  9d 54 03   stdft1 sta tabmap,x     Tab. der akt. Tabulator-Pos.
03784  c98a  ca                dex
03785  c98b  10 fa             bpl stdft1
03786  c98d
03787  c98d  60         stdft2 rts
03788  c98e
03789  c98e
03790  c98e  ===> Glocke anschlagen (CTRL g) <===
03791  c98e
03792  c98e  24 f9      beep   bit beeper       Flag enable/disable beep
03793  c990  30 fb             bmi stdft2       Glocke abgeschaltet
03794  c992
03795  c992  ===> SID für BEEP programmieren <===
03796  c992
03797  c992  a9 15             lda #21
03798  c994  8d 18 d4          sta sidvol       SID 6581 Modus/Lautstärke-Register
03799  c997
03800  c997  a0 09             ldy #9
03801  c999  a2 00             ldx #0
03802  c99b  8c 05 d4          sty sidad1       SID 6581 Attack/Decay-Register 1
03803  c99e
03804  c99e  8e 06 d4          stx sidsr1       SID 6581 Sustain/Release-Register 1
03805  c9a1
03806  c9a1  a9 30             lda #48
03807  c9a3  8d 01 d4          sta sidf2h       SID 6581 Frequenz 1 high
03808  c9a6
03809  c9a6  a9 20             lda #32
03810  c9a8  8d 04 d4          sta sidct1       SID 6581 Control-Register 1
03811  c9ab
03812  c9ab  a9 21             lda #33
03813  c9ad  8d 04 d4          sta sidct1       SID 6581 Control-Register 1
03814  c9b0
03815  c9b0  60                rts
03816  c9b1
03817  c9b1
03818  c9b1  ===> Line-Feed ausführen (CTRL j) <===
03819  c9b1
03820  c9b1  a5 ec      lfeed  lda pntr         akt. Cursor-Spalte
03821  c9b3  48                pha
03822  c9b4
03823  c9b4  20 c3 cb          jsr fndend       Cursor auf Ende (Doppel-) Zeile
03824  c9b7  20 63 c3          jsr nxln         Cursor in nächste Zeile/Test/Scroll
03825  c9ba
03826  c9ba  68                pla
03827  c9bb  85 ec             sta pntr         akt. Cursor-Spalte
03828  c9bd
03829  c9bd  60                rts
03830  c9be
03831  c9be
03832  c9be  6c 38 03   escchr jmp (escvec)     Standard: Sprung nach escape ($c9c1)
03833  c9c1
03834  c9c1
```

```
03835  c9c1  ===> Bearbeitung der ESC-Funktionen <===
03836  c9c1
03837  c9c1  c9 1b      escape cmp #esc
03838  c9c3  d0 05             bne esc01        ESC-Routine über Stapel aufrufen
03839  c9c5
03840  c9c5  ===> ESC esc: alle Modi rücksetzen <===
03841  c9c5
03842  c9c5  46 ef             lsr datax        Ausgabezeichen ungültig machen
03843  c9c7  4c 7d c7          jmp toqm         INS-, QUOTE-, RVS-, BLINK-,
                                                 Unterstreich-Modi aus (ESC o)
03844  c9ca
03845  c9ca
03846  c9ca  ===> ESC-Routine über Stapel aufrufen <===
03847  c9ca
03848  c9ca  29 7f      esc01  and #%01111111
03849  c9cc
03850  c9cc  38                sec
03851  c9cd  e9 40             sbc #$40
03852  c9cf
03853  c9cf  c9 1b             cmp #$1b
03854  c9d1  b0 0a             bcs escrts       größer als 'z' ($5a)
03855  c9d3
03856  c9d3  0a                asl a
03857  c9d4  aa                tax
03858  c9d5
03859  c9d5  bd df c9          lda escvct+1,x   Adreßtab. der ESC-Routinen
03860  c9d8  48                pha
03861  c9d9
03862  c9d9  bd de c9          lda escvct,x     Adreßtab. der ESC-Routinen
03863  c9dc  48                pha
03864  c9dd
03865  c9dd  60          escrts rts
03866  c9de
03867  c9de
03868  c9de
03869  c9de  ===> Adreßtab. der ESC-Routinen <===
03870  c9de
03871  c9de  9e ca       escvct .word eeos-1     BS ab Cursorpos. löschen (ESC @)
03872  c9e0  ec ca              .word auton-1     AUTO-INSERT-Modus ein (ESC a)
03873  c9e2  15 ca              .word sethtb-1    Cursorpos. = untere rechte BS-Ecke
                                                 (ESC b)
03874  c9e4  e9 ca              .word autoff-1    AUTO-INSERT-Modus aus (ESC c)
03875  c9e6  51 ca              .word dline-1     akt. Zeile löschen (ESC d)
03876  c9e8  0a cb              .word crsrnf-1    statischen Cursor ein (ESC e)
03877  c9ea  20 cb              .word crsrbl-1    blinkenden Cursor ein (ESC f)
03878  c9ec  36 cb              .word bellon-1    Glocke zulassen (ESC g)
03879  c9ee  39 cb              .word bellof-1    Glocke abschalten (ESC h)
03880  c9f0  3c ca              .word iline-1     Zeile an akt. Pos. einfügen (ESC i)
03881  c9f2  b0 cb              .word fndsol-1    Cursor auf Anfang (Doppel-) Zeile
03882  c9f4  51 cb              .word fndeol-1    Cursor auf Zeilenende setzen (ESC k)
03883  c9f6  e1 ca              .word scrsw0-1    BS--Scrollen ein (ESC l)
03884  c9f8  e4 ca              .word scrsw1-1    BS--Scrollen aus (ESC m)
03885  c9fa  47 cb              .word dspnml-1    INVERS-BS aus (ESC n)
03886  c9fc  7c c7              .word toqm-1      INS-, QUOTE-, RVS-, BLINK-,
                                                 Unterstreich-Modi aus (ESC o)
03887  c9fe  8a ca              .word etstol-1    Zeile Anfang - Cursor löschen (ESC
                                                 p)
03888  ca00  75 ca              .word etol-1      Zeile Cursor - Ende löschen (ESC q)
```

```
03889  ca02  3e cb            .word dsprvs-1   INVERS-BS ein (ESC r)
03890  ca04  f1 ca            .word crsral-1   Block-Cursor ein (ESC s)
03891  ca06  13 ca            .word sethtt-1   Cursorpos. = ob. li. BS-Ecke (ESC t)
03892  ca08  fd ca            .word crsrul-1   Unterstrich-Cursor ein (ESC u)
03893  ca0a  bb ca            .word suup-1     eine Zeile aufwärts scrollen (ESC v)
03894  ca0c  c9 ca            .word sddn-1     eine Zeile abwärts scrollen (ESC w)
03895  ca0e  2b cd            .word swap-1     Umschalten 40/80-Zeichen-BS (ESC x)
03896  ca10  82 c9            .word stdeft-1   Standard-Tabulatoren setzen (ESC y)
03897  ca12  7f c9            .word tabclr-1   alle Tabulatoren löschen (ESC z)
03898  ca14
03899  ca14
03900  ca14  ===> Cursorpos. = ob. li. BS-Ecke (ESC t) <===
03901  ca14
03902  ca14  18        sethtt clc
03903  ca15  24               .byte skip1
03904  ca16
03905  ca16
03906  ca16  ===> Cursorpos. = untere rechte BS-Ecke (ESC b) <===
03907  ca16
03908  ca16  38        sethtb sec
03909  ca17  a6 ec            ldx pntr         akt. Cursor-Spalte
03910  ca19  a5 eb            lda tblx         akt. Cursor-Zeile
03911  ca1b
03912  ca1b
03913  ca1b  ===> BS-Fenster definieren <===
03914  ca1b
03915  ca1b  90 11     window bcc sttop        linke (xr), obere (ac) BS-Ecke
                                               setzen
03916  ca1d
03917  ca1d
03918  ca1d  ===> rechte (xr), untere (ac) BS-Ecke setzen <===
03919  ca1d
03920  ca1d  85 e4     stbot  sta scbot        BS-Fenster: unterste Zeile
03921  ca1f  86 e7            stx scrt         BS-Fenster: Spalte rechter Rand
03922  ca21
03923  ca21  4c 32 ca         jmp rewrap       alle BS-Zeilen Normalzeilen
03924  ca24
03925  ca24
03926  ca24  ===> BS-Fenster auf volle Größe setzen <===
03927  ca24
03928  ca24  a5 ed     sreset lda lines        höchste BS-Zeile akt. Modus
03929  ca26  a6 ee            ldx colums       höchste BS-Spalte akt. Modus
03930  ca28  20 1d ca         jsr stbot        rechte (xr), untere (ac) BS-Ecke
                                               setzen
03931  ca2b
03932  ca2b  a9 00            lda #0
03933  ca2d  aa               tax
03934  ca2e
03935  ca2e
03936  ca2e  ===> linke (xr), obere (ac) BS-Ecke setzen <===
03937  ca2e
03938  ca2e  85 e5     sttop  sta sctop        BS-Fenster: oberste Zeile
03939  ca30  86 e6            stx sclf         BS-Fenster: Spalte linker Rand
03940  ca32
03941  ca32
```

```
03942  ca32  ===> alle BS-Zeilen Normalzeilen <===
03943  ca32
03944  ca32  a9 00      rewrap lda #%00000000
03945  ca34
03946  ca34  a2 04             ldx #4
03947  ca36  9d 5d 03   setal1 sta bitabl-1,x   Tab. BS-Doppelzeilen
03948  ca39  ca                dex
03949  ca3a  d0 fa             bne setal1
03950  ca3c
03951  ca3c  60                rts
03952  ca3d
03953  ca3d
03954  ca3d  ===> Zeile an akt. Pos. einfügen (ESC i) <===
03955  ca3d
03956  ca3d  20 7c c3   iline  jsr scrdwn       nach unten scrollen
03957  ca40  20 56 c1          jsr stu10        Cursor an linken Rand BS-Zeile
03958  ca43
03959  ca43  e8                inx
03960  ca44  20 76 cb          jsr getbt1       Test Zeile xr (Doppelz.: carry=1)
03961  ca47
03962  ca47  08                php
03963  ca48  20 81 cb          jsr putbit       akt. Zeile als Doppel-/Normalzeile
                                                eintragen
03964  ca4b  28                plp
03965  ca4c
03966  ca4c  b0 03             bcs linrts
03967  ca4e
03968  ca4e  38                sec
03969  ca4f  66 e8             ror lsxp         Cursor-Zeile vor input
03970  ca51  60         linrts rts
03971  ca52
03972  ca52
03973  ca52  ===> akt. Zeile löschen (ESC d) <===
03974  ca52
03975  ca52  20 b5 cb   dline  jsr fistrt       pnt auf Anfang (Doppel-) Zeile
03976  ca55
03977  ca55  a5 e5             lda sctop        BS-Fenster: oberste Zeile
03978  ca57  48                pha
03979  ca58
03980  ca58  a5 eb             lda tblx         akt. Cursor-Zeile
03981  ca5a  85 e5             sta sctop        BS-Fenster: oberste Zeile
03982  ca5c
03983  ca5c  a5 f8             lda scroll       scroll-disable-Flag
03984  ca5e  48                pha
03985  ca5f
03986  ca5f  a9 80             lda #%10000000
03987  ca61  85 f8             sta scroll       Bildschirm-Scroll verbieten
03988  ca63
03989  ca63  20 b8 c3          jsr scru15       BS unter gelöschter Zeile scroll up
03990  ca66
03991  ca66  68                pla
03992  ca67  85 f8             sta scroll       scroll-disable-Flag
03993  ca69
03994  ca69  a5 e5             lda sctop        BS-Fenster: oberste Zeile
03995  ca6b  85 eb             sta tblx         akt. Cursor-Zeile
03996  ca6d
03997  ca6d  68                pla
03998  ca6e  85 e5             sta sctop        BS-Fenster: oberste Zeile
```

```
03999  ca70
04000  ca70  38              sec             ungültig machen:
04001  ca71  66 e8           ror lsxp        Cursor-Zeile vor input
04002  ca73  4c 56 c1        jmp stu10       Cursor an linken Rand BS-Zeile
04003  ca76
04004  ca76
04005  ca76  ===> Zeile Cursor - Ende löschen (ESC q) <===
04006  ca76
04007  ca76  20 1e cc  etol   jsr savpos      Cursorpos. nach keybnk/keytmp
04008  ca79  20 aa c4  etol1  jsr clrprt      BS-Zeile löschen
04009  ca7c
04010  ca7c  e6 eb           inc tblx        akt. Cursor-Zeile
04011  ca7e  20 5c c1        jsr stupt       pnt auf Adresse akt. BS-Zeile
                                             (Cursor)
04012  ca81
04013  ca81  a4 e6           ldy sclf        BS-Fenster: Spalte linker Rand
04014  ca83  20 74 cb        jsr getbit      Test akt. Zeile (Doppelz: carry=1)
04015  ca86  b0 f1           bcs etol1
04016  ca88
04017  ca88  4c 32 c9  etout  jmp delout      Cursorpos. auf keybnk/keytmp
04018  ca8b
04019  ca8b
04020  ca8b  ===> Zeile Anfang - Cursor löschen (ESC p) <===
04021  ca8b
04022  ca8b  20 1e cc  etstol jsr savpos      Cursorpos. nach keybnk/keytmp
04023  ca8e  20 27 cc  etsto1 jsr blank       BS-Ausgabe Space, Attribute rücks.
04024  ca91
04025  ca91  c4 e6           cpy sclf        BS-Fenster: Spalte linker Rand
04026  ca93  d0 05           bne ets100
04027  ca95
04028  ca95  20 74 cb        jsr getbit      Test akt. Zeile (Doppelz: carry=1)
04029  ca98  90 ee           bcc etout
04030  ca9a
04031  ca9a  20 00 cc  ets100 jsr bakchr      Cursor links, carry=1 nicht möglich
04032  ca9d  90 ef           bcc etsto1
04033  ca9f
04034  ca9f
04035  ca9f
04036  ca9f  ===> BS ab Cursorpos. löschen (ESC @) <===
04037  ca9f
04038  ca9f  20 1e cc  eeos   jsr savpos      Cursorpos. nach keybnk/keytmp
04039  caa2  20 aa c4  erasf1 jsr clrprt      BS-Zeile löschen
04040  caa5
04041  caa5  e6 eb           inc tblx        akt. Cursor-Zeile
04042  caa7  20 5c c1        jsr stupt       pnt auf Adresse akt. BS-Zeile
                                             (Cursor)
04043  caaa
04044  caaa  a4 e6           ldy sclf        BS-Fenster: Spalte linker Rand
04045  caac  20 74 cb        jsr getbit      Test akt. Zeile (Doppelz: carry=1)
04046  caaf  b0 f1           bcs erasf1
04047  cab1
04048  cab1  a5 eb           lda tblx        akt. Cursor-Zeile
04049  cab3  c5 e4           cmp scbot       BS-Fenster: unterste Zeile
04050  cab5  90 eb           bcc erasf1
04051  cab7  f0 e9           beq erasf1
04052  cab9
04053  cab9  4c 32 c9        jmp delout      Cursorpos. auf keybnk/keytmp
04054  cabc
```

```
04055  cabc
04056  cabc  ===> eine Zeile aufwärts scrollen (ESC v) <===
04057  cabc
04058  cabc  20 1e cc   suup   jsr savpos       Cursorpos. nach keybnk/keytmp
04059  cabf  8a                txa
04060  cac0  48                pha
04061  cac1
04062  cac1  20 a6 c3          jsr scrup        nach oben scrollen
04063  cac4  68                pla
04064  cac5  85 df             sta keytmp
04065  cac7
04066  cac7  4c 32 c9          jmp delout       Cursorpos. auf keybnk/keytmp
04067  caca
04068  caca
04069  caca  ===> eine Zeile abwärts scrollen (ESC w) <===
04070  caca
04071  caca  20 1e cc   sddn   jsr savpos       Cursorpos. nach keybnk/keytmp
04072  cacd  20 74 cb          jsr getbit       Test akt. Zeile (Doppelz: carry=1)
04073  cad0  b0 03             bcs sddn2
04074  cad2
04075  cad2  38                sec              ungültig machen:
04076  cad3  66 e8             ror lsxp         Cursor-Zeile vor input
04077  cad5  a5 e5      sddn2  lda sctop        BS-Fenster: oberste Zeile
04078  cad7  85 eb             sta tblx         akt. Cursor-Zeile
04079  cad9
04080  cad9  20 7c c3          jsr scrdwn       nach unten scrollen
04081  cadc  20 85 cb          jsr clrbit       Zeile als Normalzeile eintragen
04082  cadf  4c 32 c9          jmp delout       Cursorpos. auf keybnk/keytmp
04083  cae2
04084  cae2
04085  cae2  ===> BS--Scrollen ein (ESC l) <===
04086  cae2
04087  cae2  a9 00      scrsw0 lda #%00000000
04088  cae4  2c                .byte skip2
04089  cae5
04090  cae5
04091  cae5
04092  cae5  ===> BS--Scrollen aus (ESC m) <===
04093  cae5
04094  cae5  a9 80      scrsw1 lda #%10000000
04095  cae7  85 f8             sta scroll       scroll-disable-Flag
04096  cae9
04097  cae9  60                rts
04098  caea
04099  caea
04100  caea  ===> AUTO-INSERT-Modus aus (ESC c) <===
04101  caea
04102  caea  a9 00      autoff lda #%00000000
04103  caec  2c                .byte skip2
04104  caed
04105  caed
04106  caed  ===> AUTO-INSERT-Modus ein (ESC a) <===
04107  caed
04108  caed  a9 ff      auton  lda #%11111111
04109  caef  85 f6             sta insflg       AUTO-INSERT-Flag
04110  caf1
04111  caf1  60                rts
04112  caf2
```

```
04113  caf2
04114  caf2  ===> Block-Cursor ein (ESC s) <===
04115  caf2
04116  caf2  24 d7       crsral bit mode          BS-Modus 40/80-Zeichen
04117  caf4  10 40              bpl crsrc         nicht möglich 40-Zeichen BS
04118  caf6
04119  caf6  ad 2b 0a           lda curmod        VDC Cursormodus
04120  caf9  29 e0              and #%11100000
04121  cafb
04122  cafb   xxx00000 Anfangszeile Cursor = 0 (oben)
04123  cafb
04124  cafb  4c 14 cb           jmp crsra         Cursormodus an VDC übergeben
04125  cafe
04126  cafe
04127  cafe  ===> Unterstrich-Cursor ein (ESC u) <===
04128  cafe
04129  cafe  24 d7       crsrul bit mode          BS-Modus 40/80-Zeichen
04130  cb00  10 34              bpl crsrc         nicht möglich 40-Zeichen BS
04131  cb02
04132  cb02  ===> 80-Zeichen BS <===
04133  cb02
04134  cb02  ad 2b 0a           lda curmod        VDC Cursormodus
04135  cb05  29 e0              and #%11100000
04136  cb07  09 07              ora #%00000111
04137  cb09
04138  cb09   xxx00111  Anfangszeile Cursor = 7 (unten)
04139  cb09
04140  cb09  d0 09              bne crsra         Cursormodus an VDC übergeben
04141  cb0b
04142  cb0b
04143  cb0b  ===> statischen Cursor ein (ESC e) <===
04144  cb0b
04145  cb0b  24 d7       crsrnf bit mode          BS-Modus 40/80-Zeichen
04146  cb0d  10 0b              bpl crsrd         40-Zeichen BS
04147  cb0f
04148  cb0f
04149  cb0f  ===> 80-Zeichen BS <===
04150  cb0f
04151  cb0f  ad 2b 0a           lda curmod        VDC Cursormodus
04152  cb12  29 1f              and #%00011111
04153  cb14
04154  cb14   x00xxxxx  Block-Cursor
04155  cb14
04156  cb14
04157  cb14  ===> Cursormodus an VDC übergeben <===
04158  cb14
04159  cb14  8d 2b 0a   crsra   sta curmod        VDC Cursormodus
04160  cb17
04161  cb17  4c 91 cd           jmp crsrdo        akt. CursorModus an VDC übergeben
04162  cb1a
04163  cb1a
04164  cb1a  ===> 40-Zeichen BS <===
04165  cb1a
04166  cb1a  ad 26 0a   crsrd   lda blnon         Cursor-Modus 40-Zeichen
04167  cb1d  09 40              ora #%01000000
04168  cb1f  d0 12              bne crsrb         immer
04169  cb21
04170  cb21
```

```
04171  cb21  ===> blinkenden Cursor ein (ESC f) <===
04172  cb21
04173  cb21  24 d7       crsrbl bit mode          BS-Modus 40/80-Zeichen
04174  cb23  10 09              bpl flsho1        40-Zeichen BS
04175  cb25
04176  cb25  ===> 80-Zeichen BS <===
04177  cb25
04178  cb25  ad 2b 0a           lda curmod        VDC Cursormodus
04179  cb28  29 1f              and #%00011111
04180  cb2a  09 60              ora #%01100000
04181  cb2c
04182  cb2c   x11xxxxx  Blink-Cursor
04183  cb2c
04184  cb2c  d0 e6              bne crsra         Cursormodus an VDC übergeben
04185  cb2e
04186  cb2e
04187  cb2e  ===> 40-Zeichen BS <===
04188  cb2e
04189  cb2e  ad 26 0a    flsho1 lda blnon         Cursor-Modus 40-Zeichen
04190  cb31  29 bf              and #%10111111
04191  cb33
04192  cb33
04193  cb33  ===> Cursormodus abspeichern <===
04194  cb33
04195  cb33  8d 26 0a    crsrb  sta blnon         Cursor-Modus 40-Zeichen
04196  cb36
04197  cb36  60          crsrc  rts
04198  cb37
04199  cb37
04200  cb37  ===> Glocke zulassen (ESC g) <===
04201  cb37
04202  cb37  a9 00       bellon lda #%00000000
04203  cb39  2c                 .byte skip2
04204  cb3a
04205  cb3a
04206  cb3a
04207  cb3a  ===> Glocke abschalten (ESC h) <===
04208  cb3a
04209  cb3a  a9 80       bellof lda #%10000000
04210  cb3c  85 f9              sta beeper        Flag enable/disable beep
04211  cb3e
04212  cb3e  60                 rts
04213  cb3f
04214  cb3f
04215  cb3f  ===> INVERS-BS ein (ESC r) <===
04216  cb3f
04217  cb3f  a2 18       dsprvs ldx #24           Register 24 wählen
04218  cb41  20 da cd           jsr vdcin         Daten vom VDC lesen, Register = xr
04219  cb44  09 40              ora #%01000000    Bit6 setzen (INVERS-BS)
04220  cb46  d0 07              bne dspout        immer
04221  cb48
04222  cb48
04223  cb48  ===> INVERS-BS aus (ESC n) <===
04224  cb48
04225  cb48  a2 18       dspnml ldx #24           ,Register 24 wählen
04226  cb4a  20 da cd           jsr vdcin         Daten vom VDC lesen, Register = xr
04227  cb4d  29 bf              and #%10111111    Bit6 rücksetzen (NORMAL-BS)
04228  cb4f  4c cc cd    dspout jmp vdcout        ac in Register xr VDC schreiben
```

```
04229  cb52
04230  cb52
04231  cb52  ===> Cursor auf Zeilenende setzen (ESC k) <===
04232  cb52
04233  cb52  20 c3 cb   fndeol jsr fndend       Cursor auf Ende (Doppel-) Zeile
04234  cb55  4c 3e c3          jmp movchr       Test auf Einfüge-/Doppelzeile/Scroll
04235  cb58
04236  cb58
04237  cb58  ===> Zeichen + Attribut aus BS holen <===
04238  cb58
04239  cb58  a4 ec      get1ch ldy pntr         akt. Cursor-Spalte
04240  cb5a
04241  cb5a  24 d7             bit mode         BS-Modus 40/80-Zeichen
04242  cb5c  30 07             bmi get80        80-Zeichen BS
04243  cb5e
04244  cb5e  ===> 40-Zeichen BS <===
04245  cb5e
04246  cb5e  b1 e2             lda (user),y     addrl Zeiger auf akt. Attributzeile
04247  cb60  85 f2             sta tcolor       temporäres Register für Attribut
04248  cb62
04249  cb62  b1 e0             lda (pnt),y      addrl Zeiger auf akt. BS-Zeile
04250  cb64  60                rts
04251  cb65
04252  cb65
04253  cb65  ===> 80-Zeichen BS <===
04254  cb65
04255  cb65  20 f9 cd   get80  jsr getusr       Attributadresse im VDC setzen
04256  cb68  20 d8 cd          jsr vdcget       Daten vom VDC über Update-Register
                                                lesen
04257  cb6b  85 f2             sta tcolor       temporäres Register für Attribut
04258  cb6d
04259  cb6d  20 e6 cd          jsr getpnt       Textadresse im VDC setzen
04260  cb70  20 d8 cd          jsr vdcget       Daten vom VDC über Update-Register
                                                lesen
04261  cb73  60                rts
04262  cb74
04263  cb74
04264  cb74  ===> Test akt. Zeile (Doppelz: carry=1) <===
04265  cb74
04266  cb74  a6 eb      getbit ldx tblx         akt. Cursor-Zeile
04267  cb76
04268  cb76
04269  cb76  ===> Test Zeile xr (Doppelz.: carry=1) <===
04270  cb76
04271  cb76  20 9f cb   getbt1 jsr bitpos       Bitmaske Zeile xr nach ac, Zeiger
                                                nach xr
04272  cb79  3d 5e 03          and bitabl,x     Tab. BS-Doppelzeilen
04273  cb7c  c9 01             cmp #%00000001   carry=1, wenn Bit gesetzt
04274  cb7e  4c 90 cb          jmp bitout       Abschluß, Zeilennummer nach xr
04275  cb81
04276  cb81
04277  cb81  ===> akt. Zeile als Doppel-/Normalzeile eintragen <===
04278  cb81
04279  cb81  a6 eb      putbit ldx tblx         akt. Cursor-Zeile
04280  cb83
04281  cb83
04282  cb83  ===> Zeile in xr als Doppel-/Normalzeile eintragen <===
04283  cb83
```

```
04284  cb83  b0 0e      putbt1 bcs setbit       Zeile als Doppelzeile eintragen
04285  cb85
04286  cb85
04287  cb85  ===> Zeile als Normalzeile eintragen <===
04288  cb85
04289  cb85  20 9f cb   clrbit jsr bitpos       Bitmaske Zeile xr nach ac, Zeiger
                                                nach xr
04290  cb88  49 ff             eor #%11111111   gesetztes Bit löschen
04291  cb8a  3d 5e 03          and bitabl,x     Tab. BS-Doppelzeilen
04292  cb8d
04293  cb8d
04294  cb8d  ===> Zeilenstatus abspeichern <===
04295  cb8d
04296  cb8d  9d 5e 03   bitsav sta bitabl,x     Tab. BS-Doppelzeilen
04297  cb90
04298  cb90
04299  cb90  ===> Abschluß, Zeilennummer nach xr <===
04300  cb90
04301  cb90  a6 da      bitout ldx sedsal       Nummer Bildschirmzeile laden
04302  cb92  60                rts
04303  cb93
04304  cb93
04305  cb93  ===> Zeile als Doppelzeile eintragen <===
04306  cb93
04307  cb93  24 f8      setbit bit scroll       scroll-disable-Flag
04308  cb95  70 df             bvs getbt1       Doppelzeilen nicht erlaubt
04309  cb97
04310  cb97  20 9f cb          jsr bitpos       Bitmaske Zeile xr nach ac, Zeiger
                                                nach xr
04311  cb9a  1d 5e 03          ora bitabl,x     Tab. BS-Doppelzeilen
04312  cb9d  d0 ee             bne bitsav       Zeilenstatus abspeichern
04313  cb9f
04314  cb9f
04315  cb9f  Die Tabelle der Bildschirm-Doppelzeilen besteht aus 4 Bytes
04316  cb9f  zu 8 Bit. Jede der 25 möglichen Bildschirmzeilen kann durch
04317  cb9f  Setzen ihres Bits zur Folgezeile erklärt werden. Das Bit für
04318  cb9f  Bildschirmzeile 25 ist das 25. Bit, also Bit 7 des 4. Bytes:
04319  cb9f
04320  cb9f          00000000 00000000 00000000 10000000
04321  cb9f
04322  cb9f
04323  cb9f  ===> Bitmaske Zeile xr nach ac, Zeiger nach xr <===
04324  cb9f
04325  cb9f  86 da      bitpos stx sedsal       Nummer Bildschirmzeile speichern
04326  cba1
04327  cba1  8a                txa
04328  cba2  29 07             and #%00000111   Bitposition
04329  cba4  aa                tax
04330  cba5
04331  cba5  bd 6c ce          lda bits,x       Bitmaske aus Tabelle
04332  cba8  48                pha
04333  cba9
04334  cba9  a5 da             lda sedsal       addrl Zeiger auf Startzeile
04335  cbab  4a                lsr a
04336  cbac  4a                lsr a
04337  cbad  4a                lsr a
04338  cbae  aa                tax              Zeiger in Byte-Tabelle
04339  cbaf
```

```
04340  cbaf  68              pla             Bitmaske
04341  cbb0  60              rts
04342  cbb1
04343  cbb1
04344  cbb1  ===> Cursor auf Anfang (Doppel-) Zeile <===
04345  cbb1
04346  cbb1  a4 e6     fndsol ldy sclf        BS-Fenster: Spalte linker Rand
04347  cbb3  84 ec            sty pntr        akt. Cursor-Spalte
04348  cbb5
04349  cbb5
04350  cbb5  ===> pnt auf Anfang (Doppel-) Zeile <===
04351  cbb5
04352  cbb5  20 74 cb  fistrt jsr getbit      Test akt. Zeile (Doppelz: carry=1)
04353  cbb8  90 06            bcc fnd0
04354  cbba
04355  cbba  c6 eb            dec tblx        akt. Cursor-Zeile
04356  cbbc  10 f7            bpl fistrt      pnt auf Anfang (Doppel-) Zeile
04357  cbbe
04358  cbbe  ===> erstes Nicht-Space von rechts suchen <===
04359  cbbe
04360  cbbe  e6 eb            inc tblx        akt. Cursor-Zeile
04361  cbc0  4c 5c c1  fnd0   jmp stupt       pnt auf Adresse akt. BS-Zeile
                                              (Cursor)
04362  cbc3
04363  cbc3
04364  cbc3  ===> Cursor auf Ende (Doppel-) Zeile <===
04365  cbc3
04366  cbc3  e6 eb     fndend inc tblx        akt. Cursor-Zeile
04367  cbc5  20 74 cb         jsr getbit      Test akt. Zeile (Doppelz: carry=1)
04368  cbc8  b0 f9            bcs fndend      Cursor auf Ende (Doppel-) Zeile
04369  cbca
04370  cbca  c6 eb            dec tblx        akt. Cursor-Zeile
04371  cbcc  20 5c c1         jsr stupt       pnt auf Adresse akt. BS-Zeile
                                              (Cursor)
04372  cbcf
04373  cbcf  a4 e7            ldy scrt        BS-Fenster: Spalte rechter Rand
04374  cbd1  84 ec            sty pntr        akt. Cursor-Spalte
04375  cbd3
04376  cbd3  20 58 cb  eloup2 jsr get1ch      Zeichen + Attribut aus BS holen
04377  cbd6  a6 eb            ldx tblx        akt. Cursor-Zeile
04378  cbd8
04379  cbd8  c9 20            cmp #' '
04380  cbda  d0 0e            bne endbye
04381  cbdc
04382  cbdc  c4 e6            cpy sclf        BS-Fenster: Spalte linker Rand
04383  cbde  d0 05            bne eloup3
04384  cbe0
04385  cbe0  20 74 cb         jsr getbit      Test akt. Zeile (Doppelz: carry=1)
04386  cbe3  90 05            bcc endbye
04387  cbe5
04388  cbe5  20 00 cc  eloup3 jsr bakchr      Cursor links, carry=1 nicht möglich
04389  cbe8  90 e9            bcc eloup2
04390  cbea
04391  cbea  84 ea     endbye sty indx        Spalte letztes Zeichen in Zeile
04392  cbec  60               rts
04393  cbed
04394  cbed
04395  cbed  ===> Cursor rechts, carry=1 neue Zeile <===
```

```
04396  cbed
04397  cbed  48         nxtchr pha
04398  cbee
04399  cbee  a4 ec             ldy pntr        akt. Cursor-Spalte
04400  cbf0  c4 e7             cpy scrt        BS-Fenster: Spalte rechter Rand
04401  cbf2  90 07             bcc bumpnt
04402  cbf4
04403  cbf4  ===> nächste BS-Zeile <===
04404  cbf4
04405  cbf4  20 63 c3          jsr nxln        Cursor in nächste Zeile/Test/Scroll
04406  cbf7  a4 e6             ldy sclf        BS-Fenster: Spalte linker Rand
04407  cbf9  88                dey
04408  cbfa  38                sec             Flag für Zeilenwechsel
04409  cbfb  c8         bumpnt iny
04410  cbfc  84 ec             sty pntr        akt. Cursor-Spalte
04411  cbfe
04412  cbfe  68                pla
04413  cbff  60                rts
04414  cc00
04415  cc00
04416  cc00  ===> Cursor links, carry=1 nicht möglich <===
04417  cc00
04418  cc00  a4 ec      bakchr ldy pntr        akt. Cursor-Spalte
04419  cc02  88                dey
04420  cc03  30 04             bmi bakot1      Test auf Zeile höher
04421  cc05
04422  cc05  c4 e6             cpy sclf        BS-Fenster: Spalte linker Rand
04423  cc07  b0 0f             bcs bakout
04424  cc09
04425  cc09
04426  cc09  ===> Test auf Zeile höher <===
04427  cc09
04428  cc09  a4 e5      bakot1 ldy sctop       BS-Fenster: oberste Zeile
04429  cc0b  c4 eb             cpy tblx        akt. Cursor-Zeile
04430  cc0d  b0 0e             bcs kabot2
04431  cc0f
04432  cc0f
04433  cc0f  ===> eine BS-Zeile höher <===
04434  cc0f
04435  cc0f  c6 eb             dec tblx        akt. Cursor-Zeile
04436  cc11  48                pha
04437  cc12
04438  cc12  20 5c c1          jsr stupt       pnt auf Adresse akt. BS-Zeile
                                               (Cursor)
04439  cc15  68                pla
04440  cc16  a4 e7             ldy scrt        BS-Fenster: Spalte rechter Rand
04441  cc18  84 ec      bakout sty pntr        akt. Cursor-Spalte
04442  cc1a
04443  cc1a  c4 e7             cpy scrt        BS-Fenster: Spalte rechter Rand
04444  cc1c  18                clc
04445  cc1d  60         kabot2 rts
04446  cc1e
04447  cc1e
04448  cc1e  ===> Cursorpos. nach keybnk/keytmp <===
04449  cc1e
04450  cc1e  a4 ec      savpos ldy pntr        akt. Cursor-Spalte
04451  cc20  84 de             sty keybnk
04452  cc22
```

```
04453  cc22  a6 eb           ldx tblx        akt. Cursor-Zeile
04454  cc24  86 df           stx keytmp
04455  cc26
04456  cc26  60              rts
04457  cc27
04458  cc27
04459  cc27                  .end
04460  cc27                  .lib c128.editor4
04461  cc27
```

4.1.10 Bildschirm-Editor (Fortsetzung)

```
04463  cc27
04464  cc27
04465  cc27  ===> BS-Ausgabe Space, Attribute rücks. <===
04466  cc27
04467  cc27  a5 f1      blank  lda color       akt. Attribut
04468  cc29  29 8f             and #%10001111  BLINK/RVS/Unterstreich aus
04469  cc2b  aa                tax
04470  cc2c
04471  cc2c  a9 20             lda #' '
04472  cc2e  2c                .byte skip2
04473  cc2f
04474  cc2f
04475  cc2f  ===> BS-Ausgabe ac mit akt. Farbe/Attribut <===
04476  cc2f
04477  cc2f  a6 f1      displc ldx color       akt. Attribut
04478  cc31  2c                .byte skip2
04479  cc32
04480  cc32
04481  cc32  ===> BS-Ausg. ac temp. Farbe/Attribut <===
04482  cc32
04483  cc32  a6 f2      displt ldx tcolor      temporäres Register für Attribut
04484  cc34
04485  cc34
04486  cc34  ===> BS-Ausgabe ac, Farbe/Attribut in xr <===
04487  cc34
04488  cc34  a8         disply tay
04489  cc35
04490  cc35  a9 02             lda #2
04491  cc37  8d 28 0a          sta curcnt     VIC Zähler für Cursor-Blinken
04492  cc3a
04493  cc3a  20 7c c1          jsr scolor     Zeiger in Farb-RAM nachführen
04494  cc3d  98                tya
04495  cc3e  a4 ec             ldy pntr       akt. Cursor-Spalte
04496  cc40
04497  cc40
04498  cc40  ===> BS-Ausgabe ac, Farbe/Attribut xr, Spalte yr <===
04499  cc40
04500  cc40  24 d7      displ  bit mode       BS-Modus 40/80-Zeichen
04501  cc42  30 06             bmi disp80     80-Zeichen BS
04502  cc44
04503  cc44  ===> 40-Zeichen BS <===
04504  cc44
04505  cc44  91 e0             sta (pnt),y    addrl Zeiger auf akt. BS-Zeile
04506  cc46  8a                txa
```

```
04507  cc47  91 e2           sta (user),y     addrl Zeiger auf akt. Attributzeile
04508  cc49
04509  cc49  60              rts
04510  cc4a
04511  cc4a
04512  cc4a  ===> 80-Zeichen BS <===
04513  cc4a
04514  cc4a  48        disp80 pha             Zeichen
04515  cc4b
04516  cc4b  8a               txa
04517  cc4c  48               pha             Attribut
04518  cc4d
04519  cc4d  20 f9 cd         jsr getusr      Attributadresse im VDC setzen
04520  cc50
04521  cc50  68               pla             Attribut
04522  cc51  20 ca cd         jsr vdcput      Daten für Block-/normales Schreiben
                                              an VDC
04523  cc54  20 e6 cd         jsr getpnt      Textadresse im VDC setzen
04524  cc57
04525  cc57  68               pla             Zeichen
04526  cc58  4c ca cd         jmp vdcput      Daten für Block-/normales Schreiben
                                              an VDC
04527  cc5b
04528  cc5b
04529  cc5b  ===> Parameter BS-Fenster lesen <===
04530  cc5b
04531  cc5b  38        scrorg sec
04532  cc5c  a5 e4            lda scbot       BS-Fenster: unterste Zeile
04533  cc5e  e5 e5            sbc sctop       BS-Fenster: oberste Zeile
04534  cc60  a8               tay             Anzahl Zeilen BS-Fenster
04535  cc61
04536  cc61  38               sec
04537  cc62  a5 e7            lda scrt        BS-Fenster: Spalte rechter Rand
04538  cc64  e5 e6            sbc sclf        BS-Fenster: Spalte linker Rand
04539  cc66  aa               tax             Anzahl Spalten BS-Fenster
04540  cc67
04541  cc67  a5 ee            lda colums      max. Spalten BS
04542  cc69  60               rts
04543  cc6a
04544  cc6a
04545  cc6a  ===> Cursor-Pos. (rel. BS-Fenster) lesen/schreiben <===
04546  cc6a
04547  cc6a  b0 29      plot  bcs doplo5       Cursor-Pos. (rel. BS-Fenster) lesen
04548  cc6c
04549  cc6c  ===> Test auf Parameter in BS-Fenster <===
04550  cc6c
04551  cc6c  8a               txa
04552  cc6d  65 e5            adc sctop       BS-Fenster: oberste Zeile
04553  cc6f  b0 14            bcs doplo3
04554  cc71
04555  cc71  c5 e4            cmp scbot       BS-Fenster: unterste Zeile
04556  cc73  f0 02            beq doplo1
04557  cc75  b0 0e            bcs doplo3
04558  cc77
04559  cc77  48        doplo1 pha
04560  cc78
04561  cc78  18               clc
04562  cc79  98               tya
```

```
04563  cc7a  65 e6          adc sclf        BS-Fenster: Spalte linker Rand
04564  cc7c  b0 06          bcs doplo2
04565  cc7e
04566  cc7e  c5 e7          cmp scrt        BS-Fenster: Spalte rechter Rand
04567  cc80  f0 04          beq doplo4      Cursor-Pos. (rel. BS-Fenster)
                                            schreiben
04568  cc82  90 02          bcc doplo4      Cursor-Pos. (rel. BS-Fenster)
                                            schreiben
04569  cc84
04570  cc84  68       doplo2 pla
04571  cc85  60       doplo3 rts
04572  cc86
04573  cc86
04574  cc86
04575  cc86  ===> Cursor-Pos. (rel. BS-Fenster) schreiben <===
04576  cc86
04577  cc86  85 ec    doplo4 sta pntr         akt. Cursor-Spalte
04578  cc88
04579  cc88  85 e9           sta lstp         Cursor-Spalte vor input
04580  cc8a
04581  cc8a  68              pla
04582  cc8b  85 eb           sta tblx         akt. Cursor-Zeile
04583  cc8d  85 e8           sta lsxp         Cursor-Zeile vor input
04584  cc8f
04585  cc8f  20 5c c1        jsr stupt        pnt auf Adresse akt. BS-Zeile
                                            (Cursor)
04586  cc92  20 57 cd        jsr cursor       Cursorpos. im 80-Zeichen BS setzen
04587  cc95
04588  cc95
04589  cc95  ===> Cursor-Pos. (rel. BS-Fenster) lesen <===
04590  cc95
04591  cc95  a5 eb    doplo5 lda tblx         akt. Cursor-Zeile
04592  cc97  e5 e5           sbc sctop        BS-Fenster: oberste Zeile
04593  cc99  aa              tax
04594  cc9a  38              sec
04595  cc9b  a5 ec           lda pntr         akt. Cursor-Spalte
04596  cc9d  e5 e6           sbc sclf         BS-Fenster: Spalte linker Rand
04597  cc9f  a8              tay
04598  cca0  18              clc
04599  cca1  60              rts
04600  cca2
04601  cca2
04602  cca2  ===> Funktionstaste belegen <===
04603  cca2
04604  cca2  ca       keyset dex
04605  cca3  86 dc           stx sedeal       Zeiger Fkt.tasten-Parameter
04606  cca5  84 da           sty sedsal       Länge Ersatz-Text
04607  cca7  8d aa 02        sta fetchx       LDAVEC
04608  ccaa
04609  ccaa  a8              tay
04610  ccab  b6 02           ldx bank,y       Banknummer
04611  ccad  20 6b ff        jsr kgtcfg       MMU-Wert zu Bank in xr nach ac
                                            (Sprungv.)
04612  ccb0  85 de           sta keybnk       Bank Fkt.tasten-Texte
04613  ccb2
04614  ccb2  ===> Zeiger hinter Text Fkt.taste 10 berechnen <===
04615  ccb2
04616  ccb2  a2 0a           ldx #10
```

```
04617  ccb4  20 20 cd          jsr findky       Längen aller Texte addieren
04618  ccb7  85 db             sta sedsah       Zeiger hinter letzten Text
04619  ccb9
04620  ccb9  ===> Zeiger hinter zu ersetzenden Text berechnen <===
04621  ccb9
04622  ccb9  a6 dc             ldx sedeal       addrl Zeiger auf Endezeile
04623  ccbb  e8                inx
04624  ccbc  20 20 cd          jsr findky
04625  ccbf  85 dd             sta sedeah       Zeiger hinter zu ersetzenden Text
04626  ccc1
04627  ccc1  a6 dc             ldx sedeal       addrl Zeiger auf Endezeile
04628  ccc3  a5 da             lda sedsal       Länge Ersatz-Text
04629  ccc5  38                sec              minus
04630  ccc6  fd 00 10          sbc pkybuf,x     Länge zu ersetzender Text
04631  ccc9  f0 2b             beq keyins       neuer Text gleich lang alter Text
04632  cccb  90 16             bcc kyset2       neuer Text kürzer
04633  cccd
04634  cccd  ===> neuer Text länger, Texte verschieben <===
04635  cccd
04636  cccd  18                clc              Längendifferenz
04637  ccce  65 db             adc sedsah       plus Zeiger hinter Text 10
04638  ccd0  b0 4d             bcs keyerr       Fehler, größer 255
04639  ccd2
04640  ccd2  aa                tax              Verschieben Ende
04641  ccd3  a4 db             ldy sedsah       Zeiger hinter Text 10
04642  ccd5  c4 dd      kyset1 cpy sedeah       Zeiger hinter zu ersetzenden Text
04643  ccd7  f0 1d             beq keyins       verschoben
04644  ccd9
04645  ccd9  88                dey
04646  ccda  ca                dex
04647  ccdb  b9 0a 10          lda pkydef,y     Puffer für Fkt.tasten-Befehle
04648  ccde  9d 0a 10          sta pkydef,x     Puffer für Fkt.tasten-Befehle
04649  cce1  b0 f2             bcs kyset1       immer
04650  cce3
04651  cce3
04652  cce3  65 dd      kyset2 adc sedeah       Zeiger hinter Ersatz-Text
04653  cce5  aa                tax              Verschieben Anfang
04654  cce6  a4 dd             ldy sedeah       Zeiger hinter zu ersetzenden Text
04655  cce8  c4 db      kyset3 cpy sedsah       Zeiger hinter Text 10
04656  ccea  b0 0a             bcs keyins       verschoben
04657  ccec
04658  ccec  b9 0a 10          lda pkydef,y     Puffer für Fkt.tasten-Befehle
04659  ccef  9d 0a 10          sta pkydef,x     Puffer für Fkt.tasten-Befehle
04660  ccf2  c8                iny
04661  ccf3  e8                inx
04662  ccf4  90 f2             bcc kyset3       immer
04663  ccf6
04664  ccf6
04665  ccf6  a6 dc      keyins ldx sedeal       Zeiger Anfang zu ersetz. Text
04666  ccf8  20 20 cd          jsr findky
04667  ccfb  aa                tax              Zeiger Anfang Einfügen
04668  ccfc
04669  ccfc  a4 dc             ldy sedeal       Zeiger in Längentab.
04670  ccfe  a5 da             lda sedsal       Länge Ersatz-Text
04671  cd00  99 00 10          sta pkybuf,y     in Längentab. eintragen
04672  cd03
```

```
04673  cd03  ===> neuen Text einfügen <===
04674  cd03
04675  cd03  a0 00             ldy #0
04676  cd05  c6 da       kyins1 dec sedsal       Zeichenzähler (Länge)
04677  cd07  30 15              bmi keyrts       eingefügt
04678  cd09
04679  cd09  86 df              stx keytmp       Zeiger Einfügen
04680  cd0b
04681  cd0b  a6 de              ldx keybnk       Bank Ersatz-Text
04682  cd0d  ad aa 02           lda fetchx       ZP-Zeiger auf Ersatz-Text
04683  cd10
04684  cd10  78                 sei
04685  cd11  20 a2 02           jsr fetch        Zeichen aus Ersatz-Text nach ac
04686  cd14  58                 cli
04687  cd15
04688  cd15  a6 df              ldx keytmp       Zeiger Einfügen
04689  cd17  9d 0a 10           sta pkydef,x     Zeichen in Text-Tab.
04690  cd1a
04691  cd1a  e8                 inx              Einfügen
04692  cd1b  c8                 iny              Ersatztext
04693  cd1c  d0 e7              bne kyins1       immer
04694  cd1e
04695  cd1e
04696  cd1e  18          keyrts clc              Flag für BASIC 'ok'
04697  cd1f  60          keyerr rts
04698  cd20
04699  cd20
04700  cd20  a9 00       findky lda #0
04701  cd22  18                 clc
04702  cd23  ca          fndky1 dex
04703  cd24  30 05              bmi fndky2
04704  cd26
04705  cd26  7d 00 10           adc pkybuf,x     Text-Länge addieren
04706  cd29  90 f8              bcc fndky1
04707  cd2b
04708  cd2b  60          fndky2 rts
04709  cd2c
04710  cd2c
04711  cd2c  ===> Umschalten 40/80-Zeichen-BS (ESC x) <===
04712  cd2c
04713  cd2c  85 f0       swap   sta lstchr       letztes Ausgabezeichen
04714  cd2e
04715  cd2e
04716  cd2e  ===> Umschalten 40/80-Zeichen-BS <===
04717  cd2e
04718  cd2e  a2 1a       swappr ldx #26
04719  cd30  bc 40 0a    swapm2 ldy swpout,x     Swap-Bereich Editor-Werte
04720  cd33  b5 e0              lda pnt,x        addrl Zeiger auf akt. BS-Zeile
04721  cd35
04722  cd35  9d 40 0a           sta swpout,x     Swap-Bereich Editor-Werte
04723  cd38  98                 tya
04724  cd39  95 e0              sta pnt,x        addrl Zeiger auf akt. BS-Zeile
04725  cd3b
04726  cd3b  ca                 dex
04727  cd3c  10 f2              bpl swapm2
04728  cd3e
04729  cd3e  a2 0d              ldx #13
04730  cd40  bc 60 0a    swapm3 ldy swpmap,x     Swap-Bereich Editor-Werte
```

```
04731  cd43  bd 54 03          lda tabmap,x     Tab. der akt. Tabulator-Pos.
04732  cd46
04733  cd46  9d 60 0a          sta swpmap,x     Swap-Bereich Editor-Werte
04734  cd49  98                tya
04735  cd4a  9d 54 03          sta tabmap,x     Tab. der akt. Tabulator-Pos.
04736  cd4d
04737  cd4d  ca                dex
04738  cd4e  10 f0             bpl swapm3
04739  cd50
04740  cd50  a5 d7             lda mode         BS-Modus 40/80-Zeichen
04741  cd52  49 80             eor #%10000000   Bit7 flippen
04742  cd54  85 d7             sta mode         BS-Modus 40/80-Zeichen
04743  cd56
04744  cd56  60          swrts  rts
04745  cd57
04746  cd57
04747  cd57
04748  cd57  ===> Cursorpos. im 80-Zeichen BS setzen <===
04749  cd57
04750  cd57  24 d7       cursor bit mode         BS-Modus 40/80-Zeichen
04751  cd59  10 fb              bpl swrts        40-Zeichen BS nicht möglich
04752  cd5b
04753  cd5b  a2 0e              ldx #14          Register 14 wählen
04754  cd5d  18                 clc
04755  cd5e  a5 e0              lda pnt          addrl Zeiger auf akt. BS-Zeile
04756  cd60  65 ec              adc pntr         plus akt. Cursor-Spalte
04757  cd62  48                 pha              addrl Cursor-Pos. in BS-RAM
04758  cd63
04759  cd63  a5 e1              lda pnt+1        addrh Zeiger auf akt. BS-Zeile
04760  cd65  69 00              adc #0           addrh Cursor-Pos. in BS-RAM
04761  cd67
04762  cd67  20 cc cd           jsr vdcout       an Reg. 14 übergeben
04763  cd6a
04764  cd6a  e8                 inx              Register 15 wählen
04765  cd6b  68                 pla              addrl Cursor-Pos. in BS-RAM
04766  cd6c  4c cc cd           jmp vdcout       an Reg. 15 übergeben
04767  cd6f
04768  cd6f
04769  cd6f  ===> Cursor ein <===
04770  cd6f
04771  cd6f  24 d7       crsron bit mode         BS-Modus 40/80-Zeichen
04772  cd71  10 26              bpl crsr40       40-Zeichen BS
04773  cd73
04774  cd73  ===> 80-Zeichen BS <===
04775  cd73
04776  cd73  20 7c c1           jsr scolor       Zeiger in Farb-RAM nachführen
04777  cd76  a4 ec              ldy pntr         akt. Cursor-Spalte
04778  cd78  20 f9 cd           jsr getusr       Attributadresse im VDC setzen
04779  cd7b  20 d8 cd           jsr vdcget       Daten vom VDC über Update-Register
                                                 lesen
04780  cd7e  8d 33 0a           sta curcol       VDC Cursorfarbe vor dem Blinken
04781  cd81
04782  cd81  29 f0              and #%11110000   Farbe/Helligkeit löschen
04783  cd83  85 db              sta sedsah       Attribute
04784  cd85
04785  cd85  20 f9 cd           jsr getusr       Attributadresse im VDC setzen
04786  cd88
04787  cd88  a5 f1              lda color        akt. Attribut
```

```
04788  cd8a  29 0f           and #%00001111   Attribute löschen
04789  cd8c  05 db           ora sedsah       Attribute zufügen
04790  cd8e  20 ca cd        jsr vdcput       Daten für Block-/normales Schreiben
                                              an VDC
04791  cd91
04792  cd91
04793  cd91  ===> akt. CursorModus an VDC übergeben <===
04794  cd91
04795  cd91  a2 0a     crsrdo ldx #10         Register 10 wählen
04796  cd93  ad 2b 0a         lda curmod       VDC Cursormodus
04797  cd96  4c cc cd         jmp vdcout       ac in Register xr VDC schreiben
04798  cd99
04799  cd99
04800  cd99  ===> 40-Zeichen BS <===
04801  cd99
04802  cd99  a9 00     crsr40 lda #0
04803  cd9b  8d 27 0a         sta blnsw        VIC Flag für Cursor-Blinken
04804  cd9e
04805  cd9e  60               rts
04806  cd9f
04807  cd9f
04808  cd9f  ===> Cursor aus <===
04809  cd9f
04810  cd9f  24 d7     crsrof bit mode         BS-Modus 40/80-Zeichen
04811  cda1  10 10            bpl crsrf1       40-Zeichen BS
04812  cda3
04813  cda3  ===> 80-Zeichen BS <===
04814  cda3
04815  cda3  20 f9 cd         jsr getusr       Attributadresse im VDC setzen
04816  cda6  ad 33 0a         lda curcol       VDC Cursorfarbe vor dem Blinken
04817  cda9  20 ca cd         jsr vdcput       Daten für Block-/normales Schreiben
                                              an VDC
04818  cdac
04819  cdac  a2 0a            ldx #10          Register 10 wählen
04820  cdae  a9 20            lda #%00100000   Bit5 setzen: Cursor aus
04821  cdb0  4c cc cd         jmp vdcout       ac in Register xr VDC schreiben
04822  cdb3
04823  cdb3
04824  cdb3  ===> 40-Zeichen BS <===
04825  cdb3
04826  cdb3  8d 27 0a  crsrf1 sta blnsw        VIC Flag für Cursor-Blinken
04827  cdb6
04828  cdb6  ad 26 0a         lda blnon        Bit7 Cursor-Modus testen
04829  cdb9  10 0e            bpl crsrf2       Cursor aus
04830  cdbb
04831  cdbb  29 40            and #%01000000   Bit7 löschen
04832  cdbd  8d 26 0a         sta blnon        Cursor-Modus 40-Zeichen
04833  cdc0
04834  cdc0  ad 29 0a         lda gdbln        VIC Cursorzeichen vor Blinken
04835  cdc3  ae 2a 0a         ldx gdcol        VIC Cursorfarbe vor Blinken
04836  cdc6  20 34 cc         jsr disply       BS-Ausgabe ac, Farbe/Attribut in xr
04837  cdc9  60        crsrf2 rts
04838  cdca
04839  cdca
04840  cdca  ===> Daten für Block-/normales Schreiben an VDC <===
04841  cdca
04842  cdca  a2 1f     vdcput ldx #31          Register 31 wählen
04843  cdcc
```

```
04844  cdcc
04845  cdcc  ===> ac in Register xr VDC schreiben <===
04846  cdcc
04847  cdcc  8e 00 d6   vdcout stx vdcadr       VDC 8563 Adress-Register
04848  cdcf
04849  cdcf  2c 00 d6   vdcso  bit vdcadr       Status-Bit VDC
04850  cdd2  10 fb             bpl vdcso        warten auf READY
04851  cdd4
04852  cdd4  8d 01 d6          sta vdcdat       VDC 8563 Daten-Register
04853  cdd7  60                rts
04854  cdd8
04855  cdd8
04856  cdd8  ===> Daten vom VDC über Update-Register lesen <===
04857  cdd8
04858  cdd8  a2 1f      vdcget ldx #31          Register 31 wählen
04859  cdda
04860  cdda
04861  cdda
04862  cdda  ===> Daten vom VDC lesen, Register = xr <===
04863  cdda
04864  cdda  8e 00 d6   vdcin  stx vdcadr       VDC 8563 Adress-Register
04865  cddd
04866  cddd  2c 00 d6   vdcsi  bit vdcadr       Status-Bit VDC
04867  cde0  10 fb             bpl vdcsi        warten auf READY
04868  cde2
04869  cde2  ad 01 d6          lda vdcdat       VDC 8563 Daten-Register
04870  cde5  60                rts
04871  cde6
04872  cde6
04873  cde6  ===> Textadresse im VDC setzen <===
04874  cde6
04875  cde6  a2 12      getpnt ldx #18          Register 18 wählen
04876  cde8  18                clc
04877  cde9  98                tya             Offset
04878  cdea  65 e0             adc pnt         addrl Zeiger auf akt. BS-Zeile
04879  cdec  48                pha             addrl in BS-RAM
04880  cded
04881  cded  a9 00             lda #0
04882  cdef  65 e1             adc pnt+1       addrh in BS-RAM
04883  cdf1
04884  cdf1  20 cc cd          jsr vdcout      ac in Register xr VDC schreiben
04885  cdf4
04886  cdf4  68                pla             addrl in BS-RAM
04887  cdf5  e8                inx             Register 19 wählen
04888  cdf6  4c cc cd          jmp vdcout      ac in Register xr VDC schreiben
04889  cdf9
04890  cdf9
04891  cdf9  ===> Attributadresse im VDC setzen <===
04892  cdf9
04893  cdf9  a2 12      getusr ldx #18          Register 18 wählen
04894  cdfb  18                clc
04895  cdfc  98                tya             Offset
04896  cdfd  65 e2             adc user        addrl Zeiger auf akt. Attributzeile
04897  cdff  48                pha             addrl im Attribut-RAM
04898  ce00
04899  ce00  a9 00             lda #0
04900  ce02  65 e3             adc user+1      addrh im Attribut-RAM
04901  ce04
```

```
04902  ce04  20 cc cd          jsr vdcout       ac in Register xr VDC schreiben
04903  ce07
04904  ce07  68               pla              addrl im Attribut-RAM
04905  ce08  e8               inx              Register 19 wählen
04906  ce09  4c cc cd         jmp vdcout       ac in Register xr VDC schreiben
04907  ce0c
04908  ce0c
04909  ce0c  ===> Char.gen. in VDC laden <===
04910  ce0c
04911  ce0c  a9 00      init80 lda #<chrrom     addrl Zeichengenerator-ROM
04912  ce0e  a0 d0             ldy #>chrrom     addrh Zeichengenerator-ROM
04913  ce10  85 da             sta sedsal       addrl Zeiger auf Startzeile
04914  ce12  84 db             sty sedsah       addrh Zeiger auf Startzeile
04915  ce14
04916  ce14  a2 12             ldx #18          Register 18 wählen
04917  ce16  a9 20             lda #>vdcchr
04918  ce18  20 cc cd          jsr vdcout       ac in Register xr VDC schreiben
04919  ce1b
04920  ce1b  e8                inx              Register 19 wählen
04921  ce1c  a9 00             lda #<vdcchr
04922  ce1e  20 cc cd          jsr vdcout       ac in Register xr VDC schreiben
04923  ce21
04924  ce21  ===> acht Bytes aus Char.gen.-ROM ins VDC-RAM <===
04925  ce21
04926  ce21  a0 00             ldy #0           Zeiger in Char.gen.
04927  ce23  a2 0e      int80a ldx #14
04928  ce25  a9 da             lda #<sedsal     addrl Zeiger auf Startzeile
04929  ce27  20 74 ff          jsr kifetv       LDA (FETVEC),y von Bank in xr
                                                (Sprungv.)
04930  ce2a  20 ca cd          jsr vdcput       Daten für Block-/normales Schreiben
                                                an VDC
04931  ce2d
04932  ce2d  c8                iny
04933  ce2e  c0 08             cpy #8
04934  ce30  90 f1             bcc int80a
04935  ce32
04936  ce32  ===> acht Bytes $00 ins VDC-RAM <===
04937  ce32
04938  ce32  a9 00             lda #0
04939  ce34  20 ca cd   int80b jsr vdcput       Daten für Block-/normales Schreiben
                                                an VDC
04940  ce37  88                dey
04941  ce38  d0 fa             bne int80b
04942  ce3a
04943  ce3a  ===> Zeiger in Char.gen.-ROM um 8 erhöhen <===
04944  ce3a
04945  ce3a  18                clc
04946  ce3b  a5 da             lda sedsal       addrl Zeiger auf Startzeile
04947  ce3d  69 08             adc #8
04948  ce3f  85 da             sta sedsal       addrl Zeiger auf Startzeile
04949  ce41
04950  ce41  90 e0             bcc int80a
04951  ce43  e6 db             inc sedsah       addrh Zeiger auf Startzeile
04952  ce45
04953  ce45  a5 db             lda sedsah       addrh Zeiger auf Startzeile
04954  ce47  c9 e0             cmp #>chrrom+$1000     Ende Char.gen.+1
04955  ce49  90 d8             bcc int80a
04956  ce4b
```

```
04957  ce4b  60              rts
04958  ce4c
04959  ce4c
04960  ce4c  ===> Tab. der Farbtasten <===
04961  ce4c
04962  ce4c  90        coltab .byte $90       Tastencode für schwarz
04963  ce4d  05               .byte $05       Tastencode für weiß
04964  ce4e  1c               .byte $1c       Tastencode für rot
04965  ce4f  9f               .byte $9f       Tastencode für grün
04966  ce50  9c               .byte $9c       Tastencode für violett
04967  ce51  1e               .byte $1e       Tastencode für dunkelgrün
04968  ce52  1f               .byte $1f       Tastencode für blau
04969  ce53  9e               .byte $9e       Tastencode für gelb
04970  ce54  81               .byte $81       Tastencode für hellbraun
04971  ce55  95               .byte $95       Tastencode für braun
04972  ce56  96               .byte $96       Tastencode für rosa
04973  ce57  97               .byte $97       Tastencode für dunkelgrau
04974  ce58  98               .byte $98       Tastencode für grau
04975  ce59  99               .byte $99       Tastencode für hellgrün
04976  ce5a  9a               .byte $9a       Tastencode für hellblau
04977  ce5b  9b               .byte $9b       Tastencode für hellgrau
04978  ce5c
04979  ce5c
04980  ce5c  ===> Tab. der Farbwerte 80-Zeichen BS <===
04981  ce5c
04982  ce5c  00        coladj .byte %00000000 VDC-Farbwert für schwarz
04983  ce5d  0f               .byte %00001111 VDC-Farbwert für weiß
04984  ce5e  08               .byte %00001000 VDC-Farbwert für rot
04985  ce5f  07               .byte %00000111 VDC-Farbwert für grün
04986  ce60  0b               .byte %00001011 VDC-Farbwert für violett
04987  ce61  04               .byte %00000100 VDC-Farbwert für dunkelgrün
04988  ce62  02               .byte %00000010 VDC-Farbwert für blau
04989  ce63  0d               .byte %00001101 VDC-Farbwert für gelb
04990  ce64  0a               .byte %00001010 VDC-Farbwert für hellbraun
04991  ce65  0c               .byte %00001100 VDC-Farbwert für braun
04992  ce66  09               .byte %00001001 VDC-Farbwert für rosa
04993  ce67  06               .byte %00000110 VDC-Farbwert für dunkelgrau
04994  ce68  01               .byte %00000001 VDC-Farbwert für grau
04995  ce69  05               .byte %00000101 VDC-Farbwert für hellgrün
04996  ce6a  03               .byte %00000011 VDC-Farbwert für hellblau
04997  ce6b  0e               .byte %00001110 VDC-Farbwert für hellgrau
04998  ce6c
04999  ce6c
05000  ce6c  ===> Tabelle der Bit-Masken <===
05001  ce6c
05002  ce6c  80 40 20  bits   .byte %10000000, %01000000, %00100000, %00010000
05003  ce6f  10
05004  ce70  08 04 02         .byte %00001000, %00000100, %00000010, %00000001
05005  ce73  01
05006  ce74
05007  ce74
05008  ce74  ===> Editor-Vorgaben für 40-Zeichen BS <===
05009  ce74
05010  ce74  00 04     loczp  .word linz1     Bildschirmzeile 1
05011  ce76  00 d8            .word colram    VIC Farb-RAM
05012  ce78  18               .byte 24        untere Zeile akt. Bildschirmfenster
05013  ce79  00               .byte 0         obere Zeile akt. Bildschirmfenster
05014  ce7a  00               .byte 0         linke Spalte akt. Bildschirmfenster
```

```
05015  ce7b  27              .byte 39        rechte Spalte akt. Bildschirmfenster
05016  ce7c  00              .byte 0         akt. Cursorzeile vor input (Vorgabe)
05017  ce7d  00              .byte 0         akt. Cursorspalte vor input
                                             (Vorgabe)
05018  ce7e  00              .byte 0         Pos. letztes Zeichen in Zeile
                                             (Vorgabe)
05019  ce7f  00              .byte 0         akt. Cursorzeile (Vorgabe)
05020  ce80  00              .byte 0         akt. Cursorspalte (Vorgabe)
05021  ce81  18              .byte 24        max. Anzahl Bildschirmzeilen (0-24)
05022  ce82  27              .byte 39        max. Anzahl Bildschirmspalten (0-39)
05023  ce83  00              .byte 0         akt. Ausgabezeichen (Vorgabe)
05024  ce84  00              .byte 0         letzes ausgegebenes Zeichen
                                             (Vorgabe)
05025  ce85  0d              .byte 13        Attribut nächstes auszugebendes
                                             Zeichen (Vorgabe grau)
05026  ce86  0d              .byte 13        temp. Reg. für Zeichenattribut
                                             (insert/delete, Vorgabe)
05027  ce87  00              .byte 0         REVERS-Flag (Vorgabe: REVERS off)
05028  ce88  00              .byte 0         QUOTE-Modus-Flag (Vorgabe: aus)
05029  ce89  00              .byte 0         INSERT-Modus-Flag (Vorgabe: aus)
05030  ce8a  00              .byte 0         AUTO-INSERT-Modus-Flag (Vorgabe:
                                             aus)
05031  ce8b  00              .byte 0         Disable Flag Shift-CBM/CTRL s
                                             (Vorgabe: 0 = enable)
05032  ce8c  00              .byte 0         Scroll-Flag (Vorgabe: 0 = phys./log.
                                             Scroll ein)
05033  ce8d  00              .byte 0         Disable Flag BEEPER (Vorgabe: 0 =
                                             BEEP ein)
05034  ce8e
05035  ce8e
05036  ce8e  ===> Editor-Vorgaben für 80-Zeichen-BS <===
05037  ce8e
05038  ce8e  00 00     locabs .word vdcscn     Basisadresse VDC BS-RAM (intern)
05039  ce90  00 08            .word vdccol     Basisadr. VDC Attribut-RAM (intern)
05040  ce92  18               .byte 24         untere Zeile akt. Bildschirmfenster
05041  ce93  00               .byte 0          obere Zeile akt. Bildschirmfenster
05042  ce94  00               .byte 0          linke Spalte akt. Bildschirmfenster
05043  ce95  4f               .byte 79         rechte Spalte akt. Bildschirmfenster
05044  ce96  00               .byte 0          akt. Cursorzeile vor input (Vorgabe)
05045  ce97  00               .byte 0          akt. Cursorspalte vor input
                                             (Vorgabe)
05046  ce98  00               .byte 0          Pos. letztes Zeichen in Zeile
                                             (Vorgabe)
05047  ce99  00               .byte 0          akt. Cursorzeile (Vorgabe)
05048  ce9a  00               .byte 0          akt. Cursorspalte (Vorgabe)
05049  ce9b  18               .byte 24         max. Anzahl Bildschirmzeilen (0-24)
05050  ce9c  4f               .byte 79         max. Anzahl Bildschirmspalten (0-79)
05051  ce9d  00               .byte 0          akt. Ausgabezeichen (Vorgabe)
05052  ce9e  00               .byte 0          letzes ausgegebenes Zeichen
                                             (Vorgabe)
05053  ce9f  07               .byte 7          Attribut nächstes auszugebendes
                                             Zeichen (Vorgabe grau)
05054  cea0  07               .byte 7          temp. Reg. für Zeichenattribut
                                             (insert/delete, Vorgabe)
05055  cea1  00               .byte 0          REVERS-Flag (Vorgabe: REVERS off)
05056  cea2  00               .byte 0          QUOTE-Modus-Flag (Vorgabe: aus)
05057  cea3  00               .byte 0          INSERT-Modus-Flag (Vorgabe: aus)
```

```
05058  cea4  00              .byte 0         AUTO-INSERT-Modus-Flag (Vorgabe:
                                             aus)
05059  cea5  00              .byte 0         Disable Flag Shift-CBM/CTRL s
                                             (Vorgabe: 0 = enable)
05060  cea6  00              .byte 0         Scroll-Flag (Vorgabe: 0 = phys./log.
                                             Scroll ein)
05061  cea7  00              .byte 0         Disable Flag BEEPER (Vorgabe: 0 =
                                             BEEP ein)
05062  cea8
05063  cea8
05064  cea8  ===> Längen der Standard-Texte der Funktionstasten <===
05065  cea8
05066  cea8  07 06 0a   pkytbl .byte 7, 6, 10, 7, 6, 4, 5, 8, 9, 5
05067  ceab  07 06 04
05068  ceae  05 08 09
05069  ceb1  05
05070  ceb2
05071  ceb2
05072  ceb2  ===> Standardtext Funktionstaste 1 <===
05073  ceb2
05074  ceb2  47 52 41   pky1   .byte 'graphic'
05075  ceb5  50 48 49
05076  ceb8  43
05077  ceb9
05078  ceb9
05079  ceb9  ===> Standardtext Funktionstaste 2 <===
05080  ceb9
05081  ceb9  44 4c 4f   pky2   .byte 'dload"'
05082  cebc  41 44 22
05083  cebf
05084  cebf
05085  cebf  ===> Standardtext Funktionstaste 3 <===
05086  cebf
05087  cebf  44 49 52   pky3   .byte 'directory', cr
05088  cec2  45 43 54
05089  cec5  4f 52 59
05090  cec8  0d
05091  cec9
05092  cec9
05093  cec9  ===> Standardtext Funktionstaste 4 <===
05094  cec9
05095  cec9  53 43 4e   pky4   .byte 'scnclr', cr
05096  cecc  43 4c 52
05097  cecf  0d
05098  ced0
05099  ced0
05100  ced0  ===> Standardtext Funktionstaste 5 <===
05101  ced0
05102  ced0  44 53 41   pky5   .byte 'dsave"'
05103  ced3  56 45 22
05104  ced6
05105  ced6
05106  ced6  ===> Standardtext Funktionstaste 6 <===
05107  ced6
05108  ced6  52 55 4e   pky6   .byte 'run', cr
05109  ced9  0d
05110  ceda
05111  ceda
```

```
05112  ceda  ===> Standardtext Funktionstaste 7 <===
05113  ceda
05114  ceda  4c 49 53   pky7   .byte 'list', cr
05115  cedd  54 0d
05116  cedf
05117  cedf
05118  cedf  ===> Standardtext Funktionstaste 8 <===
05119  cedf
05120  cedf  4d 4f 4e   pky8   .byte 'monitor', cr
05121  cee2  49 54 4f
05122  cee5  52 0d
05123  cee7
05124  cee7
05125  cee7  ===> Standardtext Funktionstaste 9 <===
05126  cee7
05127  cee7  44 cc 22   runtb  .byte 'dL"*', cr, 'run', cr
05128  ceea  2a 0d 52
05129  ceed  55 4e 0d
05130  cef0
05131  cef0
05132  cef0  ===> Standardtext Funktionstaste 10 <===
05133  cef0
05134  cef0  48 45 4c   helptb .byte 'help', cr
05135  cef3  50 0d
05136  cef5            pkyend
05137  cef5
05138  cef5
05139  cef5                    *=$cffe
05140  cffe
05141  cffe  00 ff             .byte $00, $ff   Prüfsumme
05142  d000
05143  d000
05144  d000
05145  d000                    .end
05146  d000                    .lib c128.init
05147  d000
```

4.1.11 Initialisierung

```
05149  d000
05150  d000
05151  d000                    * = $e000
05152  e000
05153  e000
05154  e000  ===> System-Start (RESET) <===
05155  e000
05156  e000  a2 ff      start  ldx #$ff
05157  e002  78               sei
05158  e003  9a               txs
05159  e004  d8               cld
05160  e005
05161  e005  a9 00            lda #%00000000
05162  e007
05163  e007   00xxxxxx  RAM Bank 0
05164  e007   xx00xxxx  KERNAL ROM    $c000-$ffff
05165  e007   xxxx00xx  BASIC ROM hi  $8000-$bfff
```

```
05166  e007   xxxxxx0x  BASIC ROM lo  $4000-$7fff
05167  e007   xxxxxxx0  I/O-Chips     $d000-$dfff
05168  e007
05169  e007  8d 00 ff          sta mmucr        MMU 8722 Config. Reg. (Kopie)
05170  e00a
05171  e00a  ===> Initialisierung der MMU 8722 Register <===
05172  e00a
05173  e00a  a2 0a              ldx #10
05174  e00c  bd 4b e0   syssr1  lda mmutbl,x     Tab. RESET-Werte der MMU 8722
05175  e00f  9d 00 d5           sta mmucrl,x     alle MMU Register schreiben
05176  e012  ca                 dex
05177  e013  10 f7             bpl syssr1
05178  e015
05179  e015  8d 04 0a          sta initst       Flag Status nach RESET, BASIC-IRQ
05180  e018
05181  e018  20 cd e0           jsr instll       Systemroutinen in alle Banks
                                                 kopieren
05182  e01b  20 f0 e1           jsr secure       Test auf CBM-ID in RAM Bank 1
05183  e01e  20 42 e2           jsr poll         Test auf Steck-Module
05184  e021  20 09 e1           jsr ioinit       I/O-Bausteine initialisieren
05185  e024  20 3d f6           jsr ud60         Abfrage Stopp/Shift-Taste/Spalte7
05186  e027  48                 pha              CPU Status retten
05187  e028  30 07              bmi scold        Stopp-Taste n. gedrückt, Kaltstart
05188  e02a
05189  e02a  ===> RESET+STOPP: Test / Warmstart <===
05190  e02a
05191  e02a  a9 a5              lda #%10100101   Bitmuster nach Kaltstart
05192  e02c
05193  e02c  enthält die Speicherstelle $0a02 beim Einschalten zufällig
05194  e02c  dieses Bitmuster, wird kein Kaltstart durchgeführt.
05195  e02c
05196  e02c  cd 02 0a           cmp dejavu       Flag (KERNAL-) Warm/Kalt-Start
05197  e02f  f0 03              beq swarm
05198  e031
05199  e031  ===> Kaltstart Operating System <===
05200  e031
05201  e031  20 93 e0   scold   jsr ramtas       ZP, Kass.-Puffer u. Vectoren
                                                 einrichten
05202  e034
05203  e034
05204  e034  ===> Warmstart Operating System <===
05205  e034
05206  e034  20 56 e0   swarm   jsr restor       Standard I/O-Vectoren einrichten
05207  e037  20 00 c0           jsr jpcint       Video-Chips/Editor init.
05208  e03a
05209  e03a  68                 pla              Status gedrückte Taste bei RESET
05210  e03b  58                 cli
05211  e03c  30 03              bmi smod         Stopp-Taste nicht gedrückt
05212  e03e
05213  e03e  ===> RESET+STOPP: Warmstart + MONITOR <===
05214  e03e
05215  e03e  4c 00 b0           jmp moncal       Monitor-Einsprung über CALL
05216  e041
05217  e041
05218  e041  ===> Test C128 BASIC / C 64 - Modus <===
05219  e041
05220  e041  c9 df       smod   cmp #%11011111   Test auf gedr. CBM-Taste
05221  e043  f0 03              beq go64         CBM-Taste gedrückt, C 64 Modus
```

```
05222  e045
05223  e045  6c 00 0a          jmp (sysvec)     C 128 BASIC Kaltstart
05224  e048
05225  e048
05226  e048  ===> RESET+CBM: Kaltstart C 64 <===
05227  e048
05228  e048  4c 4b e2   go64   jmp c64mod       C 64 Kaltstart
05229  e04b
05230  e04b
05231  e04b  ===> Tab. RESET-Werte der MMU 8722 <===
05232  e04b
05233  e04b  00         mmutbl .byte %00000000  Config. Reg.
05234  e04c  00                .byte %00000000  Preconfig. Reg. A
05235  e04d  00                .byte %00000000  Preconfig. Reg. B
05236  e04e  00                .byte %00000000  Preconfig. Reg. C
05237  e04f  00                .byte %00000000  Preconfig. Reg. D
05238  e050
05239  e050   00xxxxxx  RAM Bank 0
05240  e050   xx00xxxx  KERNAL ROM    $c000-$ffff
05241  e050   xxxx00xx  BASIC ROM hi  $8000-$bfff
05242  e050   xxxxxx0x  BASIC ROM lo  $4000-$7fff
05243  e050   xxxxxxx0  I/O-Chips     $d000-$dfff
05244  e050
05245  e050  bf                .byte %10111111  Mode Config. Reg.
05246  e051
05247  e051   1xxxxxxx  80-Zeichen BS
05248  e051   x0xxxxxx  C 128 Modus
05249  e051   xx11111x  keine Wirkung
05250  e051   xxxxxxx1  CPU 8502
05251  e051
05252  e051  04                .byte %00000100  RAM Config. Reg.
05253  e052
05254  e052   00xxxxxx  RAM Bank 0
05255  e052   xx00xxxx  256K Block 0
05256  e052   xxxx01xx  untere COMMON AREA
05257  e052   xxxxxx00  COMMON AREA 1 KByte
05258  e052
05259  e052  00 00      .dbyte $0000           Zeropage-Zeiger
05260  e054  01 00      .dbyte $0100           CPU Stapel-Zeiger
05261  e056
05262  e056
05263  e056  ===> Standard I/O-Vectoren einrichten <===
05264  e056
05265  e056  a2 73      restor ldx #<vectss     Adreßtabelle der Standard-Vectoren
05266  e058  a0 e0             ldy #>vectss     Adreßtabelle der Standard-Vectoren
05267  e05a  18                clc
05268  e05b
05269  e05b
05270  e05b  ===> User-/Standard I/O-Vectoren einrichten <===
05271  e05b
05272  e05b  86 c3      vector stx tmp2         addrl Speicherzeiger
05273  e05d  84 c4             sty tmp2+1       addrh Speicherzeiger
05274  e05f
05275  e05f  a0 1f             ldy #31
05276  e061  b9 14 03   movos1 lda iirq,y       Tab. Standard-Vectoren
05277  e064  b0 02             bcs movos2       Standard-Vectoren einrichten
05278  e066
05279  e066  b1 c3             lda (tmp2),y
```

```
05280  e068  99 14 03   movos2 sta iirq,y       Tab. System-Vectoren
05281  e06b  90 02             bcc movos3
05282  e06d  91 c3             sta (tmp2),y
05283  e06f
05284  e06f  88         movos3 dey
05285  e070  10 ef             bpl movos1
05286  e072
05287  e072  60                rts
05288  e073
05289  e073
05290  e073  ===> Adreßtabelle der Standard-Vectoren <===
05291  e073
05292  e073  65 fa      vectss .word nirq        IRQ-Routine
05293  e075  03 b0             .word monbrk      Monitor-Einsprung über BREAK
05294  e077  40 fa             .word nnmi        NMI-Routine
05295  e079  bd ef             .word nopen       log. File öffnen, Ausgabe fn
05296  e07b  88 f1             .word nclose      log. File schließen
05297  e07d  06 f1             .word nchkin      Eingabekanal (la in xr) öffnen
05298  e07f  4c f1             .word nckout      Ausgabe-Kanal (la in xr) öffnen
05299  e081  26 f2             .word nclrch      Ein-/Ausgabe-Kanal schließen
05300  e083  06 ef             .word nbasin      Zeichen vom aktiven Kanal nach ac
05301  e085  79 ef             .word nbsout      ac auf aktivem Kanal ausgeben
05302  e087  6e f6             .word nstop       Stopp-Taste abfragen
05303  e089  eb ee             .word ngetin      Zeichen vom aktiven Kanal nach ac
05304  e08b  22 f2             .word nclall      offene Files = 0, Ersatz-I/O-Kanäle
05305  e08d  06 b0             .word moncmd      Monitor-Kommando ausführen
05306  e08f  6c f2             .word nload       Lesen vom logischen File
05307  e091  4e f5             .word nsave       Abspeichern auf logisches File
05308  e093
05309  e093
05310  e093  ===> ZP, Kass.-Puffer u. Vectoren einrichten <===
05311  e093
05312  e093  a9 00      ramtas lda #0
05313  e095  a8                tay
05314  e096
05315  e096
05316  e096  ===> ZP mit $00 füllen <===
05317  e096
05318  e096  99 02 00   ramtz0 sta bank,y       Banknummer
05319  e099  c8                iny
05320  e09a  d0 fa             bne ramtz0       ZP mit $00 füllen
05321  e09c
05322  e09c  a0 0b             ldy #>tbuffr     Kassettenpuffer
05323  e09e  84 b3             sty tape1+1      addrh Zeiger Kassettenpuffer
05324  e0a0  85 b2             sta tape1        addrl Zeiger Kassettenpuffer
05325  e0a2
05326  e0a2  a0 0c             ldy #>rsibuf     RS 232 Eingabe-Puffer
05327  e0a4  84 c9             sty ribuf+1      RS 232 addrh Eingabe-Puffer
05328  e0a6  85 c8             sta ribuf        RS 232 addrl Eingabe-Puffer
05329  e0a8
05330  e0a8  a0 0d             ldy #>rsobuf     RS 232 Ausgabe-Puffer
05331  e0aa  84 cb             sty robuf+1      RS 232 addrh Ausgabepuffer
05332  e0ac  85 ca             sta robuf        RS 232 addrl Ausgabepuffer
05333  e0ae
05334  e0ae  18                clc
05335  e0af  a0 ff             ldy #>mmucr      MMU 8722 Config. Reg. (Kopie)
05336  e0b1  a2 00             ldx #<mmucr      MMU 8722 Config. Reg. (Kopie)
```

```
05337  e0b3  20 6b f7          jsr settop       xr,yr in Zeiger höchste
                                                Speicheradresse
05338  e0b6
05339  e0b6  a0 1c             ldy #>rambot     Anfang Speicher BASIC-Programm
05340  e0b8  a2 00             ldx #<rambot     Anfang Speicher BASIC-Programm
05341  e0ba  20 7a f7          jsr setbot       xr,yr in Zeiger Anfang
                                                Benutzerspeicher
05342  e0bd
05343  e0bd  a0 40             ldy #>coldvc     Vector (BASIC-) Kaltstart
05344  e0bf  a2 00             ldx #<coldvc     Vector (BASIC-) Kaltstart
05345  e0c1  8c 01 0a          sty sysvec+1     Vector (BASIC-) Warmstart
05346  e0c4  8e 00 0a          stx sysvec       Vector (BASIC-) Warmstart ($4003)
05347  e0c7
05348  e0c7  a9 a5             lda #%10100101   Bitmuster nach Kaltstart
05349  e0c9  8d 02 0a          sta dejavu       Flag (KERNAL-) Warm/Kalt-Start
05350  e0cc
05351  e0cc  60                rts
05352  e0cd
05353  e0cd
05354  e0cd  ===> Systemroutinen in alle Banks kopieren <===
05355  e0cd
05356  e0cd  a0 03       instll ldy #3
05357  e0cf  b9 05 e1    instl1 lda insttb,y     MMU Config. Werte für alle Banks
05358  e0d2  8d 00 ff           sta mmucr        MMU 8722 Config. Reg. (Kopie)
05359  e0d5
05360  e0d5   bei obigen Config.-Werten wird aus dem ROM gelesen und in
05361  e0d5   dahinter liegendes RAM geschrieben:
05362  e0d5
05363  e0d5  ===> NMI-, IRQ- und RESET-Routinen in alle Banks <===
05364  e0d5
05365  e0d5  a2 3f              ldx #63
05366  e0d7  bd 05 ff    instl2 lda nmi,x        NMI-Routine ROM
05367  e0da  9d 05 ff           sta nmi,x        NMI-Routine RAM
05368  e0dd  ca                 dex
05369  e0de  10 f7              bpl instl2
05370  e0e0
05371  e0e0  ===> NMI-, IRQ- und RESET-Vectoren in alle Banks <===
05372  e0e0
05373  e0e0  a2 05              ldx #5
05374  e0e2  bd fa ff    instl3 lda sysnmi,x     Vector ROM
05375  e0e5  9d fa ff           sta sysnmi,x     Vector RAM
05376  e0e8  ca                 dex
05377  e0e9  10 f7              bpl instl3
05378  e0eb
05379  e0eb  88                 dey
05380  e0ec  10 e1              bpl instl1
05381  e0ee
05382  e0ee  ===> Routinen zum indirekten Lesen, Schreiben Vergleichen <===
05383  e0ee            JSR und JMP beliebige Bank ins RAM kopieren
05384  e0ee
05385  e0ee  a2 59              ldx #89
05386  e0f0  bd 00 f8    instl4 lda fetcho,x     Routinen aus ROM
05387  e0f3  9d a2 02           sta fetch,x      ins RAM übertragen
05388  e0f6  ca                 dex
05389  e0f7  10 f7              bpl instl4
05390  e0f9
```

```
05391  e0f9  ===> Abschluß DMA-Routine ins RAM kopieren <===
05392  e0f9
05393  e0f9  a2 0c            ldx #12
05394  e0fb  bd 5a f8   instl5 lda dodmao,x     Abschluß DMA Routine (RAM $03f0)
05395  e0fe  9d f0 03         sta dodma,x      DMA-Register setzen
05396  e101  ca               dex
05397  e102  10 f7            bpl instl5
05398  e104
05399  e104  60               rts
05400  e105
05401  e105
05402  e105  ===> MMU-Werte für System-Installierung <===
05403  e105
05404  e105  00         insttb .byte %00000000
05405  e106
05406  e106   00xxxxxx  RAM Bank 0
05407  e106   xx00xxxx  KERNAL ROM     $c000-$ffff
05408  e106   xxxx00xx  BASIC ROM hi   $8000-$bfff
05409  e106   xxxxxx0x  BASIC ROM lo   $4000-$7fff
05410  e106   xxxxxxx0  I/O-Chips      $d000-$dfff
05411  e106
05412  e106  40               .byte %01000000
05413  e107
05414  e107   01xxxxxx  RAM Bank 1
05415  e107   xx00xxxx  KERNAL ROM     $c000-$ffff
05416  e107   xxxx00xx  BASIC ROM hi   $8000-$bfff
05417  e107   xxxxxx0x  BASIC ROM lo   $4000-$7fff
05418  e107   xxxxxxx0  I/O-Chips      $d000-$dfff
05419  e107
05420  e107  80               .byte %10000000
05421  e108
05422  e108   10xxxxxx  RAM Bank 2 (hier RAM Bank 0)
05423  e108   xx00xxxx  KERNAL ROM     $c000-$ffff
05424  e108   xxxx00xx  BASIC ROM hi   $8000-$bfff
05425  e108   xxxxxx0x  BASIC ROM lo   $4000-$7fff
05426  e108   xxxxxxx0  I/O-Chips      $d000-$dfff
05427  e108
05428  e108  c0               .byte %11000000
05429  e109
05430  e109   11xxxxxx  RAM Bank 3 (hier RAM Bank 1)
05431  e109   xx00xxxx  KERNAL ROM     $c000-$ffff
05432  e109   xxxx00xx  BASIC ROM hi   $8000-$bfff
05433  e109   xxxxxx0x  BASIC ROM lo   $4000-$7fff
05434  e109   xxxxxxx0  I/O-Chips      $d000-$dfff
05435  e109
05436  e109
05437  e109
05438  e109  ===> I/O-Bausteine initialisieren <===
05439  e109
05440  e109  a9 7f      ioinit lda #%01111111
05441  e10b
05442  e10b   01111111  CLEAR alle IRQ-Maskenbits
05443  e10b
05444  e10b  8d 0d dc          sta d1icr        CIA1 Interrupt Control-Register
05445  e10e  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
05446  e111
05447  e111   01111111  Tastatur Spalte 7 abfragen (Stopp/CTRL/CBM)
05448  e111
```

```
05449  e111  8d 00 dc          sta d1pra
05450  e114
05451  e114  a9 08             lda #%00001000
05452  e116
05453  e116   0xxxxxxx  60 Hz an Time-Of-Day
05454  e116   x0xxxxxx  Serial Port Input
05455  e116   xx0xxxxx  Timer A zählt Phi2-Pulse
05456  e116   xxx0xxxx  wirkunsglos
05457  e116   xxxx1xxx  Timer A One-Shot
05458  e116   xxxxx0xx  Timer A Pulse
05459  e116   xxxxxx0x  PB6 normal Operation
05460  e116   xxxxxxx0  Stop Timer A
05461  e116
05462  e116  8d 0e dc          sta d1cra        CIA1 Control-Register a
05463  e119  8d 0e dd          sta d2cra        CIA2 Control-Register A
05464  e11c
05465  e11c   0xxxxxxx  Write Time-Of-Day-Reg. stellt Uhr
05466  e11c   x00xxxxx  Timer B zählt Phi2-Pulse
05467  e11c   xxx0xxxx  wirkunsglos
05468  e11c   xxxx1xxx  Timer B One-Shot
05469  e11c   xxxxx0xx  Timer B Pulse
05470  e11c   xxxxxx0x  PB7 normal Operation
05471  e11c   xxxxxxx0  Stop Timer B
05472  e11c
05473  e11c  8d 0f dc          sta d1crb        CIA1 Control-Register b
05474  e11f  8d 0f dd          sta d2crb        CIA2 Control-Register B
05475  e122
05476  e122  a2 00             ldx #%00000000
05477  e124
05478  e124   00000000  Port B Input Tastatur Reihen
05479  e124
05480  e124  8e 03 dc          stx d1ddrb       CIA1 Datenrichtungs-Reg. b
05481  e127  8e 03 dd          stx d2ddrb       CIA2 Datenrichtungs-Reg. B
05482  e12a
05483  e12a  ca                dex
05484  e12b
05485  e12b   11111111  Port A Output Tastatur Spalten
05486  e12b
05487  e12b  8e 02 dc          stx d1ddra       CIA1 Datenrichtungs-Reg. a
05488  e12e
05489  e12e  a9 07             lda #%00000111
05490  e130
05491  e130   0xxxxxxx  DATA Input Serial Device
05492  e130   x0xxxxxx  CLOCK Input Serial Device
05493  e130   xx0xxxxx  DATA Output Serial Device
05494  e130   xxx0xxxx  CLOCK Output Serial Device
05495  e130   xxxx0xxx  ATN Output Serial Device
05496  e130   xxxxx2xx  TXD Transmit Data
05497  e130   xxxxxx11  Adressbits 15/14 BS-RAM VIC
05498  e130
05499  e130  8d 00 dd          sta d2pra        CIA2 Datenport A
05500  e133
05501  e133  a9 3f             lda #%00111111
05502  e135
05503  e135   0xxxxxxx  INPUT  DATA Serial Device
05504  e135   x0xxxxxx  INPUT  CLOCK Serial Device
05505  e135   xx1xxxxx  OUTPUT DATA Serial Device
05506  e135   xxx1xxxx  OUTPUT CLOCK Serial Device
```

```
05507  e135   xxxx1xxx  OUTPUT ATN Serial Device
05508  e135   xxxxx1xx  OUTPUT TXD Transmit Data
05509  e135   xxxxxx11  OUTPUT Adressbits 15/14 BS-RAM VIC
05510  e135
05511  e135  8d 02 dd          sta d2ddra       CIA2 Datenrichtungs-Reg. A
05512  e138
05513  e138  a9 e3             lda #%11100011
05514  e13a
05515  e13a   1xxxxxxx  wirkungslos (unbenutzt)
05516  e13a   x1xxxxxx  wirkungslos (Input)
05517  e13a   xx1xxxxx  Kassettenmotor
05518  e13a   xxx0xxxx  wirkungslos (Input)
05519  e13a   xxxx0xxx  Kassette serial out
05520  e13a   xxxxx0xx  Char.gen. ROM C 64 ein ($d000-$dfff)
05521  e13a   xxxxxx1x  KERNAL ROM C 64 ein    ($e000-$ffff)
05522  e13a   xxxxxxx1  BASIC ROM C 64 ein     ($a000-$bfff)
05523  e13a
05524  e13a  85 01             sta r8502        8502 CPU-Port Datenregister
05525  e13c
05526  e13c  a9 2f             lda #%00101111
05527  e13e
05528  e13e   0xxxxxxx  Input  unbenutzt
05529  e13e   x0xxxxxx  Input  CAPS-LOCK-Taste
05530  e13e   xx1xxxxx  Output Steuerung Kassettenmotor
05531  e13e   xxx0xxxx  Input  Tastenschalter Rekorder
05532  e13e   xxxx1xxx  Output Kassette serial out
05533  e13e   xxxxx1xx  Output Auswahl I/O - ROM        ($d000-$dfff)
05534  e13e   xxxxxx1x  Output Auswahl KERNAL ROM / RAM ($e000-$ffff)
05535  e13e   xxxxxxx1  Output Auswahl BASIC ROM / RAM  ($a000-$bfff)
05536  e13e
05537  e13e  85 00             sta d8502        8502 CPU-Port Datenrichtungsregister
05538  e140
05539  e140  ===> Test auf PAL / NTSC <===
05540  e140
05541  e140  a2 ff             ldx #255         vorbesetzen PAL/NTSC mit PAL
05542  e142  ad 11 d0   ioin01 lda viccr1       VIC 8564 Control-Register 1
05543  e145
05544  e145   0xxxxxxx  MSB (=Bit8) Raster-Wert Reg. 18 VIC
05545  e145
05546  e145  10 fb             bpl ioin01       warten auf MSB =1
05547  e147
05548  e147  a9 08      ioin02 lda #%00001000
05549  e149  cd 12 d0          cmp vicrwr       lo-Byte Raster-Wert
05550  e14c  90 06             bcc ioin03       PAL
05551  e14e
05552  e14e  ad 11 d0          lda viccr1       VIC 8564 Control-Register 1
05553  e151  30 f4             bmi ioin02       warten auf MSB =0
05554  e153
05555  e153  e8                inx              NTSC
05556  e154  8e 03 0a   ioin03 stx palnts       Flag PAL/NTSC
05557  e157
05558  e157  a9 00             lda #0
05559  e159  8d 37 0a          sta speed        CPU-Takt 1 MHz
05560  e15c  8d 39 0a          sta blnkng       ganzer BS Randfarbe
05561  e15f  8d 0a 0a          sta irqtmp+1     addrh Tape-IRQ rücksetzen
05562  e162  8d 3a 0a          sta holdof       Flag 'Sprites abschalten'
05563  e165  8d 36 0a          sta palcnt       Korrekturwert für Systemuhr 50 Hz
05564  e168  85 99             sta dfltn        Geräteadresse Tastatur
```

```
05565  e16a
05566  e16a  a9 03           lda #3          Geräteadresse Bildschirm
05567  e16c  85 9a           sta dflto       Vorgabe Ausgabe-Gerätenummer (3)
05568  e16e
05569  e16e  ===> 8564 VIC initialisieren <===
05570  e16e
05571  e16e  a2 30           ldx #48
05572  e170  bd c7 e2  ioivic lda victbl,x     Standardwerte VIC nach RESET
05573  e173  9d 00 d0        sta vic,x       Basisadresse VIC 8564
05574  e176  ca              dex
05575  e177  10 f7           bpl ioivic
05576  e179
05577  e179  ===> 8563 VDC initialisieren <===
05578  e179
05579  e179  a2 00           ldx #0
05580  e17b  20 dc e1        jsr vdcint      Standardwerte in VDC schreiben
05581  e17e
05582  e17e  ad 00 d6        lda vdcadr      VDC 8563 Adress-Register
05583  e181  29 07           and #%00000111
05584  e183
05585  e183   keine Dokumentation von Commodore erhältlich
05586  e183
05587  e183  f0 05           beq ioin05
05588  e185
05589  e185  a2 3b           ldx #59         Reg. 25 korrigieren
05590  e187  20 dc e1        jsr vdcint      Standardwerte in VDC schreiben
05591  e18a
05592  e18a  2c 03 0a  ioin05 bit palnts      Flag PAL/NTSC
05593  e18d  10 05           bpl ioin06
05594  e18f
05595  e18f  a2 3e           ldx #62         Reg. 4 und 7 korrigieren
05596  e191  20 dc e1        jsr vdcint      Standardwerte in VDC schreiben
05597  e194
05598  e194  ===> Test auf Char.gen bereits geladen <===
05599  e194
05600  e194  ad 04 0a  ioin06 lda initst      Flag Status nach RESET, BASIC-IRQ
05601  e197  30 15           bmi ioisid      Char.gen. bereits geladen
05602  e199
05603  e199  20 27 c0        jsr jint80      Charg.gen. in VDC laden
05604  e19c
05605  e19c  a9 80           lda #%10000000  Char.gen. in VDC geladen
05606  e19e  0d 04 0a        ora initst      Flag Status nach RESET, BASIC-IRQ
05607  e1a1  8d 04 0a        sta initst      Flag Status nach RESET, BASIC-IRQ
05608  e1a4
05609  e1a4  ===> Leerlauf-Warteschleife 0.3 s <===
05610  e1a4
05611  e1a4  a2 ff           ldx #$ff
05612  e1a6  a0 ff           ldy #$ff
05613  e1a8  88        ioidly dey
05614  e1a9  d0 fd           bne ioidly
05615  e1ab  ca              dex
05616  e1ac  d0 fa           bne ioidly
05617  e1ae
05618  e1ae
05619  e1ae  ===> SID 6581 initialisieren <===
05620  e1ae
05621  e1ae  a9 00     ioisid lda #%00000000  alle Register =0
05622  e1b0
```

```
05623  e1b0  a2 18            ldx #24
05624  e1b2  9d 00 d4  iosid1 sta sid,x       Basisadresse SID 6581
05625  e1b5  ca               dex
05626  e1b6  10 fa            bpl iosid1
05627  e1b8
05628  e1b8
05629  e1b8  ===> Test auf FSD <===
05630  e1b8
05631  e1b8  a2 01      iokeys ldx #%00000001   nur Raster-IRQ zulassen
05632  e1ba  8e 1a d0         stx vicier       VIC 8564 Interrupt Enable Register
05633  e1bd
05634  e1bd  ca               dex              =0
05635  e1be  8e 1c 0a         stx serial       Flag für Fast-Serial-Device
05636  e1c1  8e 0f 0a         stx enabl        CIA2 NMI-Flag
05637  e1c4
05638  e1c4  ca               dex               $ffff = 65ms
05639  e1c5  8e 06 dc         stx d1t2l        CIA1 TIMER B low
05640  e1c8  8e 07 dc         stx d1t2h        disable alle NMI
05641  e1cb
05642  e1cb  a2 11            ldx #%00010001
05643  e1cd
05644  e1cd   xxx1xxxx  FORCE LOAD
05645  e1cd   xxxxxxx1  Start TIMER B
05646  e1cd
05647  e1cd  8e 0f dc         stx d1crb        CIA1 Control-Register b
05648  e1d0
05649  e1d0  20 c3 e5         jsr spinp        Serial Port INPUT
05650  e1d3  20 d6 e5         jsr spout        Serial Port OUTPUT
05651  e1d6  20 c3 e5         jsr spinp        Serial Port INPUT
05652  e1d9  4c 4e e5         jmp clklo        CLOCK-Leitung lo
05653  e1dc
05654  e1dc
05655  e1dc  ===> Standardwerte in VDC schreiben <===
05656  e1dc
05657  e1dc  bc f8 e2   vdcint ldy vdctbl,x     Tab. Standardwerte VDC
05658  e1df  30 0d           . bmi stvdc1       Endezeichen Tab. gefunden
05659  e1e1
05660  e1e1  e8               inx
05661  e1e2  bd f8 e2         lda vdctbl,x     Tab. Standardwerte VDC
05662  e1e5
05663  e1e5  e8               inx
05664  e1e6  8c 00 d6         sty vdcadr       VDC 8563 Adress-Register
05665  e1e9  8d 01 d6         sta vdcdat       VDC 8563 Daten-Register
05666  e1ec  10 ee            bpl vdcint       Standardwerte in VDC schreiben
05667  e1ee
05668  e1ee
05669  e1ee  ===> Endezeichen Tab. gefunden <===
05670  e1ee
05671  e1ee  e8         stvdc1 inx
05672  e1ef  60               rts
05673  e1f0
05674  e1f0
05675  e1f0  ===> Test auf CBM-ID in RAM Bank 1 <===
05676  e1f0
05677  e1f0  a2 f5      secure ldx #<locker     CBM-ID
05678  e1f2  a0 ff            ldy #>locker     CBM-ID
05679  e1f4  86 c3            stx tmp2
05680  e1f6  84 c4            sty tmp2+1
```

```
05681  e1f8
05682  e1f8  a9 c3             lda #<tmp2
05683  e1fa  8d aa 02          sta fetchx       LDAVEC
05684  e1fd
05685  e1fd  ===> CBM-ID in RAM Bank 1 suchen <===
05686  e1fd
05687  e1fd  a0 02             ldy #2
05688  e1ff  a2 7f       tstmo1 ldx #%01111111
05689  e201
05690  e201   01xxxxxx  RAM Bank 1
05691  e201   xx111111  nur 64K RAM Bank 1
05692  e201
05693  e201  20 a2 02          jsr fetch        LDA (LDAVEC),y, Bank xr
05694  e204
05695  e204  d9 c4 e2          cmp cbmkey,y     Text ROM-ID
05696  e207  d0 1b             bne mod128       kein CBM-ID
05697  e209
05698  e209  88                dey
05699  e20a  10 f3             bpl tstmo1
05700  e20c
05701  e20c  ===> Initialisierungs Adresse RAM Bank 1 lesen <===
05702  e20c
05703  e20c  a2 f8             ldx #<system     Adresse Initialisierung akt. System
05704  e20e  a0 ff             ldy #>system     Adresse Initialisierung akt. System
05705  e210  86 c3             stx tmp2
05706  e212  84 c4             sty tmp2+1
05707  e214
05708  e214  a0 01             ldy #1
05709  e216  a2 7f       tstmo2 ldx #%01111111
05710  e218
05711  e218   01xxxxxx  RAM Bank 1
05712  e218   xx111111  nur 64K RAM Bank 1
05713  e218
05714  e218  20 a2 02          jsr fetch        LDA (LDAVEC),y, Bank xr
05715  e21b  99 02 00          sta bank,y       als ind. Sprung Adresse
05716  e21e
05717  e21e  88                dey
05718  e21f  10 f5             bpl tstmo2
05719  e221
05720  e221  6c 02 00          jmp (bank)       Initialisierung akt. System
05721  e224
05722  e224
05723  e224  ===> Initialisierung C 128 <===
05724  e224
05725  e224  a9 40       mod128 lda #%01000000
05726  e226
05727  e226   01xxxxxx  RAM Bank 1
05728  e226   xx00xxxx  KERNAL ROM     $c000-$ffff
05729  e226   xxxx00xx  BASIC ROM hi   $8000-$bfff
05730  e226   xxxxxx0x  BASIC ROM lo   $4000-$7fff
05731  e226   xxxxxxx0  I/O-Chips      $d000-$dfff
05732  e226
05733  e226  8d 00 ff          sta mmucr        MMU 8722 Config. Reg. (Kopie)
05734  e229
05735  e229  a9 24             lda #<mod128     Initialisierung C 128
05736  e22b  a0 e2             ldy #>mod128     Initialisierung C 128
05737  e22d  8d f8 ff          sta system       in RAM Bank 1 als Adresse
05738  e230  8c f9 ff          sty system+1     Initialisierung
```

```
05739  e233
05740  e233  ===> CBM-ID in RAM Bank 1 eintragen <===
05741  e233
05742  e233  a2 03             ldx #3
05743  e235  bd c3 e2   nomod1 lda cbmkey-1,x   Text ROM-ID
05744  e238  9d f4 ff          sta locker-1,x   CBM-ID
05745  e23b  ca                dex
05746  e23c  d0 f7             bne nomod1
05747  e23e
05748  e23e  8e 00 ff          stx mmucr        MMU 8722 Config. Reg. (Kopie)
05749  e241
05750  e241   00xxxxxx  RAM Bank 0
05751  e241   xx00xxxx  KERNAL ROM    $c000-$ffff
05752  e241   xxxx00xx  BASIC ROM hi  $8000-$bfff
05753  e241   xxxxxx0x  BASIC ROM lo  $4000-$7fff
05754  e241   xxxxxxx0  I/O-Chips     $d000-$dfff
05755  e241
05756  e241  60                rts
05757  e242
05758  e242
05759  e242  ===> Test auf Steck-Module <===
05760  e242
05761  e242  ad 05 d5   poll    lda mmumcr       MMU 8722 Modus Config. Reg.
05762  e245  29 30             and #%00110000
05763  e247  c9 30             cmp #%00110000
05764  e249
05765  e249   xx1xxxxx  Steckmodul GAME nicht belegt (C 64)
05766  e249   xxx1xxxx  Steckmodul EXROMIN nicht belegt (C 64)
05767  e249
05768  e249  f0 20             beq pol128       Test auf Steckmodul C 18
05769  e24b
05770  e24b
05771  e24b  ===> Kaltstart C 64 <===
05772  e24b
05773  e24b  a9 e3      c64mod lda #%11100011
05774  e24d
05775  e24d   11x00xxx  keine Wirkung
05776  e24d   xx1xxxxx  Kassettenmotor aus
05777  e24d   xxxxx0xx  Char.gen. ROM ein $d000-$dfff C 64
05778  e24d   xxxxxx1x  KERNAL ROM ein    $e000-$ffff C 64
05779  e24d   xxxxxxx1  BASIC ROM ein     $b000-$cfff C 64
05780  e24d
05781  e24d  85 01             sta r8502        8502 CPU-Port Datenregister
05782  e24f
05783  e24f  a9 2f             lda #%00101111   Definition I/O-Leitungen
05784  e251
05785  e251   7.......  unbenutzt
05786  e251   .6......  Input  CAPS LOCK Taste
05787  e251   ..5.....  Output Steuerung Kassettenmotor
05788  e251   ...4....  Input  Taste Kassettenrekorder
05789  e251   ....3...  Output seriell Kassette
05790  e251   .....2..  Output Auswahl I/O        / ROM $d000-$dfff C 64
05791  e251   ......1.  Output Auswahl Kernal-ROM / RAM $e000-$ffff C 64
05792  e251   .......0  Output Auswahl BASIC-ROM  / RAM $a000-$bfff C 64
05793  e251
05794  e251  85 00             sta d8502        8502 CPU-Port Datenrichtungsregister
05795  e253
```

```
05796  e253  ===> Routine zur MMU Configuration in ZP kopieren <===
05797  e253
05798  e253  a2 08            ldx #8
05799  e255  bd 62 e2   c64md1 lda c64beg-1,x   Zeichen aus Config. Routine
05800  e258  95 01            sta bank-1,x     nach ZP
05801  e25a  ca                dex
05802  e25b  d0 f8            bne c64md1
05803  e25d
05804  e25d  8e 30 d0          stx vicspd       VIC 8564 Speed-Register
05805  e260
05806  e260   xxxxxxx0  1 Megahertz Systemtakt
05807  e260
05808  e260  4c 02 00          jmp bank         Config. Routine in ZP aufrufen
05809  e263
05810  e263
05811  e263  ===> Routine wird in ZP kopiert und aufgerufen <===
05812  e263
05813  e263  a9 f7      c64beg lda #%11110111
05814  e265
05815  e265   1xxxxxxx  40 Zeichen BS
05816  e265   x1xxxxxx  C 64 - Modus
05817  e265   xx11011x  keine Wirkung
05818  e265   xxxxxxx1  CPU 8502
05819  e265
05820  e265  8d 05 d5          sta mmumcr       MMU 8722 Modus Config. Reg.
05821  e268
05822  e268  6c fc ff          jmp (sysres)     System RESET C 64
05823  e26b
05824  e26b
05825  e26b  ===> Test auf Steckmodul C 18 <===
05826  e26b
05827  e26b  a2 03      pol128 ldx #3           Zähler für je 2 PA aus 2 ROM Banks
05828  e26d  8e c0 0a          stx curbnk       zwischenspeichern
05829  e270
05830  e270  ===> Tab. physikal. Adr. (ROM Module) rücksetzen <===
05831  e270
05832  e270  a9 00             lda #0
05833  e272  9d c1 0a   pl128a sta pat,x        Tab. der Flags init. ROM-Module
05834  e275  ca                dex
05835  e276  10 fa             bpl pl128a
05836  e278
05837  e278  85 9e             sta ptr1         addrl FETVEC
05838  e27a
05839  e27a  a0 09      pl128b ldy #9           Zeiger auf CBM-ID
05840  e27c  ae c0 0a          ldx curbnk       Zeiger Adreßtab. Modultest
05841  e27f  bd bc e2          lda fkaddr,x     Adreßtab. ROM-ID's
05842  e282  85 9f             sta ptr2         addrh FETVEC
05843  e284
05844  e284  bd c0 e2          lda fkbank,x     Bank-Nr. ROM-Modul
05845  e287  85 02             sta bank         Banknummer
05846  e289
05847  e289  a6 02      pl128c ldx bank         Bank-Nr. ROM-Modul
05848  e28b  a9 9e             lda #<ptr1       Zeiger auf Modultest
05849  e28d  20 d0 f7          jsr indfet       LDA (FETVEC),y von Bank in xr
05850  e290
05851  e290  d9 bd e2          cmp fkaddr+1,y   yr auf CBM-ID
05852  e293  d0 21             bne pl128d       hier kein ROM-Modul
05853  e295
```

```
05854  e295  88                dey
05855  e296  c0 07             cpy #7          Untergrenze CBM-ID
05856  e298  b0 ef             bcs pl128c
05857  e29a
05858  e29a  ===> CBM-ID gefunden, <===
05859  e29a
05860  e29a  a6 02             ldx bank        Banknummer
05861  e29c  a9 9e             lda #<ptr1
05862  e29e  20 d0 f7          jsr indfet      LDA (FETVEC),y von Bank in xr
05863  e2a1
05864  e2a1  ae c0 0a          ldx curbnk      Zeiger in PAT
05865  e2a4  9d c1 0a          sta pat,x       Flag für ROM-Modul
05866  e2a7
05867  e2a7  c9 01             cmp #%00000001  Test Initial. ROM-Modul
05868  e2a9  d0 0b             bne pl128d      keine Initial. erforderlich
05869  e2ab
05870  e2ab  ===> ROM-Modul initialisieren<===
05871  e2ab
05872  e2ab  a5 9e             lda ptr1        addrl Modul-Initialisierung
05873  e2ad  a4 9f             ldy ptr2        addrh Modul-Initialisierung
05874  e2af  85 04             sta pclo        8502 CPU PC lo
05875  e2b1  84 03             sty pchi        8502 CPU PC hi
05876  e2b3
05877  e2b3  20 cd 02          jsr jsrfar      JSR, Bank xr
05878  e2b6  ce c0 0a   pl128d dec curbnk      akt. Bank Fkt.tasten
05879  e2b9  10 bf             bpl pl128b
05880  e2bb
05881  e2bb  60                rts
05882  e2bc
05883  e2bc
05884  e2bc  ===> Adreßtab. ROM-ID's <===
05885  e2bc
05886  e2bc  c0 80 c0   fkaddr .byte >$c000, >$8000, >$c000, >$8000
05887  e2bf  80
05888  e2c0
05889  e2c0
05890  e2c0  ===> Bank-Tab. ROM-ID's <===
05891  e2c0
05892  e2c0  04 04 08   fkbank .byte 4, 4, 8, 8
05893  e2c3  08
05894  e2c4
05895  e2c4
05896  e2c4  ===> Text ROM-ID <===
05897  e2c4
05898  e2c4  43 42 4d   cbmkey .byte 'cbm'
05899  e2c7
05900  e2c7
05901  e2c7  ===> Standardwerte VIC nach RESET <===
05902  e2c7
05903  e2c7  00 00      victbl .byte %00000000, %00000000 Reg.  0/1:
05904  e2c9   lo-Bytes Abszisse/Ordinate Sprite0
05905  e2c9
05906  e2c9  00 00             .byte %00000000, %00000000 Reg.  2/3:
05907  e2cb   lo-Bytes Abszisse/Ordinate Sprite1
05908  e2cb
05909  e2cb  00 00             .byte %00000000, %00000000 Reg.  4/5:
05910  e2cd   lo-Bytes Abszisse/Ordinate Sprite2
05911  e2cd
```

```
05912  e2cd  00 00             .byte %00000000, %00000000 Reg.  6/7:
05913  e2cf   lo-Bytes Abszisse/Ordinate Sprite3
05914  e2cf
05915  e2cf  00 00             .byte %00000000, %00000000 Reg.  8/9:
05916  e2d1   lo-Bytes Abszisse/Ordinate Sprite4
05917  e2d1
05918  e2d1  00 00             .byte %00000000, %00000000 Reg. 10/11:
05919  e2d3   lo-Bytes Abszisse/Ordinate Sprite5
05920  e2d3
05921  e2d3  00 00             .byte %00000000, %00000000 Reg. 12/13:
05922  e2d5   lo-Bytes Abszisse/Ordinate Sprite6
05923  e2d5
05924  e2d5  00 00             .byte %00000000, %00000000 Reg. 14/15:
05925  e2d7   lo-Bytes Abszisse/Ordinate Sprite7
05926  e2d7
05927  e2d7  00                .byte %00000000  Reg. 16:
05928  e2d8   MSB Koordinaten Sprite0-7
05929  e2d8
05930  e2d8  1b         .byte  %00011011        Reg. 17:
05931  e2d9
05932  e2d9   0xxxxxxx  MSB Raster-Wert Reg. 18
05933  e2d9   x0xxxxxx  Extended Color Text aus
05934  e2d9   xx0xxxxx  Bit Map Modus aus
05935  e2d9   xxx1xxxx  BS mit Rand (Blank-Modus aus)
05936  e2d9   xxxx1xxx  25-Zeilen BS
05937  e2d9   xxxxx011  Soft-Scroll vert. Dot-Bits
05938  e2d9
05939  e2d9  ff         .byte  %11111111        Reg. 18:
05940  e2da  lo-Byte Raster-Wert
05941  e2da
05942  e2da  00 00      .byte  %00000000, %00000000   Reg. 19/20:
05943  e2dc  Koordinaten Light-Pen (x/y)
05944  e2dc
05945  e2dc  00         .byte  %00000000        Reg. 21:
05946  e2dd  Sprites 7-0 aus (disabled)
05947  e2dd
05948  e2dd  08         .byte  %00001000        Reg. 22:
05949  e2de
05950  e2de   00xxxxxx  unbenutzt
05951  e2de   xx0xxxxx  immer 0 (reserviert Weiterentw.)
05952  e2de   xxx0xxxx  Priorität aus
05953  e2de   xxxx1xxx  40-Spalten / Zeile
05954  e2de   xxxxx000  Soft-Scroll hor. Dot-Bits
05955  e2de
05956  e2de  00         .byte  %00000000        Reg. 23:
05957  e2df   Sprites 7-0 vert. nicht verdoppeln
05958  e2df
05959  e2df  14         .byte  %00010100        Reg. 24:
05960  e2e0
05961  e2e0   0001xxxx  Adreßbits 13-10 Basisadresse BS-RAM
05962  e2e0   xxxx010x  Adreßbits 13-11 Basisadresse Chargen. ROM
05963  e2e0   xxxxxxx0  unbenutzt
05964  e2e0
05965  e2e0  ff         .byte  %11111111        Reg. 25:
05966  e2e1  wirkungslos, IRQ READ Register
05967  e2e1
05968  e2e1  01         .byte  %00000001        Reg. 26:
05969  e2e2
```

```
05970  e2e2   0000xxxx  unbenutzt
05971  e2e2   xxxx0xxx  Light-Pen IRQ disabled
05972  e2e2   xxxxx0xx  Sprite/Sprite Kollision-IRQ disabled
05973  e2e2   xxxxxx0x  Sprite/Hintergrund Kollision-IRQ disabled
05974  e2e2   xxxxxxx1  Raster IRQ enabled
05975  e2e2
05976  e2e2  00         .byte  %00000000        Reg. 27:
05977  e2e3  Priorität Sprites 7-0 aus
05978  e2e3
05979  e2e3  00         .byte  %00000000        Reg. 28:
05980  e2e4  Multi-Color-Modus Sprites 7-0 aus
05981  e2e4
05982  e2e4  00         .byte  %00000000        Reg. 29:
05983  e2e5   Sprites 7-0 hor. nicht verdoppeln
05984  e2e5
05985  e2e5  00         .byte  %00000000        Reg. 30:
05986  e2e6  Flags für Kollision Sprite/Sprites 7-0 rücksetzen
05987  e2e6
05988  e2e6  00         .byte  %00000000        Reg. 31:
05989  e2e7  Flags für Kollision Hintergrund/Sprites 7-0 rücksetzen
05990  e2e7
05991  e2e7  0d         .byte  13               Reg. 32:
05992  e2e8  Rahmen Farbe 14 (hellgrün)
05993  e2e8
05994  e2e8  0b         .byte  11               Reg. 33:
05995  e2e9  Hintergrund #0: Farbe 12 (dunkelgrau)
05996  e2e9
05997  e2e9  01         .byte  1                Reg. 34:
05998  e2ea  Hintergrund #1: Farbe 2 (weiß)
05999  e2ea
06000  e2ea  02         .byte  2                Reg. 35:
06001  e2eb  Hintergrund #2: Farbe 3 (rot)
06002  e2eb
06003  e2eb  03         .byte  3                Reg. 36:
06004  e2ec  Hintergrund #3: Farbe 4 (grün)
06005  e2ec
06006  e2ec  01         .byte  1                Reg. 37:
06007  e2ed  Sprite Multicolor #0: Farbe 2 (weiß)
06008  e2ed
06009  e2ed  02         .byte  2                Reg. 38:
06010  e2ee  Sprite Multicolor #1: Farbe 3 (rot)
06011  e2ee
06012  e2ee  00         .byte  0                Reg. 39:
06013  e2ef  Sprite0: Farbe 1 (schwarz)
06014  e2ef
06015  e2ef  01         .byte  1                Reg. 40:
06016  e2f0  Sprite1: Farbe 2 (weiß)
06017  e2f0
06018  e2f0  02         .byte  2                Reg. 41:
06019  e2f1  Sprite2: Farbe 3 (rot)
06020  e2f1
06021  e2f1  03         .byte  3                Reg. 42:
06022  e2f2  Sprite3: Farbe 4 (grün)
06023  e2f2
06024  e2f2  04         .byte  4                Reg. 43:
06025  e2f3  Sprite4: Farbe 5 (violett)
06026  e2f3
06027  e2f3  05         .byte  5                Reg. 44:
```

```
06028  e2f4  Sprite5: Farbe 6 (dunkelgrün)
06029  e2f4
06030  e2f4  06          .byte  6                Reg. 45:
06031  e2f5  Sprite6: Farbe 7 (blau)
06032  e2f5
06033  e2f5  07          .byte  7                Reg. 46:
06034  e2f6  Sprite7: Farbe 8 (gelb)
06035  e2f6
06036  e2f6  ff          .byte  %11111111        Reg. 49:
06037  e2f7  wirkungslos, da Input von Tastatur
06038  e2f7
06039  e2f7  fc          .byte  %11111100        Reg. 48:
06040  e2f8
06041  e2f8   1111110x ;unbenutzt
06042  e2f8   xxxxxxx0 ;CPU Takt 1 MHz
06043  e2f8
06044  e2f8
06045  e2f8  ===> Tab. Standardwerte VDC <===
06046  e2f8
06047  e2f8  00 7e       vdctbl .byte 0, 126     Register 0:
06048  e2fa  127 Zeichen zw. zwei hor. Synchronimpulsen
06049  e2fa
06050  e2fa  01 50              .byte 1, 80      Register 1:
06051  e2fc  80 Spalten pro BS-Zeile
06052  e2fc
06053  e2fc  02 66              .byte 2, 102     Register 2:
06054  e2fe  Position des hor. Synchronimpulses: 102
06055  e2fe
06056  e2fe  03 49              .byte 3, $49     Register 3:
06057  e300  Breite vertikaler Synchronimpuls = 4
06058  e300  Breite horizontaler Synchronimpuls = 9
06059  e300
06060  e300  04 20              .byte 4, 32      Register 4:
06061  e302  Gesamtzahl BS-Zeilen = 33
06062  e302
06063  e302  05 00              .byte 5, %00000000     Register 5:
06064  e304  Feinabgleich (Anzahl der Bildzeilen)
06065  e304
06066  e304  06 19              .byte 6, 25      Register 6:
06067  e306  Gesamtzahl dargestellter BS-Zeilen = 25
06068  e306
06069  e306  07 1d              .byte 7, 29      Register 7:
06070  e308                                      Pos. vert. Synchronimpuls = 29
06071  e308
06072  e308  08 00              .byte 8, %000000000    Register 8:
06073  e30a  kein Zeilensprung
06074  e30a
06075  e30a  09 07              .byte 9, 7       Register 9:
06076  e30c                                      Anzahl-1 der Bildzeilen/BS-Zeile = 7
06077  e30c
06078  e30c  0a 20              .byte 10, %00100000    Register 10:
06079  e30e  Cursor ausgeschaltet (=2), Startzeile im Zeichen = 0 (oben)
06080  e30e
06081  e30e  0b 07              .byte 11, 7      Register 11:
06082  e310                                      Cursor-Endzeile im Zeichen = 7
                                                 (unten)
06083  e310
06084  e310  0c 00              .byte 12, >vdcscn      Register 12 und
```

```
06085  e312  0d 00            .byte 13, <vdcscn      Register 13:
06086  e314  Startadresse Bildschirm-RAM
06087  e314
06088  e314  0e 00            .byte 14, >vdcscn      Register 14 und
06089  e316  0f 00            .byte 15, <vdcscn      Register 15:
06090  e318  Adresse der akt. Cursorposition im BS-RAM
06091  e318
06092  e318  14 08            .byte 20, >vdccol      Register 20 und
06093  e31a  15 00            .byte 21, <vdccol      Register 21:
06094  e31c  Startadresse Attribut-RAM
06095  e31c
06096  e31c  17 08            .byte 23, 8      Register 23:
06097  e31e  Anzahl Bildzeilen/BS-Zeile (ohne Zwischenraum)
06098  e31e
06099  e31e  18 20            .byte 24, %00100000    Register 24:
06100  e320  Block schreiben, REVERS aus, Blinken langsam, keine BS-Zeilen kopieren
06101  e320
06102  e320  19 40            .byte 25, %01000000    Register 25:
06103  e322  Textmodus, Vordergrundfarbe/Attribut laut Register 26
06104  e322  horizontaler Zwischenraum in Hintergrundfarbe,
06105  e322  Pixelgröße = 1 Dot-Takt, BS um 0 Pixel nach links schieben
06106  e322
06107  e322  1a f0            .byte 26, $f0    Register 26:
06108  e324  Vordergrundfarbe hellgrau = 15, Hintergrundfarbe schwarz = 0
06109  e324
06110  e324  1b 00            .byte 27, 0      Register 27:
06111  e326  Anzahl Zeichen/BS-Zeile
06112  e326
06113  e326  1c 20            .byte 28, %00100000    Register 28:
06114  e328  Adreßbits A15-A13 Zeichengenerator: $20, DRAM 4416
06115  e328
06116  e328  1d 07            .byte 29, 7      Register 29:
06117  e32a  Bildzeilen-Zähler für Unterstreichen
06118  e32a
06119  e32a  22 7d            .byte 34, 125    Register 34:
06120  e32c  Anzahl Block-Write- bzw. Block-Copy-Zyklen = 125
06121  e32c
06122  e32c  23 64            .byte 35, 100    Register 35:
06123  e32e  Zeichen von Anfang akt. BS-Zeile bis pos. Flanke Display-Enable-Pins
06124  e32e
06125  e32e  24 05            .byte 36, 5      Register 36:
06126  e330  Zeichen von Anfang akt. BS-Zeile bis neg. Flanke Display-Enable-Pins
06127  e330
06128  e330  16 78            .byte 22, $78    Register 22:
06129  e332  Zeichenbreite in Pixel (7/8, mit/ohne Zwischenraum)
06130  e332  ff               .byte $ff        Flag Ende Tabelle
06131  e333
06132  e333
06133  e333  ===>  offset 59  <===
06134  e333
06135  e333  19 47            .byte 25, %01000111    Register 25:
06136  e335  Textmodus, Vordergrundfarbe/Attribut laut Register 26
06137  e335  horizontaler Zwischenraum in Hintergrundfarbe,
06138  e335  Pixelgröße = 1 Dot-Takt, BS um 7 Pixel nach links schieben
06139  e335
06140  e335  ff               .byte $ff        Flag Ende Tabelle
06141  e336
06142  e336
```

```
06143  e336  ===>  offset 62  <===
06144  e336
06145  e336  04 27            .byte 4, 39      Register 4:
06146  e338  Gesamtzahl BS-Zeilen = 40
06147  e338
06148  e338  07 20            .byte 7, 32      Register 7:
06149  e33a                                    Pos. vert. Synchronimpuls = 32
06150  e33a
06151  e33a  ff               .byte $ff        Flag Ende Tabelle
06152  e33b
06153  e33b
06154  e33b                   .end
06155  e33b                   .lib c128.floppy
06156  e33b
```

4.1.12 Floppy (Serial Device)

```
06158  e33b
06159  e33b
06160  e33b  *******   Der serielle IEC-Bus des C 128 arbeitet    *******
06161  e33b  *******    - wie der echte, parallele IEC-Bus -      *******
06162  e33b  *******   mit negativer Logik, d.h. 0 = hi, 1 = lo.  *******
06163  e33b
06164  e33b
06165  e33b  ===> Ausgabe TALK auf IEC-Bus <===
06166  e33b
06167  e33b  09 40      talk   ora #%01000000   Bit6 in Geräteadresse = TALK
06168  e33d  2c               .byte skip2
06169  e33e
06170  e33e
06171  e33e  ===> Ausgabe LISTEN auf IEC-Bus <===
06172  e33e
06173  e33e  09 20      listn  ora #%00100000   Bit5 in Geräteadresse = LISTEN
06174  e340  20 ec e7          jsr rsp232       Test / Warten auf Ende RS 232 I/O
06175  e343
06176  e343
06177  e343  ===> Zeichen mit ATN ausgeben <===
06178  e343
06179  e343  48         list1  pha
06180  e344
06181  e344  24 94             bit c3po         IEC-Puffer-Flag
06182  e346  10 0a             bpl list2
06183  e348
06184  e348  38               sec
06185  e349  66 a3             ror pcntr
06186  e34b  20 8c e3          jsr isour        ein Byte auf IEC-Bus ausgeben
06187  e34e  46 94             lsr c3po         IEC-Puffer-Flag
06188  e350  46 a3             lsr pcntr
06189  e352  68         list2  pla
06190  e353  85 95             sta bsour        Zeichen für IEC-Ausgabe
06191  e355
06192  e355  20 73 e5          jsr dissei       Test auf / Sprites ausschalten
06193  e358  20 57 e5          jsr datahi       DATA-OUT hi
06194  e35b
06195  e35b  ad 00 dd          lda d2pra        CIA2 Datenport A
06196  e35e  29 08             and #%00001000   ATN OUT maskieren
```

```
06197  e360  d0 12            bne list4        ATN lo
06198  e362
06199  e362  20 d6 e5         jsr spout        Serial Port OUTPUT
06200  e365  a9 ff            lda #%11111111   8 Bits 1
06201  e367  8d 0c dc         sta d1sdr        CIA1 Serial Data Register
06202  e36a
06203  e36a  20 bc e5         jsr slowb        Lesen FSD nach Ende Senden
06204  e36d
06205  e36d  ===> Warteschleife 0.1 ms <===
06206  e36d
06207  e36d  8a               txa
06208  e36e  a2 14            ldx #20
06209  e370  ca         list3 dex
06210  e371  d0 fd            bne list3
06211  e373  aa               tax
06212  e374
06213  e374  ad 00 dd   list4 lda d2pra        CIA2 Datenport A
06214  e377  09 08            ora #%00001000   ATN OUT lo
06215  e379  8d 00 dd         sta d2pra        CIA2 Datenport A
06216  e37c
06217  e37c
06218  e37c  ===> Zeichen mit ATN ausgeben <===
06219  e37c
06220  e37c  20 73 e5   isoura jsr dissei      Test auf / Sprites ausschalten
06221  e37f  20 4e e5         jsr clklo        CLOCK-Leitung lo
06222  e382  20 57 e5         jsr datahi       DATA-OUT hi
06223  e385
06224  e385  ===> Warteschleife 1 ms <===
06225  e385
06226  e385  8a               txa
06227  e386  a2 b8            ldx #184
06228  e388  ca         isora1 dex
06229  e389  d0 fd            bne isora1
06230  e38b  aa               tax
06231  e38c
06232  e38c
06233  e38c  ===> ein Byte auf IEC-Bus ausgeben <===
06234  e38c
06235  e38c  20 73 e5   isour  jsr dissei      Test auf / Sprites ausschalten
06236  e38f  20 57 e5         jsr datahi       DATA-OUT hi
06237  e392  20 69 e5         jsr debpia       DATA IN Carry, CLOCK IN Bit7
06238  e395  90 03            bcc isr01        DATA IN hi
06239  e397
06240  e397  4c 28 e4         jmp nodev        'device not present' in Status
06241  e39a
06242  e39a
06243  e39a  2c 0d dc   isr01  bit d1icr        IRQ-Flag Register rücksetzen
06244  e39d  20 45 e5         jsr clkhi        CLOCK-Leitung hi
06245  e3a0
06246  e3a0  24 a3            bit pcntr
06247  e3a2  10 0a            bpl isr04
06248  e3a4
06249  e3a4
06250  e3a4  ===> Warten auf DATA IN = lo <===
06251  e3a4
06252  e3a4  20 69 e5   isr02  jsr debpia       DATA IN Carry, CLOCK IN Bit7
06253  e3a7  90 fb            bcc isr02        DATA IN hi
06254  e3a9
```

```
06255  e3a9
06256  e3a9  ===> Warten auf DATA IN = hi <===
06257  e3a9
06258  e3a9  20 69 e5   isr03  jsr debpia       DATA IN Carry, CLOCK IN Bit7
06259  e3ac  b0 fb             bcs isr03        DATA IN lo
06260  e3ae
06261  e3ae  ad 00 dd   isr04  lda d2pra        Floppy-Port
06262  e3b1  cd 00 dd          cmp d2pra        "entprellen"
06263  e3b4  d0 f8             bne isr04
06264  e3b6
06265  e3b6  48                pha
06266  e3b7
06267  e3b7  ad 0d dc          lda d1icr        CIA1 Interrupt Control-Register
06268  e3ba  29 08             and #%00001000   IRQ-Bit Serial Port full/empty
06269  e3bc  f0 05             beq isr05
06270  e3be
06271  e3be  ===> SDR IRQ: Fast Serial Device (FSD) <===
06272  e3be
06273  e3be  a9 c0             lda #%11000000
06274  e3c0  8d 1c 0a          sta serial       Flag für Fast-Serial-Device
06275  e3c3
06276  e3c3  68         isr05  pla              Floppy-Port vom Stapel
06277  e3c4  10 e8             bpl isr04        DATA IN hi
06278  e3c6  09 10             ora #%00010000   DATA OUT lo
06279  e3c8  8d 00 dd          sta d2pra        CIA2 Datenport A
06280  e3cb
06281  e3cb  29 08             and #%00001000   ATN OUT maskieren
06282  e3cd  d0 13             bne isr06        ATN lo
06283  e3cf
06284  e3cf  2c 1c 0a          bit serial       Flag für Fast-Serial-Device
06285  e3d2  10 0e             bpl isr06        Ausgabe Zeichen langsame Floppy
06286  e3d4
06287  e3d4  ===> Ausgabe Zeichen Fast Serial Device <===
06288  e3d4
06289  e3d4  20 d6 e5          jsr spout        Serial Port OUTPUT
06290  e3d7  a5 95             lda bsour        Zeichen für IEC-Ausgabe
06291  e3d9  8d 0c dc          sta d1sdr        CIA1 Serial Data Register
06292  e3dc
06293  e3dc  20 bc e5          jsr slowb        Lesen FSD nach Ende Senden
06294  e3df  4c 12 e4          jmp isr08
06295  e3e2
06296  e3e2
06297  e3e2  ===> Ausgabe Zeichen langsame Floppy <===
06298  e3e2
06299  e3e2  a9 08      isr06  lda #8
06300  e3e4  85 a5             sta cntdn        Bitzähler initialisieren
06301  e3e6
06302  e3e6
06303  e3e6  ===> Ausgabe nächstes Datenbit <===
06304  e3e6
06305  e3e6  ad 00 dd   isr07  lda d2pra        Floppy-Port
06306  e3e9  cd 00 dd          cmp d2pra        "entprellen"
06307  e3ec  d0 f8             bne isr07        Ausgabe nächstes Datenbit
06308  e3ee
06309  e3ee  0a                asl a            Bit7 DATA IN ins Carry
06310  e3ef  90 34             bcc frmerr       DATA IN hi, timeout
06311  e3f1
06312  e3f1  66 95             ror bsour        nächstes Ausgabebit ins Carry
```

```
06313  e3f3  b0 05           bcs isrhi        Bit ist 1 (hi)
06314  e3f5
06315  e3f5  ===> Ausgabe ein Bit lo <===
06316  e3f5
06317  e3f5  20 60 e5        jsr datalo       DATA-OUT lo
06318  e3f8  d0 03           bne isrclk       immer
06319  e3fa
06320  e3fa
06321  e3fa  ===> Ausgabe Bit-Wert 1 (hi) <===
06322  e3fa
06323  e3fa  20 57 e5  isrhi  jsr datahi       DATA-OUT hi
06324  e3fd  20 45 e5  isrclk jsr clkhi        CLOCK-Leitung hi
06325  e400  ea               nop
06326  e401  ea               nop
06327  e402  ea               nop
06328  e403  ea               nop
06329  e404
06330  e404  ad 00 dd         lda d2pra        CIA2 Datenport A
06331  e407  29 df            and #%11011111   DATA OUT hi
06332  e409  09 10            ora #%00010000   CLOCK OUT lo
06333  e40b  8d 00 dd         sta d2pra        CIA2 Datenport A
06334  e40e
06335  e40e  c6 a5            dec cntdn
06336  e410  d0 d4            bne isr07        Ausgabe nächstes Datenbit
06337  e412
06338  e412  8a        isr08  txa
06339  e413  48               pha
06340  e414
06341  e414  ===> Warteschleife bis ca. 1 ms <===
06342  e414
06343  e414  a2 22            ldx #34
06344  e416  20 69 e5  isr09  jsr debpia       DATA IN Carry, CLOCK IN Bit7
06345  e419  b0 05            bcs isr10
06346  e41b
06347  e41b  68               pla
06348  e41c  aa               tax
06349  e41d  4c 9f e5         jmp discli       Test auf / Sprites einschalten
06350  e420
06351  e420
06352  e420  ca        isr10  dex
06353  e421  d0 f3            bne isr09
06354  e423
06355  e423  68               pla
06356  e424  aa               tax
06357  e425
06358  e425
06359  e425  ===> 'Timeout' in Status <===
06360  e425
06361  e425  a9 03      frmerr lda #%00000011   Bits 0/1: Timeout Schreiben/Lesen
06362  e427  2c               .byte skip2
06363  e428
06364  e428
06365  e428  ===> 'device not present' in Status <===
06366  e428
06367  e428  a9 80       nodev  lda #%10000000  Bit7: device not present
06368  e42a
06369  e42a
```

```
06370  e42a  ===> ac-Bits im Status setzen <===
06371  e42a
06372  e42a  48          csberr pha
06373  e42b
06374  e42b  ad 1c 0a           lda serial       Flag für Fast-Serial-Device
06375  e42e  29 7f              and #%01111111
06376  e430  8d 1c 0a           sta serial       Flag für Fast-Serial-Device
06377  e433
06378  e433  68                 pla
06379  e434  20 57 f7           jsr udst         ac-Bits in Status setzen
06380  e437  20 9f e5           jsr discli       Test auf / Sprites einschalten
06381  e43a  18                 clc
06382  e43b  4c 35 e5           jmp dlabye
06383  e43e
06384  e43e
06385  e43e  ===> Zeichen vom IEC-Bus nach ac <===
06386  e43e
06387  e43e  20 73 e5    acptr  jsr dissei       Test auf / Sprites ausschalten
06388  e441
06389  e441  a9 00              lda #0
06390  e443  85 a5              sta cntdn        Zähler für 2 Warteschleifen
06391  e445
06392  e445  2c 0d dc           bit d1icr        IRQ-Flags rücksetzen
06393  e448  8a                 txa
06394  e449  48                 pha
06395  e44a
06396  e44a  20 45 e5           jsr clkhi        CLOCK-Leitung hi
06397  e44d
06398  e44d
06399  e44d  ===> Warten auf CLOCK IN lo <===
06400  e44d
06401  e44d  20 69 e5    acp00a jsr debpia       DATA IN Carry, CLOCK IN Bit7
06402  e450  10 fb              bpl acp00a       CLOCK IN hi
06403  e452
06404  e452
06405  e452  ===> 0.25 ms warten auf DATA IN hi <===
06406  e452
06407  e452  a2 0d       acp0a1 ldx #13
06408  e454
06409  e454  ad 00 dd           lda d2pra        CIA2 Datenport A
06410  e457  29 df              and #%11011111   DATA OUT hi
06411  e459  8d 00 dd           sta d2pra        CIA2 Datenport A
06412  e45c
06413  e45c  ad 00 dd    acp0a2 lda d2pra        Floppy-Port
06414  e45f  cd 00 dd           cmp d2pra        "entprellen"
06415  e462  d0 f8              bne acp0a2
06416  e464
06417  e464  0a                 asl a            CLOCK IN nach Bit7
06418  e465  10 1d              bpl acp01        CLOCK IN hi
06419  e467
06420  e467  ca                 dex
06421  e468  d0 f2              bne acp0a2
06422  e46a
06423  e46a  ===> Fehler Timeout beim Lesen <===
06424  e46a
06425  e46a  a5 a5              lda cntdn
06426  e46c  d0 0f              bne acp0a3       Abbruch wegen Timeout
06427  e46e
```

```
06428  e46e  20 60 e5          jsr datalo       DATA-OUT lo
06429  e471  20 45 e5          jsr clkhi        CLOCK-Leitung hi
06430  e474
06431  e474  a9 40             lda #%01000000   Bit6: EOF-Status setzen
06432  e476  20 57 f7          jsr udst         ac-Bits in Status setzen
06433  e479
06434  e479  e6 a5             inc cntdn
06435  e47b  d0 d5             bne acp0a1       immer
06436  e47d
06437  e47d
06438  e47d  ===> Abbruch wegen Timeout <===
06439  e47d
06440  e47d  68         acp0a3 pla
06441  e47e  aa                tax
06442  e47f  a9 02             lda #%00000010   Bit1: Timeout Lesen
06443  e481  4c 2a e4          jmp csberr       ac-Bits im Status setzen
06444  e484
06445  e484
06446  e484  a2 08      acp01  ldx #8           Zähler für acht Datenbits
06447  e486
06448  e486
06449  e486
06450  e486  ===> Warten auf CLOCK IN lo/IRQ <===
06451  e486
06452  e486  ad 0d dc   acp03  lda d1icr        CIA1 Interrupt Control-Register
06453  e489  29 08             and #%00001000   IRQ-Flag Serial Port full/empty
06454  e48b  d0 28             bne acp04        gesetzt, Fast Serial Device
06455  e48d
06456  e48d  ad 00 dd   acp03a lda d2pra        Floppy-Port
06457  e490  cd 00 dd          cmp d2pra        "entprellen"
06458  e493  d0 f8             bne acp03a
06459  e495
06460  e495  0a                asl a            CLOCK IN nach Bit7, DATA IN Carry
06461  e496  10 ee             bpl acp03        CLOCK IN hi
06462  e498
06463  e498
06464  e498  ===> Datenbit ins Datenbyte <===
06465  e498
06466  e498  66 a4      acp03b ror bsour1       DATA IN ins Datenbyte
06467  e49a
06468  e49a
06469  e49a  ===> Warten auf CLOCK IN hi <===
06470  e49a
06471  e49a  ad 00 dd   acp03c lda d2pra        Floppy-Port
06472  e49d  cd 00 dd          cmp d2pra        "entprellen"
06473  e4a0  d0 f8             bne acp03c       Warten auf CLOCK IN hi
06474  e4a2
06475  e4a2  0a                asl a            CLOCK IN nach Bit7
06476  e4a3  30 f5             bmi acp03c       CLOCK IN lo
06477  e4a5
06478  e4a5  ca                dex
06479  e4a6  f0 17             beq acp04a       Abschluß Lesen Floppy
06480  e4a8
06481  e4a8
06482  e4a8  ===> Warten auf CLOCK IN lo <===
06483  e4a8
06484  e4a8  ad 00 dd   acp03d lda d2pra        CIA2 Datenport A
06485  e4ab  cd 00 dd          cmp d2pra        CIA2 Datenport A
```

```
06486  e4ae  d0 f8            bne acp03d      Warten auf CLOCK IN lo
06487  e4b0
06488  e4b0  0a              asl a           CLOCK IN nach Bit7, DATA IN Carry
06489  e4b1  10 f5           bpl acp03d      Warten auf CLOCK IN lo
06490  e4b3  30 e3           bmi acp03b      Datenbit ins Datenbyte
06491  e4b5
06492  e4b5
06493  e4b5  ===> Zeichen vom FSD nach ac <===
06494  e4b5
06495  e4b5  ad 0c dc   acp04 lda d1sdr       CIA1 Serial Data Register
06496  e4b8  85 a4            sta bsour1
06497  e4ba
06498  e4ba  a9 c0            lda #%11000000
06499  e4bc  8d 1c 0a         sta serial      Flag für Fast-Serial-Device
06500  e4bf
06501  e4bf
06502  e4bf  ===> Abschluß Lesen Floppy <===
06503  e4bf
06504  e4bf  68         acp04a pla
06505  e4c0  aa               tax
06506  e4c1  20 60 e5         jsr datalo      DATA-OUT lo
06507  e4c4
06508  e4c4  24 90            bit status      Bit6 testen (1=EOF)
06509  e4c6  50 03            bvc acp04b      kein EOF
06510  e4c8
06511  e4c8  20 38 e5         jsr dladlh
06512  e4cb  20 9f e5   acp04b jsr discli      Test auf / Sprites einschalten
06513  e4ce  a5 a4            lda bsour1
06514  e4d0  18               clc             Flag ok
06515  e4d1  60               rts
06516  e4d2
06517  e4d2
06518  e4d2  ===> Ausgabe Sek.Adr. nach LISTEN auf IEC-Bus <===
06519  e4d2
06520  e4d2  85 95      secnd sta bsour       Zeichen für IEC-Ausgabe
06521  e4d4
06522  e4d4  20 7c e3         jsr isoura      Zeichen mit ATN ausgeben
06523  e4d7
06524  e4d7
06525  e4d7  ===> NOT ATN HI nach Ausgabe Sek.Adr. auf IEC-Bus <===
06526  e4d7
06527  e4d7  ad 00 dd   scatn lda d2pra       CIA2 Datenport A
06528  e4da  29 f7            and #%11110111  ATN OUT hi
06529  e4dc  8d 00 dd         sta d2pra       CIA2 Datenport A
06530  e4df
06531  e4df  60               rts
06532  e4e0
06533  e4e0
06534  e4e0  ===> Ausgabe Sek.Adr. nach TALK auf IEC-Bus <===
06535  e4e0
06536  e4e0  85 95      tksa  sta bsour       Zeichen für IEC-Ausgabe
06537  e4e2  20 7c e3         jsr isoura      Zeichen mit ATN ausgeben
06538  e4e5
06539  e4e5  24 90            bit status      Bit7 testen (1=device not present)
06540  e4e7  30 4c            bmi dlabye      device not present
06541  e4e9
06542  e4e9
```

```
06543  e4e9  ===> NOT ATN HI nach Ausgabe TALK auf IEC-Bus <===
06544  e4e9
06545  e4e9  20 73 e5   tkatn  jsr dissei       Test auf / Sprites ausschalten
06546  e4ec  20 60 e5          jsr datalo       DATA-OUT lo
06547  e4ef  20 d7 e4          jsr scatn        NOT ATN HI nach Ausgabe Sek.Adr. auf
                                                IEC-Bus
06548  e4f2  20 45 e5          jsr clkhi        CLOCK-Leitung hi
06549  e4f5
06550  e4f5
06551  e4f5  ===> Warten auf CLOCK IN hi <===
06552  e4f5
06553  e4f5  ad 00 dd   tkatn1 lda d2pra        Floppy-Port
06554  e4f8  cd 00 dd          cmp d2pra        "entprellen"
06555  e4fb  d0 f8             bne tkatn1       Warten auf CLOCK IN hi
06556  e4fd
06557  e4fd  0a                asl a            CLOCK IN nach Bit7
06558  e4fe  30 f5             bmi tkatn1       CLOCK IN lo
06559  e500
06560  e500  4c 9f e5          jmp discli       Test auf / Sprites einschalten
06561  e503
06562  e503
06563  e503
06564  e503  ===> Ausgabe ac auf IEC-Bus <===
06565  e503
06566  e503  24 94      ciout  bit c3po         IEC-Puffer-Flag
06567  e505  30 05             bmi ci2          noch ein Zeichen auszugeben
06568  e507
06569  e507  38                sec              Flag setzen:
06570  e508  66 94             ror c3po         noch ein Zeichen auszugeben
06571  e50a  d0 05             bne ci4          immer
06572  e50c
06573  e50c
06574  e50c  48        ci2     pha              neues Zeichen auf Stapel
06575  e50d
06576  e50d  20 8c e3          jsr isour        letztes Zeichen ausgeben
06577  e510  68                pla              neues Zeichen vom Stapel
06578  e511  85 95      ci4    sta bsour        Zeichen für IEC-Ausgabe
06579  e513
06580  e513  18                clc
06581  e514  60                rts
06582  e515
06583  e515
06584  e515  ===> Ausgabe UNTALK auf IEC-Bus <===
06585  e515
06586  e515  20 73 e5   untlk  jsr dissei       Test auf / Sprites ausschalten
06587  e518  20 4e e5          jsr clklo        CLOCK-Leitung lo
06588  e51b
06589  e51b  ad 00 dd          lda d2pra        CIA2 Datenport A
06590  e51e  09 08             ora #%00001000   ATN OUT lo
06591  e520  8d 00 dd          sta d2pra        CIA2 Datenport A
06592  e523
06593  e523  a9 5f             lda #95          Code UNTALK
06594  e525  2c                .byte skip2
06595  e526
06596  e526
06597  e526  ===> Ausgabe UNLISTEN auf IEC-Bus <===
06598  e526
06599  e526  a9 3f      nunlsn lda #63          Code UNLISTEN
```

```
06600  e528  48              pha
06601  e529
06602  e529  ad 1c 0a         lda serial      Flag für Fast-Serial-Device
06603  e52c  29 7f            and #%01111111
06604  e52e  8d 1c 0a         sta serial      Flag für Fast-Serial-Device
06605  e531
06606  e531  68              pla
06607  e532  20 43 e3         jsr list1       Zeichen mit ATN ausgeben
06608  e535  20 d7 e4   dlabye jsr scatn       NOT ATN HI nach Ausgabe Sek.Adr. auf
                                               IEC-Bus
06609  e538  8a         dladlh txa
06610  e539
06611  e539  ===> Warteschleife ca. 0.1 ms <===
06612  e539
06613  e539  a2 0a            ldx #10
06614  e53b  ca         dlad00 dex
06615  e53c  d0 fd            bne dlad00
06616  e53e
06617  e53e  aa                tax
06618  e53f  20 45 e5          jsr clkhi       CLOCK-Leitung hi
06619  e542  4c 57 e5          jmp datahi      DATA-OUT hi
06620  e545
06621  e545
06622  e545  ===> CLOCK-Leitung hi <===
06623  e545
06624  e545  ad 00 dd   clkhi  lda d2pra       CIA2 Datenport A
06625  e548  29 ef            and #%11101111  CLOCK OUT hi
06626  e54a  8d 00 dd          sta d2pra       CIA2 Datenport A
06627  e54d
06628  e54d  60                rts
06629  e54e
06630  e54e
06631  e54e  ===> CLOCK-Leitung lo <===
06632  e54e
06633  e54e  ad 00 dd   clklo  lda d2pra       CIA2 Datenport A
06634  e551  09 10            ora #%00010000  CLOCK OUT lo
06635  e553  8d 00 dd          sta d2pra       CIA2 Datenport A
06636  e556
06637  e556  60                rts
06638  e557
06639  e557
06640  e557  ===> DATA-OUT hi <===
06641  e557
06642  e557  ad 00 dd   datahi lda d2pra       CIA2 Datenport A
06643  e55a  29 df            and #%11011111  DATA OUT hi
06644  e55c  8d 00 dd          sta d2pra       CIA2 Datenport A
06645  e55f
06646  e55f  60                rts
06647  e560
06648  e560
06649  e560  ===> DATA-OUT lo <===
06650  e560
06651  e560  ad 00 dd   datalo lda d2pra       CIA2 Datenport A
06652  e563  09 20            ora #%00100000  DATA OUT lo
06653  e565  8d 00 dd          sta d2pra       CIA2 Datenport A
06654  e568
06655  e568  60                rts
06656  e569
```

```
06657  e569
06658  e569
06659  e569  ===> DATA IN Carry, CLOCK IN Bit7 <===
06660  e569
06661  e569  ad 00 dd   debpia lda d2pra        Floppy-Datenport
06662  e56c  cd 00 dd          cmp d2pra        "entprellen"
06663  e56f  d0 f8             bne debpia       DATA IN Carry, CLOCK IN Bit7
06664  e571
06665  e571  0a                asl a            DATA IN Carry, CLOCK IN Bit7
06666  e572  60                rts
06667  e573
06668  e573
06669  e573  ===> Test auf / Sprites ausschalten <===
06670  e573
06671  e573  78         dissei sei
06672  e574
06673  e574
06674  e574  ===> Test auf / Sprites ausschalten <===
06675  e574
06676  e574  2c 3a 0a   getsp  bit holdof       Flag 'Sprites abschalten'
06677  e577  30 25             bmi dismo3       Bit7 =1: Sprites nicht
                                                ein-/ausschalten
06678  e579
06679  e579  2c 37 0a          bit speed        Speicher für akt. CPU-Takt
06680  e57c  30 20             bmi dismo3       Bit7 gesetzt, Sprites ausgeschaltet
06681  e57e
06682  e57e  ad 30 d0          lda vicspd       VIC 8564 Speed-Register
06683  e581
06684  e581   11111100  bei 1 MHz CPU-Takt
06685  e581   11111101  bei 2 MHz CPU-Takt
06686  e581
06687  e581  8d 37 0a          sta speed        Bit7 gesetzt, Flag Sprites aus
06688  e584
06689  e584  ad 15 d0          lda vicmoe       VIC 8564 Sprite Enable Register
06690  e587  8d 38 0a          sta sprits       Sprite Einschaltzustand retten
06691  e58a
06692  e58a  a9 00             lda #%00000000   Sprites 7-0 ausschalten
06693  e58c  8d 15 d0          sta vicmoe       VIC 8564 Sprite Enable Register
06694  e58f
06695  e58f  8d 30 d0          sta vicspd       CPU-Takt 1 MHz einschalten
06696  e592
06697  e592  ad 38 0a          lda sprits       Speicher Einschaltzustand Sprites
06698  e595  f0 07             beq dismo3       es waren keine Sprites eingeschaltet
06699  e597
06700  e597  8a                txa              xr retten
06701  e598
06702  e598  ===> Warteschleife 1.3 ms <===
06703  e598
06704  e598  a2 00             ldx #0
06705  e59a  ca         dismo2 dex
06706  e59b  d0 fd             bne dismo2
06707  e59d
06708  e59d  aa                tax              xr zurück
06709  e59e  60         dismo3 rts
06710  e59f
06711  e59f
```

```
06712  e59f  ===> Test auf / Sprites einschalten <===
06713  e59f
06714  e59f  2c 3a 0a   discli bit holdof       Flag 'Sprites abschalten'
06715  e5a2  30 16             bmi enamo1       Bit7 =1: Sprites nicht
                                                ein-/ausschalten
06716  e5a4
06717  e5a4  2c 37 0a          bit speed        Speicher für akt. CPU-Takt
06718  e5a7  10 11             bpl enamo1       Sprites bereits eingeschaltet
06719  e5a9
06720  e5a9  ad 38 0a          lda sprits       Speicher Einschaltzustand Sprites
06721  e5ac  8d 15 d0          sta vicmoe       Sprites wieder einschalten
06722  e5af
06723  e5af  ad 37 0a          lda speed        Speicher für akt. CPU-Takt
06724  e5b2  8d 30 d0          sta vicspd       alten CPU-Takt wieder einschalten
06725  e5b5
06726  e5b5  a9 00             lda #%00000000   Bit7 rücksetzen (Sprites
                                                eingeschaltet)
06727  e5b7  8d 37 0a          sta speed        Speicher für akt. CPU-Takt
06728  e5ba
06729  e5ba  58         enamo1 cli
06730  e5bb  60                rts
06731  e5bc
06732  e5bc
06733  e5bc
06734  e5bc  ===> Lesen FSD nach Ende Senden <===
06735  e5bc
06736  e5bc  ad 0d dc   slowb  lda d1icr        CIA1 Interrupt Control-Register
06737  e5bf  29 08             and #%00001000   IRQ-Bit Serial maskieren
06738  e5c1  f0 f9             beq slowb        warten bis gesetzt
06739  e5c3
06740  e5c3
06741  e5c3  ===> Serial Port INPUT <===
06742  e5c3
06743  e5c3  ad 0e dc   spinp  lda d1cra        CIA1 Control-Register a
06744  e5c6  29 80             and #%10000000   TODIN erhalten
06745  e5c8  09 08             ora #%00001000   ONE SHOT
06746  e5ca  8d 0e dc          sta d1cra        CIA1 Control-Register a
06747  e5cd
06748  e5cd  ad 05 d5          lda mmumcr       MMU 8722 Modus Config. Reg.
06749  e5d0  29 f7             and #%11110111   FSDIR-Bit rücksetzen
06750  e5d2  8d 05 d5          sta mmumcr       MMU 8722 Modus Config. Reg.
06751  e5d5
06752  e5d5  60                rts
06753  e5d6
06754  e5d6
06755  e5d6  ===> Serial Port OUTPUT <===
06756  e5d6
06757  e5d6  ad 05 d5   spout  lda mmumcr       MMU 8722 Modus Config. Reg.
06758  e5d9  09 08             ora #%00001000   FSDIR-Bit setzen
06759  e5db  8d 05 d5          sta mmumcr       MMU 8722 Modus Config. Reg.
06760  e5de
06761  e5de  a9 7f             lda #%01111111   alle IRQ disabled
06762  e5e0  8d 0d dc          sta d1icr        CIA1 Interrupt Control-Register
06763  e5e3
06764  e5e3  ===> CIA SDR OUTPUT, TIMER A BAUD RATE GEN. <===
06765  e5e3
06766  e5e3  a9 00             lda #>4          TIMER A auf 1/4 PHI2 programmieren
06767  e5e5  8d 05 dc          sta d1t1h        CIA1 TIMER A high
```

```
06768  e5e8
06769  e5e8  a9 04           lda #<4          SDR Shift mit 1/2 TIMER A Underflow
                                              Rate
06770  e5ea  8d 04 dc        sta d1t1l        CIA1 TIMER A low
06771  e5ed
06772  e5ed  ad 0e dc        lda d1cra        CIA1 Control-Register a
06773  e5f0  29 80           and #%10000000   TODIN erhalten
06774  e5f2  09 55           ora #%01010101
06775  e5f4
06776  e5f4   x1xxxxxx  Serial Port Output Mode
06777  e5f4   xx0xxxxx  TIMER A zählt PHI2
06778  e5f4   xxx1xxxx  FORCE LOAD
06779  e5f4   xxxx0xxx  continous
06780  e5f4   xxxxx1xx  Toggle
06781  e5f4   xxxxxx0x  PB6 normal
06782  e5f4   xxxxxxx1  Start TIMER A
06783  e5f4
06784  e5f4  8d 0e dc        sta d1cra        CIA1 Control-Register a
06785  e5f7
06786  e5f7  2c 0d dc        bit d1icr        IRQ-Flags rücksetzen
06787  e5fa  60              rts
06788  e5fb
06789  e5fb
06790  e5fb
06791  e5fb  ===> Lesen/Schreiben FSD <===
06792  e5fb
06793  e5fb  90 c6      spinot bcc spinp       Serial Port INPUT
06794  e5fd  b0 d7            bcs spout        Serial Port OUTPUT
06795  e5ff
06796  e5ff
06797  e5ff                   .end
06798  e5ff                   .lib c128.rs232
06799  e5ff
```

4.1.13 RS232-Schnittstelle

```
06801  e5ff
06802  e5ff
06803  e5ff  *****  Die RS232-Schnittstelle ist identisch mit      *****
06804  e5ff  *****  der des C 64. Die Leitungen haben TTL-Pegel    *****
06805  e5ff  *****  und sind auf USER-PORT (CIA 2) zugänglich:     *****
06806  e5ff
06807  e5ff   PIN   6526                          EIA     ABV
06808  e5ff
06809  e5ff    C    PB0   RECEIVED DATA          (BB)    Sin   IN
06810  e5ff    D    PB1   REQUEST TO SEND        (CA)    RTS   OUT
06811  e5ff    E    PB2   DATA TERMINAL READY    (CD)    DTR   OUT
06812  e5ff    F    PB3   RING INDICATOR  *      (CE)    RI    IN
06813  e5ff    H    PB4   RECEIVED LINE SIGNAL   (CF)    DCD   IN
06814  e5ff    J    PB5   unbenutzt
06815  e5ff    K    PB6   CLEAR TO SEND          (CB)    CTS   IN
06816  e5ff    L    PB7   DATA SET READY         (CC)    DSR   IN
06817  e5ff
06818  e5ff    B   FLAG2  RECEIVED DATA          (BB)    Sin   IN
06819  e5ff    M    PA2   TRANSMITTED DATA       (BA)    Sout  OUT
06820  e5ff
```

```
06821  e5ff    A    GND   PROTECTIVE GROUND       (AA)    GND
06822  e5ff    N    GND   SIGNAL GROUND           (AB)    GND
06823  e5ff
06824  e5ff    * RING INDICATOR wird vom System nicht ausgewertet
06825  e5ff
06826  e5ff
06827  e5ff
06828  e5ff  ===> RS 232 Ausgabe (NMI) <===
06829  e5ff
06830  e5ff  a5 b4       rstrab lda snsw1        RS 232 Ausgabe Bitzähler
06831  e601  f0 47              beq rstbgn       nächstes Byte auf RS 232 senden
06832  e603  30 3f              bmi rst050       STOPP-BIT
06833  e605
06834  e605  ===> Datenbyte rotieren <===
06835  e605
06836  e605  46 b6              lsr rodata       RS 232 Ausgabe Byte-Puffer
06837  e607  a2 00              ldx #%00000000   SPACE
06838  e609  90 01              bcc rst005       PARITY berechnen
06839  e60b
06840  e60b  ca                 dex              %11111111 ;MARK
06841  e60c
06842  e60c
06843  e60c  ===> PARITY berechnen <===
06844  e60c
06845  e60c  8a          rst005 txa
06846  e60d  45 bd              eor ochar        RS 232 Ausgabe-Parität
06847  e60f  85 bd              sta ochar        RS 232 Ausgabe-Parität
06848  e611
06849  e611  c6 b4              dec snsw1        RS 232 Ausgabe Bitzähler
06850  e613  f0 06              beq rst010       Test auf PARITY
06851  e615
06852  e615
06853  e615  ===> Ausgabebit aus xr maskieren <===
06854  e615
06855  e615  8a          rstext txa
06856  e616  29 04              and #%00000100   Ausgabebit maskieren
06857  e618
06858  e618   .....2..  nächstes Ausgabebit RS 232 Senden
06859  e618
06860  e618  85 b5              sta nxtbit       RS 232 nächstes Ausgabe-Bit
06861  e61a
06862  e61a  60                 rts
06863  e61b
06864  e61b
06865  e61b  ===> Test auf PARITY <===
06866  e61b
06867  e61b  a9 20       rst010 lda #%00100000
06868  e61d
06869  e61d   765.....  PARITY
06870  e61d   xx0xxxxx  NO    PARITY
06871  e61d   001xxxxx  ODD   PARITY
06872  e61d   011xxxxx  EVEN  PARITY
06873  e61d   101xxxxx  MARK  PARITY
06874  e61d   111xxxxx  SPACE PARITY
06875  e61d
06876  e61d  2c 11 0a           bit m51cdr       RS 232 Command-Register
06877  e620  f0 14              beq rspno        NO PARITY
06878  e622  30 1c              bmi rst040       Feste PARITY
```

```
06879  e624  70 14            bvs rst030       EVEN PARITY
06880  e626
06881  e626  ===> ODD PARITY <===
06882  e626
06883  e626  a5 bd            lda ochar        RS 232 Ausgabe-Parität
06884  e628  d0 01            bne rspext       Ausgabe SPACE
06885  e62a
06886  e62a
06887  e62a  ===> Ausgabe MARK <===
06888  e62a
06889  e62a  ca        rswext dex              %11111111 MARK
06890  e62b
06891  e62b
06892  e62b  ===> Ausgabe SPACE <===
06893  e62b
06894  e62b  c6 b4     rspext dec snsw1        RS 232 Ausgabe Bitzähler
06895  e62d
06896  e62d  ad 10 0a          lda m51ctr       RS 232 CTRL-Register
06897  e630
06898  e630   0xxxxxxx  1 Stopp-Bit
06899  e630   1xxxxxxx  2 Stopp-Bits
06900  e630
06901  e630  10 e3            bpl rstext       Ausgabebit aus xr maskieren
06902  e632
06903  e632  c6 b4            dec snsw1        RS 232 Ausgabe Bitzähler
06904  e634  d0 df            bne rstext       2 Stopp-Bits
06905  e636
06906  e636
06907  e636  ===> NO PARITY <===
06908  e636
06909  e636  e6 b4     rspno  inc snsw1        RS 232 Ausgabe Bitzähler
06910  e638  d0 f0            bne rswext       Ausgabe MARK
06911  e63a
06912  e63a
06913  e63a  ===> EVEN PARITY <===
06914  e63a
06915  e63a  a5 bd     rst030 lda ochar        RS 232 Ausgabe-Parität
06916  e63c  f0 ed            beq rspext       Ausgabe SPACE
06917  e63e  d0 ea            bne rswext       Ausgabe MARK
06918  e640
06919  e640
06920  e640  ===> Feste PARITY <===
06921  e640
06922  e640  70 e9     rst040 bvs rspext       Ausgabe SPACE
06923  e642  50 e6            bvc rswext       Ausgabe MARK
06924  e644
06925  e644
06926  e644  ===> STOPP-BIT <===
06927  e644
06928  e644  e6 b4     rst050 inc snsw1        RS 232 Ausgabe Bitzähler
06929  e646  a2 ff            ldx #%11111111
06930  e648  d0 cb            bne rstext       Ausgabebit aus xr maskieren
06931  e64a
06932  e64a
06933  e64a  ===> nächstes Byte auf RS 232 senden <===
06934  e64a
06935  e64a  ad 11 0a  rstbgn lda m51cdr       RS 232 Command-Register
06936  e64d  4a                lsr a           Bit0 ins Carry
```

```
06937  e64e
06938  e64e   xxxxxxx0  3-Draht
06939  e64e   xxxxxxx1  x-Draht
06940  e64e
06941  e64e  90 07             bcc rst060       nicht gesetzt, 3-Draht
06942  e650
06943  e650  ===> X-Draht Handshake <===
06944  e650
06945  e650  2c 01 dd          bit d2prb        CIA2 Datenport B
06946  e653
06947  e653   7.......  DSR  DATA SET READY
06948  e653   .6......  CTS  CLEAR TO SEND
06949  e653
06950  e653  10 1d             bpl dsrerr       Fehler DSR fehlt
06951  e655  50 1e             bvc ctserr       Fehler CTS fehlt
06952  e657
06953  e657
06954  e657  ===> Ausgabe PARITY initialisieren <===
06955  e657
06956  e657  a9 00       rst060 lda #%00000000
06957  e659  85 bd              sta ochar        RS 232 Ausgabe-Parität
06958  e65b
06959  e65b  ===> Startbit SPACE <===
06960  e65b
06961  e65b  85 b5              sta nxtbit       RS 232 nächstes Ausgabe-Bit
06962  e65d
06963  e65d  ae 15 0a           ldx bitnum       RS 232 Anzahl Datenbits
06964  e660  86 b4              stx snsw1        RS 232 Ausgabe Bitzähler
06965  e662
06966  e662  ac 1a 0a           ldy rodbs        RS 232 Zeiger auf einzules. Byte
06967  e665  cc 1b 0a           cpy rodbe        RS 232 Zeiger auf auszugeb. Byte
06968  e668  f0 13              beq rsodne       Ausgabepuffer leer, Senden aus
06969  e66a
06970  e66a  ===> nächstes Byte aus Ausgabepuffer <===
06971  e66a
06972  e66a  b1 ca              lda (robuf),y    RS 232 addrl Ausgabepuffer
06973  e66c  85 b6              sta rodata       RS 232 Ausgabe Byte-Puffer
06974  e66e
06975  e66e  ee 1a 0a           inc rodbs        RS 232 Zeiger auf einzules. Byte
06976  e671  60                 rts
06977  e672
06978  e672
06979  e672  ===> DSR fehlt, Status setzen <===
06980  e672
06981  e672  a9 40       dsrerr lda #%01000000
06982  e674  2c                 .byte skip2
06983  e675
06984  e675
06985  e675  ===> CLEAR TO SEND (CTS) fehlt, Status setzen <===
06986  e675
06987  e675  a9 10       ctserr lda #%00010000
06988  e677  0d 14 0a           ora rsstat       RS 232 Status-Byte
06989  e67a  8d 14 0a           sta rsstat       RS 232 Status-Byte
06990  e67d
06991  e67d
06992  e67d  ===> Ausgabepuffer leer, Senden aus <===
06993  e67d
06994  e67d  a9 01       rsodne lda #%00000001   CLEAR IRQ TIMER A UNDERFLOW (Senden)
```

```
06995  e67f
06996  e67f
06997  e67f  ===> enable akt. NMI <===
06998  e67f
06999  e67f            oenabl
07000  e67f
07001  e67f   7.......  1=SET, 0=CLEAR IRQ ENABLE BIT
07002  e67f   ...4....  FLAG-IRQ (RS INPUT)
07003  e67f   ......1.  TIMER B UNDERFLOW  (Empfangen)
07004  e67f   .......0  TIMER A UNDERFFLOW (Senden)
07005  e67f
07006  e67f  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07007  e682
07008  e682  4d 0f 0a          eor enabl        CIA2 NMI-Flag
07009  e685  09 80             ora #%10000000   Bit7: 1=SET IRQ-FLAG
07010  e687  8d 0f 0a          sta enabl        CIA2 NMI-Flag
07011  e68a
07012  e68a   7.......  SET IRQ ENABLE BIT
07013  e68a   ...4....  FLAG-IRQ (RS INPUT)
07014  e68a   ......1.  TIMER B UNDERFLOW  (Empfangen)
07015  e68a   .......0  TIMER A UNDERFFLOW (Senden)
07016  e68a
07017  e68a  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07018  e68d
07019  e68d  60                rts
07020  e68e
07021  e68e
07022  e68e  ===> Anzahl Datenbits festlegen <===
07023  e68e
07024  e68e  a2 09       bitcnt ldx #9           Anzahl Datenbits+1
07025  e690  a9 20              lda #%00100000
07026  e692  2c 10 0a           bit m51ctr       RS 232 CTRL-Register
07027  e695
07028  e695   x00xxxxx  8 Datenbits
07029  e695   x01xxxxx  7 Datenbits
07030  e695   x10xxxxx  6 Datenbits
07031  e695   x11xxxxx  5 Datenbits
07032  e695
07033  e695  f0 01             beq bit010
07034  e697
07035  e697  ca                dex
07036  e698  50 02       bit010 bvc bit020
07037  e69a
07038  e69a  ca                dex
07039  e69b  ca                dex
07040  e69c  60          bit020 rts
07041  e69d
07042  e69d
07043  e69d  ===> RS 232 Empfang (NMI) <===
07044  e69d
07045  e69d  a6 a9       rsrcvr ldx rinone       RS 232 Flag für Startbit
07046  e69f  d0 33              bne rsrtrt       Startbit
07047  e6a1
07048  e6a1  c6 a8              dec bitci        Bitzähler erniedrigen
07049  e6a3  f0 3a              beq rsr030       alle Bits empfangen, Byte in Puffer
07050  e6a5  30 0d              bmi rsr020       Stopp-Bit
07051  e6a7
```

```
07052  e6a7  ===> PARITY berechnen <===
07053  e6a7
07054  e6a7  a5 a7             lda inbit
07055  e6a9  45 ab             eor riprty       RS 232 Eingabe-Parität
07056  e6ab  85 ab             sta riprty       RS 232 Eingabe-Parität
07057  e6ad
07058  e6ad  ===> akt. Datenbit in Byte rotieren <===
07059  e6ad
07060  e6ad  46 a7             lsr inbit
07061  e6af  66 aa             ror ridata       RS 232 Eingabe Byte
07062  e6b1  60         rsrext rts
07063  e6b2
07064  e6b2
07065  e6b2  ===> NO PARITY <===
07066  e6b2
07067  e6b2  c6 a8      rsr018 dec bitci        RS 232 Bitzähler
07068  e6b4
07069  e6b4
07070  e6b4  ===> Stopp-Bit <===
07071  e6b4
07072  e6b4  a5 a7      rsr020 lda inbit        Stoppbit-Zähler
07073  e6b6  f0 6b             beq rsr060       FRAME-Fehler oder BREAK
07074  e6b8
07075  e6b8  ad 10 0a          lda m51ctr       RS 232 CTRL-Register
07076  e6bb
07077  e6bb   0xxxxxxx  1 Stopp-Bit
07078  e6bb   1xxxxxxx  2 Stopp-Bits
07079  e6bb
07080  e6bb  0a                asl a            clc = 1, sec = 2 Stopp-Bits
07081  e6bc  a9 01             lda #1
07082  e6be  65 a8             adc bitci        RS 232 Bitzähler
07083  e6c0  d0 ef             bne rsrext
07084  e6c2
07085  e6c2
07086  e6c2  ===> RS 232 Empfang initialisieren <===
07087  e6c2
07088  e6c2  a9 90      rsrabl lda #%10010000
07089  e6c4
07090  e6c4   1xxxxxxx  SET IRQ ENABLE BIT
07091  e6c4   xxx1xxxx  FLAG IRQ (RS 232 RECEIVED DATA)
07092  e6c4
07093  e6c4  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07094  e6c7
07095  e6c7  0d 0f 0a          ora enabl        CIA2 NMI-Flag
07096  e6ca  8d 0f 0a          sta enabl        CIA2 NMI-Flag
07097  e6cd  85 a9             sta rinone       RS 232 Flag für Startbit
07098  e6cf
07099  e6cf  a9 02             lda #%00000010
07100  e6d1
07101  e6d1   xxxxxx1x  TIMER B UNDERFLOW IRQ (RS in) enable
07102  e6d1
07103  e6d1  4c 7f e6          jmp oenabl       enable akt. NMI
07104  e6d4
07105  e6d4
07106  e6d4  ===> Warten auf Startbit <===
07107  e6d4
07108  e6d4  a5 a7      rsrtrt lda inbit
07109  e6d6  d0 ea             bne rsrabl       kein Startbit, erneut initialisieren
```

```
07110  e6d8
07111  e6d8  ===> Startbit empfangen <===
07112  e6d8
07113  e6d8  85 a9            sta rinone       Flag für Startbit empfangen
07114  e6da
07115  e6da  a9 01            lda #%00000001   Vorbesetzen der
07116  e6dc  85 ab            sta riprty       RS 232 Eingabe-Parität
07117  e6de
07118  e6de  60               rts
07119  e6df
07120  e6df
07121  e6df  ===> alle Bits empfangen, Byte in Puffer <===
07122  e6df
07123  e6df  ac 18 0a   rsr030 ldy ridbe        RS 232 Zeiger auf eingeles. Byte
07124  e6e2
07125  e6e2  ===> Test auf Empfangspuffer voll <===
07126  e6e2
07127  e6e2  c8               iny
07128  e6e3  cc 19 0a         cpy ridbs        RS 232 Zeiger auf auszules. Byte
07129  e6e6  f0 2a            beq recerr       Empfänger-Puffer voll, Status setzen
07130  e6e8
07131  e6e8  8c 18 0a         sty ridbe        RS 232 Zeiger auf eingeles. Byte
07132  e6eb  88               dey
07133  e6ec
07134  e6ec  a5 aa            lda ridata       RS 232 Eingabe Byte
07135  e6ee
07136  e6ee  ===> Datenbyte < 8-Bit nach rechts schieben <===
07137  e6ee
07138  e6ee  ae 15 0a         ldx bitnum       RS 232 Anzahl Datenbits
07139  e6f1  e0 09      rsr031 cpx #9
07140  e6f3  f0 04            beq rsr032       Byte in RS 232 Empfangspuffer
07141  e6f5
07142  e6f5  4a               lsr a
07143  e6f6  e8               inx
07144  e6f7  d0 f8            bne rsr031       immer
07145  e6f9
07146  e6f9
07147  e6f9  ===> Byte in RS 232 Empfangspuffer <===
07148  e6f9
07149  e6f9  91 c8      rsr032 sta (ribuf),y    RS 232 addrl Eingabe-Puffer
07150  e6fb
07151  e6fb  a9 20            lda #%00100000
07152  e6fd
07153  e6fd   765.....  PARITY
07154  e6fd   xx0xxxxx  NO    PARITY
07155  e6fd   001xxxxx  ODD   PARITY
07156  e6fd   011xxxxx  EVEN  PARITY
07157  e6fd   101xxxxx  MARK  PARITY
07158  e6fd   111xxxxx  SPACE PARITY
07159  e6fd
07160  e6fd  2c 11 0a         bit m51cdr       RS 232 Command-Register
07161  e700  f0 b0            beq rsr018       NO PARITY
07162  e702  30 ad            bmi rsrext       feste PARITY, Bit verwerfen
07163  e704
07164  e704  ===> PARITY Prüfung <===
07165  e704
07166  e704  a5 a7            lda inbit        empfangenes PARITY-BIT
07167  e706  45 ab            eor riprty       errechnete PARITY
```

```
07168  e708  f0 03             beq rsr050       Prüfung ODD PARITY
07169  e70a
07170  e70a  ===> Prüfung EVEN PARITY <===
07171  e70a
07172  e70a  70 a5             bvs rsrext       EVEN PARITY: ok
07173  e70c
07174  e70c  2c                .byte skip2
07175  e70d
07176  e70d
07177  e70d  ===> Prüfung ODD PARITY <===
07178  e70d
07179  e70d  50 a2      rsr050 bvc rsrext       ODD PARITY: ok
07180  e70f
07181  e70f  ===> PARITY-Fehler, Status setzen <===
07182  e70f
07183  e70f  a9 01             lda #%00000001
07184  e711  2c                .byte skip2
07185  e712
07186  e712
07187  e712  ===> Empfänger-Puffer voll, Status setzen <===
07188  e712
07189  e712  a9 04      recerr lda #%00000100
07190  e714  2c                .byte skip2
07191  e715
07192  e715
07193  e715  ===> BREAK bei Empfang, Status setzen <===
07194  e715
07195  e715  a9 80      breake lda #%10000000
07196  e717  2c                .byte skip2
07197  e718
07198  e718
07199  e718  ===> FRAME-Fehler, Status setzen <===
07200  e718
07201  e718  a9 02      framee lda #%00000010
07202  e71a  0d 14 0a          ora rsstat       RS 232 Status-Byte
07203  e71d  8d 14 0a          sta rsstat       RS 232 Status-Byte
07204  e720
07205  e720  4c c2 e6          jmp rsrabl       RS 232 Empfang initialisieren
07206  e723
07207  e723
07208  e723  ===> FRAME-Fehler oder BREAK <===
07209  e723
07210  e723  a5 aa      rsr060 lda ridata       RS 232 Eingabe Byte
07211  e725  d0 f1             bne framee       FRAME-Fehler, Status setzen
07212  e727  f0 ec             beq breake       BREAK bei Empfang, Status setzen
07213  e729
07214  e729
07215  e729  ===> Open RS 232 Ausgabe <===
07216  e729
07217  e729  85 9a      cko232 sta dflto        Vorgabe Ausgabe-Gerätenummer (3)
07218  e72b
07219  e72b  ad 11 0a          lda m51cdr       RS 232 Command-Register
07220  e72e  4a                lsr a
07221  e72f
07222  e72f   xxxxxxx0  3-Draht
07223  e72f   xxxxxxx1  x-Draht
07224  e72f
07225  e72f  90 29             bcc cko100       3-Draht Handshake
```

```
07226  e731
07227  e731  ===> X-Draht Handshake <===
07228  e731
07229  e731  a9 02           lda #%00000010
07230  e733  2c 01 dd        bit d2prb        CIA2 Datenport B
07231  e736
07232  e736   7.......  DSR  DATA SET READY
07233  e736   ......1.  RTS  REQUEST TO SEND
07234  e736
07235  e736  10 1d           bpl ckdsrx       DSR fehlt
07236  e738  d0 20           bne cko100       RTS fehlt
07237  e73a
07238  e73a
07239  e73a  ===> Warten auf Datenempfang beendet <===
07240  e73a
07241  e73a  ad 0f 0a   cko020 lda enabl        CIA2 NMI-Flag
07242  e73d
07243  e73d   xxxxxx1x  TIMER B UNDERFLOW IRQ (RS in) enable
07244  e73d
07245  e73d  29 02           and #%00000010
07246  e73f  d0 f9           bne cko020       Warten auf Datenempfang beendet
07247  e741
07248  e741
07249  e741  ===> Warten auf CLEAR TO SEND <===
07250  e741
07251  e741  2c 01 dd   cko030 bit d2prb        CIA2 Datenport B
07252  e744
07253  e744   .6......  CTS  CLEAR TO SEND
07254  e744
07255  e744  70 fb           bvs cko030       Warten auf CLEAR TO SEND
07256  e746
07257  e746  ad 01 dd        lda d2prb        CIA2 Datenport B
07258  e749  09 02           ora #%00000010
07259  e74b
07260  e74b   ......1.  RTS  REQUEST TO SEND
07261  e74b
07262  e74b  8d 01 dd        sta d2prb        CIA2 Datenport B
07263  e74e
07264  e74e  2c 01 dd   cko040 bit d2prb        CIA2 Datenport B
07265  e751
07266  e751   7.......  DSR  DATA SET READY
07267  e751   .6......  CTS  CLEAR TO SEND
07268  e751
07269  e751  70 07           bvs cko100
07270  e753  30 f9           bmi cko040
07271  e755
07272  e755
07273  e755  ===> DSR fehlt, Status setzen <===
07274  e755
07275  e755  a9 40      ckdsrx lda #%01000000
07276  e757  8d 14 0a         sta rsstat       RS 232 Status-Byte
07277  e75a
07278  e75a  18         cko100 clc
07279  e75b  60               rts
07280  e75c
07281  e75c
```

```
07282  e75c  ===> Puffer voll, warten ... <===
07283  e75c
07284  e75c  20 70 e7    bsobad jsr bso100
07285  e75f
07286  e75f
07287  e75f  ===> Ausgabe in RS 232 Puffer <===
07288  e75f
07289  e75f  ac 1b 0a    bso232 ldy rodbe        RS 232 Zeiger auf auszugeb. Byte
07290  e762  c8                 iny
07291  e763  cc 1a 0a           cpy rodbs        RS 232 Zeiger auf einzules. Byte
07292  e766  f0 f4              beq bsobad       Puffer voll, warten ...
07293  e768
07294  e768  8c 1b 0a           sty rodbe        RS 232 Zeiger auf auszugeb. Byte
07295  e76b  88                 dey
07296  e76c  a5 9e              lda ptr1
07297  e76e  91 ca              sta (robuf),y    RS 232 addrl Ausgabepuffer
07298  e770
07299  e770  ad 0f 0a    bso100 lda enabl        CIA2 NMI-Flag
07300  e773
07301  e773   xxxxxxx1  TIMER A UNDERFLOW IRQ (RS out) enabled
07302  e773
07303  e773  4a                 lsr a
07304  e774  b0 1e              bcs bso120       RS 232 Ausgabe ein
07305  e776
07306  e776  ===> RS 232 Ausgabe einschalten <===
07307  e776
07308  e776  a9 10              lda #%00010000
07309  e778
07310  e778   0xxxxxxx  CLEAR IRQ FLAG
07311  e778   xxx1xxxx  FLAG IRQ (RS 232 RECEIVED DATA)
07312  e778
07313  e778  8d 0e dd           sta d2cra        CIA2 Control-Register A
07314  e77b
07315  e77b  ad 16 0a           lda baudof       RS 232 Timer-Konst. lo BAUD-Rate
07316  e77e  8d 04 dd           sta d2t1l        CIA2 TIMER A low
07317  e781
07318  e781  ad 17 0a           lda baudof+1     RS 232 Timer-Konst. hi BAUD-Rate
07319  e784  8d 05 dd           sta d2t1h        CIA2 TIMER A high
07320  e787
07321  e787  a9 81              lda #%10000001
07322  e789  20 7f e6           jsr oenabl       enable akt. NMI
07323  e78c  20 4a e6           jsr rstbgn       nächstes Byte auf RS 232 senden
07324  e78f
07325  e78f  a9 11              lda #%00010001
07326  e791
07327  e791   0xxxxxxx  CLEAR IRQ FLAG
07328  e791   xxx1xxxx  FLAG IRQ (RS 232 RECEIVED DATA)
07329  e791   xxxxxxx1  TIMER A UNDERFLOW IRQ (RS out)
07330  e791
07331  e791  8d 0e dd           sta d2cra        CIA2 Control-Register A
07332  e794
07333  e794  60          bso120 rts
07334  e795
07335  e795
07336  e795  ===> Open RS 232 Eingabe <===
07337  e795
07338  e795  85 99       cki232 sta dfltn        Vorgabe Eingabe-Gerätenummer (0)
07339  e797
```

```
07340  e797  ad 11 0a          lda m51cdr       RS 232 Command-Register
07341  e79a
07342  e79a   xxxxxxx0  3-Draht
07343  e79a   xxxxxxx1  x-Draht
07344  e79a
07345  e79a  4a               lsr a
07346  e79b  90 28            bcc cki100       3-Draht Handshake
07347  e79d
07348  e79d  ===> X-Draht Handshake <===
07349  e79d
07350  e79d  29 08            and #%00001000
07351  e79f
07352  e79f   xxx0xxxx  FULL DUPLEX
07353  e79f
07354  e79f  f0 24            beq cki100       FULL DUPLEX
07355  e7a1
07356  e7a1  ===> HALF DUPLEX <===
07357  e7a1
07358  e7a1  a9 02            lda #%00000010
07359  e7a3  2c 01 dd         bit d2prb        CIA2 Datenport B
07360  e7a6
07361  e7a6   7.......  DSR  DATA SET READY
07362  e7a6   ......1.  RTS  REQUEST TO SEND
07363  e7a6
07364  e7a6  10 ad            bpl ckdsrx       Fehler DSR fehlt
07365  e7a8  f0 22            beq cki110       RTS nicht gesetzt (Senden aus)
07366  e7aa
07367  e7aa
07368  e7aa  ===> Warten auf Ende Senden <===
07369  e7aa
07370  e7aa  ad 0f 0a   cki010 lda enabl        CIA2 NMI-Flag
07371  e7ad
07372  e7ad   xxxxxxx1  TIMER A UNDERFLOW IRQ (RS out) enable
07373  e7ad
07374  e7ad  4a               lsr a
07375  e7ae  b0 fa            bcs cki010       Warten auf Ende Senden
07376  e7b0
07377  e7b0  ===> RTS  REQUEST TO SEND aus <===
07378  e7b0
07379  e7b0  ad 01 dd         lda d2prb        CIA2 Datenport B
07380  e7b3  29 fd            and #%11111101
07381  e7b5
07382  e7b5   ......0.  RTS  REQUEST TO SEND
07383  e7b5
07384  e7b5  8d 01 dd         sta d2prb        CIA2 Datenport B
07385  e7b8
07386  e7b8
07387  e7b8
07388  e7b8  ===> Warten auf DATA TERMINAL READY <===
07389  e7b8
07390  e7b8  ad 01 dd   cki020 lda d2prb        CIA2 Datenport B
07391  e7bb  29 04            and #%00000100
07392  e7bd
07393  e7bd   xxxxx1xx  DTR  DATA TERMINAL READY
07394  e7bd
07395  e7bd  f0 f9            beq cki020       Warten auf DATA TERMINAL READY
07396  e7bf
07397  e7bf  a9 90      cki080 lda #%10010000
```

```
07398  e7c1
07399  e7c1   1xxxxxxx  SET IRQ ENABLE BIT
07400  e7c1   xxx1xxxx  FLAG IRQ (RS 232 RECEIVED DATA)
07401  e7c1
07402  e7c1  18                clc
07403  e7c2  4c 7f e6          jmp oenabl        enable akt. NMI
07404  e7c5
07405  e7c5
07406  e7c5  ===> Test Empfang / Warten auf Empfang <===
07407  e7c5
07408  e7c5  ad 0f 0a    cki100 lda enabl         CIA2 NMI-Flag
07409  e7c8  29 12              and #%00010010
07410  e7ca
07411  e7ca   ...4....  FLAG-IRQ (RS INPUT)
07412  e7ca   ......1.  TIMER B UNDERFLOW  (Empfangen)
07413  e7ca
07414  e7ca  f0 f3              beq cki080
07415  e7cc
07416  e7cc  18          cki110 clc
07417  e7cd  60                 rts
07418  e7ce
07419  e7ce
07420  e7ce  ===> get von RS 232 <===
07421  e7ce
07422  e7ce  ad 14 0a    bsi232 lda rsstat        RS 232 Status-Byte
07423  e7d1
07424  e7d1  ac 19 0a           ldy ridbs         RS 232 Zeiger auf auszules. Byte
07425  e7d4  cc 18 0a           cpy ridbe         RS 232 Zeiger auf eingeles. Byte
07426  e7d7  f0 0b              beq bsi010        Status 'Empfängerpuffer leer'
07427  e7d9
07428  e7d9  ===> Status 'Empfängerpuffer leer' rücksetzen <===
07429  e7d9
07430  e7d9  29 f7              and #%11110111
07431  e7db  8d 14 0a           sta rsstat        RS 232 Status-Byte
07432  e7de
07433  e7de  b1 c8              lda (ribuf),y     RS 232 addrl Eingabe-Puffer
07434  e7e0  ee 19 0a           inc ridbs         RS 232 Zeiger auf auszules. Byte
07435  e7e3  60                 rts
07436  e7e4
07437  e7e4
07438  e7e4  ===> Status 'Empfängerpuffer leer' <===
07439  e7e4
07440  e7e4  09 08       bsi010 ora #%00001000
07441  e7e6  8d 14 0a           sta rsstat        RS 232 Status-Byte
07442  e7e9
07443  e7e9  a9 00              lda #0            Ersatzwert leerer Empfangspuffer
07444  e7eb  60                 rts
07445  e7ec
07446  e7ec
07447  e7ec  ===> Test / Warten auf Ende RS 232 I/O <===
07448  e7ec
07449  e7ec  48          rsp232 pha
07450  e7ed
07451  e7ed  ad 0f 0a           lda enabl         CIA2 NMI-Flag
07452  e7f0  f0 11              beq rspok
07453  e7f2
07454  e7f2
```

```
07455  e7f2  ===> Warten auf Ende RS 232 I/O <===
07456  e7f2
07457  e7f2  ad 0f 0a   rspoff lda enabl        CIA2 NMI-Flag
07458  e7f5  29 03             and #%00000011
07459  e7f7
07460  e7f7   xxxxxx1x  TIMER B UNDERFLOW IRQ (RS in)
07461  e7f7   xxxxxxx1  TIMER A UNDERFLOW IRQ (RS out)
07462  e7f7
07463  e7f7  d0 f9             bne rspoff       Warten auf Ende RS 232 I/O
07464  e7f9
07465  e7f9  ===> IRQ DISABLE RS 232 Empfang <===
07466  e7f9
07467  e7f9  a9 10             lda #%00010000
07468  e7fb
07469  e7fb   0xxxxxxx  CLEAR IRQ ENABLE BIT
07470  e7fb   xxx1xxxx  FLAG IRQ (RS 232 RECEIVED DATA)
07471  e7fb
07472  e7fb  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07473  e7fe
07474  e7fe  a9 00             lda #%00000000   RS 232 disabled
07475  e800  8d 0f 0a          sta enabl        CIA2 NMI-Flag
07476  e803
07477  e803  68         rspok  pla
07478  e804  60                rts
07479  e805
07480  e805
07481  e805  ===> RS 232-NMI bearbeiten <===
07482  e805
07483  e805  98         nmi232 tya              CIA2 d2icr (IRQ Flags)
07484  e806  2d 0f 0a          and enabl        CIA2 NMI-Flag
07485  e809
07486  e809   ...4....  FLAG-IRQ (RS INPUT)
07487  e809   ......1.  TIMER B UNDERFLOW  (Empfangen)
07488  e809   .......0  TIMER A UNDERFFLOW (Senden)
07489  e809
07490  e809  aa                tax
07491  e80a  29 01             and #%00000001
07492  e80c
07493  e80c   .......0  TIMER A UNDERFFLOW (Senden)
07494  e80c
07495  e80c  f0 28             beq nnmi30       Test IRQ TIMER B UNDERFLOW (RS in)
07496  e80e
07497  e80e  ===> IRQ RS 232 Senden <===
07498  e80e
07499  e80e  ad 00 dd          lda d2pra        CIA2 Datenport A
07500  e811  29 fb             and #%11111011
07501  e813
07502  e813   .....2..  TXD  TRANSMIT DATA
07503  e813
07504  e813  05 b5             ora nxtbit       RS 232 nächstes Ausgabe-Bit
07505  e815
07506  e815   .....2..  nächstes Ausgabebit RS 232 Senden
07507  e815
07508  e815  8d 00 dd          sta d2pra        CIA2 Datenport A
07509  e818
07510  e818  ad 0f 0a          lda enabl        CIA2 NMI-Flag
07511  e81b  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07512  e81e
```

```
07513  e81e  ===> Test auf IRQ RS 232 Empfang <===
07514  e81e
07515  e81e  8a               txa             IRQ Flags für RS 232
07516  e81f  29 12            and #%00010010
07517  e821
07518  e821   ...4....  FLAG-IRQ (RS INPUT)
07519  e821   ......1.  TIMER B UNDERFLOW  (Empfangen)
07520  e821
07521  e821  f0 0d            beq nnmi25
07522  e823
07523  e823  ===> Test auf IRQ TIMER B UNDERFLOW <===
07524  e823
07525  e823  29 02            and #%00000010
07526  e825
07527  e825   ......1.  TIMER B UNDERFLOW  (Empfangen)
07528  e825
07529  e825  f0 06            beq nnmi22      kein UNDERFLOW
07530  e827
07531  e827  ===> IRQ TIMER B UNDERFLOW <===
07532  e827
07533  e827  20 78 e8         jsr t2nmi       NMI-Routine IRQ FLAG (RS in)
07534  e82a  4c 30 e8         jmp nnmi25
07535  e82d
07536  e82d
07537  e82d  20 a9 e8  nnmi22 jsr flnmi       NMI-Routine für RS 232 Ausgabe
07538  e830  20 ff e5  nnmi25 jsr rstrab      RS 232 Ausgabe (NMI)
07539  e833  4c 49 e8         jmp nmirti      RS IRQ ENABLE
07540  e836
07541  e836
07542  e836  ===> Test IRQ TIMER B UNDERFLOW (RS in) <===
07543  e836
07544  e836  8a         nnmi30 txa
07545  e837  29 02            and #%00000010
07546  e839
07547  e839   ......1.  TIMER B UNDERFLOW  (Empfangen)
07548  e839
07549  e839  f0 06            beq nnmi40      kein UNDERFLOW
07550  e83b
07551  e83b  20 78 e8         jsr t2nmi       NMI-Routine IRQ FLAG (RS in)
07552  e83e  4c 49 e8         jmp nmirti      RS IRQ ENABLE
07553  e841
07554  e841
07555  e841  ===> Test FLAG IRQ (RS in) <===
07556  e841
07557  e841  8a         nnmi40 txa
07558  e842  29 10            and #%00010000
07559  e844
07560  e844   ...4....  FLAG-IRQ (RS INPUT)
07561  e844
07562  e844  f0 03            beq nmirti      RS IRQ ENABLE
07563  e846
07564  e846
07565  e846  ===> FLAG IRQ (RS in) <===
07566  e846
07567  e846  20 a9 e8         jsr flnmi       NMI-Routine für RS 232 Ausgabe
07568  e849
07569  e849
```

```
07570  e849  ===> RS IRQ ENABLE <===
07571  e849
07572  e849  ad 0f 0a   nmirti lda enabl        CIA2 NMI-Flag
07573  e84c  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07574  e84f
07575  e84f  60                rts
07576  e850
07577  e850
07578  e850   Die Konstanten c für die Baudrate der RS 232 - Schnittstelle
07579  e850   werden berechnet nach c = (Systemtakt/(2*Baudrate))-100,
07580  e850   wobei der Systemtakt NTSC (USA) 1.02273 MHz und der Systemtakt
07581  e850   PAL (Europa)  0.98525 MHz beträgt.
07582  e850
07583  e850
07584  e850  ===> Tab. RS 232 Baud-Rate Konst. NTSC <===
07585  e850
07586  e850  c1 27      baudo  .word 10177          50   Baud
07587  e852  3e 1a             .word 6718           75   Baud
07588  e854  c5 11             .word 4549          110   Baud
07589  e856  74 0e             .word 3700        134.5   Baud
07590  e858  ed 0c             .word 3309          150   Baud
07591  e85a  45 06             .word 1605          300   Baud
07592  e85c  f0 02             .word 752           600   Baud
07593  e85e  46 01             .word 326          1200   Baud
07594  e860  b8 00             .word 184          1800   Baud
07595  e862  71 00             .word 113          1200   Baud
07596  e864
07597  e864
07598  e864  ===> Tab. RS 232 Baud-Rate Konst. PAL <===
07599  e864
07600  e864  19 26      baudop .word 9753           50   Baud
07601  e866  44 19             .word 6468           75   Baud
07602  e868  1a 11             .word 4378          110   Baud
07603  e86a  e8 0d             .word 3560        134.5   Baud
07604  e86c  70 0c             .word 3184          150   Baud
07605  e86e  06 06             .word 1542          300   Baud
07606  e870  d1 02             .word 721           600   Baud
07607  e872  37 01             .word 311          1200   Baud
07608  e874  ae 00             .word 174          1800   Baud
07609  e876  69 00             .word 105          1200   Baud
07610  e878
07611  e878
07612  e878  ===> NMI-Routine IRQ FLAG (RS in) <===
07613  e878
07614  e878  ad 01 dd   t2nmi  lda d2prb        CIA2 Datenport B
07615  e87b  29 01             and #%00000001
07616  e87d
07617  e87d   .......0  RD  RECEIVED DATA
07618  e87d
07619  e87d  85 a7             sta inbit        Bit0 = akt. Empfangs-Bit
07620  e87f
07621  e87f  ad 06 dd          lda d2t2l        CIA2 TIMER B low
07622  e882  e9 28             sbc #40          40 Zyklen Korrektur
07623  e884  6d 16 0a          adc baudof       RS 232 Timer-Konst. lo BAUD-Rate
07624  e887  8d 06 dd          sta d2t2l        CIA2 TIMER B low
07625  e88a
07626  e88a  ad 07 dd          lda d2t2h        CIA2 TIMER B high
07627  e88d  6d 17 0a          adc baudof+1     RS 232 Timer-Konst. hi BAUD-Rate
```

```
07628  e890  8d 07 dd          sta d2t2h        CIA2 TIMER B high
07629  e893
07630  e893  a9 11             lda #%00010001
07631  e895
07632  e895   0xxxxxxx  write TOD-Register setzt Uhr
07633  e895   x00xxxxx  TIMER B zählt PHI2
07634  e895   xxx1xxxx  FORCE LOAD
07635  e895   xxxx0xxx  CONTINOUS
07636  e895   xxxxx0xx  PULSE
07637  e895   xxxxxx0x  PB7 normal Operation
07638  e895   xxxxxxx1  Start TIMER B (RS in)
07639  e895
07640  e895  8d 0f dd          sta d2crb        CIA2 Control-Register B
07641  e898
07642  e898  ad 0f 0a          lda enabl        CIA2 NMI-Flag
07643  e89b  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
07644  e89e
07645  e89e  a9 ff             lda #%11111111
07646  e8a0  8d 06 dd          sta d2t2l        CIA2 TIMER B low
07647  e8a3  8d 07 dd          sta d2t2h        CIA2 TIMER B high
07648  e8a6
07649  e8a6  4c 9d e6          jmp rsrcvr       RS 232 Empfang (NMI)
07650  e8a9
07651  e8a9
07652  e8a9  ===> NMI-Routine für RS 232 Ausgabe <===
07653  e8a9
07654  e8a9  ad 12 0a   flnmi  lda m51ajb       RS 232 BAUD-Rate Konst. lo
07655  e8ac  8d 06 dd          sta d2t2l        CIA2 TIMER B low
07656  e8af
07657  e8af  ad 13 0a          lda m51ajb+1     RS 232 BAUD-Rate Konst. hi
07658  e8b2  8d 07 dd          sta d2t2h        CIA2 TIMER B high
07659  e8b5
07660  e8b5  a9 11             lda #%00010001
07661  e8b7
07662  e8b7   0xxxxxxx  write TOD-Register setzt Uhr
07663  e8b7   x00xxxxx  TIMER B zählt PHI2
07664  e8b7   xxx1xxxx  FORCE LOAD
07665  e8b7   xxxx0xxx  CONTINOUS
07666  e8b7   xxxxx0xx  PULSE
07667  e8b7   xxxxxx0x  PB7 normal Operation
07668  e8b7   xxxxxxx1  Start TIMER B (RS in)
07669  e8b7
07670  e8b7  8d 0f dd          sta d2crb        CIA2 Control-Register B
07671  e8ba
07672  e8ba  a9 12             lda #%00010010
07673  e8bc
07674  e8bc   ...4....  FLAG-IRQ (RS INPUT)
07675  e8bc   ......1.  TIMER B UNDERFLOW  (Empfangen)
07676  e8bc
07677  e8bc  4d 0f 0a          eor enabl        gesetzte Bits löschen
07678  e8bf  8d 0f 0a          sta enabl        CIA2 NMI-Flag
07679  e8c2
07680  e8c2  a9 ff             lda #%11111111
07681  e8c4  8d 06 dd          sta d2t2l        CIA2 TIMER B low
07682  e8c7  8d 07 dd          sta d2t2h        CIA2 TIMER B high
07683  e8ca
07684  e8ca  ae 15 0a          ldx bitnum       RS 232 Anzahl Datenbits
07685  e8cd  86 a8             stx bitci        RS 232 Bitzähler
```

```
07686  e8cf
07687  e8cf  60              rts
07688  e8d0
07689  e8d0
07690  e8d0                  .end
07691  e8d0                  .lib c128.tape1
07692  e8d0
```

4.1.14 Kassettenrekorder

```
07694  e8d0
07695  e8d0
07696  e8d0  *****  Die Kassettenrekorder-Routinen wurden nur     *****
07697  e8d0  *****  teilweise dokumentiert, da sie nur wenig      *****
07698  e8d0  *****  Interesse finden. Sie sind sehr zeitkritisch  *****
07699  e8d0  *****  und daher kaum zu modifizieren.               *****
07700  e8d0
07701  e8d0
07702  e8d0  ===> Header-Block auf Kassette suchen/anzeigen <===
07703  e8d0
07704  e8d0  a5 93       fah    lda verck        Flag für load/verify
07705  e8d2  48                 pha
07706  e8d3
07707  e8d3  20 f2 e9           jsr rblk         Block von Kassette in Puffer
                                                 einlesen
07708  e8d6  68                 pla
07709  e8d7  85 93              sta verck        Flag für load/verify
07710  e8d9  b0 3d              bcs fah40        rts mit gesetztem Equal-Flag
07711  e8db
07712  e8db  a0 00              ldy #0
07713  e8dd  b1 b2              lda (tape1),y    addrl Zeiger Kassettenpuffer
07714  e8df  c9 05              cmp #5           Marke für EOT
07715  e8e1  f0 34              beq fah46
07716  e8e3
07717  e8e3  c9 01              cmp #1           Header-Marke BASIC-Programm
07718  e8e5  f0 08              beq fah50
07719  e8e7
07720  e8e7  c9 03              cmp #3           Header-Marke Maschinen-Programm
07721  e8e9  f0 04              beq fah50
07722  e8eb
07723  e8eb  c9 04              cmp #4           Header-Marke Datenfile
07724  e8ed  d0 e1              bne fah          Header-Block auf Kassette
                                                 suchen/anzeigen
07725  e8ef
07726  e8ef  aa          fah50  tax              Marke
07727  e8f0  24 9d              bit msgflg       Flag disable Ausgabe OPSYS-Meldungen
07728  e8f2  10 22              bpl fah45        rts mit rückgesetztem Equal-Flag
07729  e8f4
07730  e8f4  a0 63              ldy #99          Zeiger auf 'found'
07731  e8f6  20 22 f7           jsr msg          Ausgabe Systemmeldungen, yr=offset
07732  e8f9
07733  e8f9  a0 05              ldy #5           Zeiger auf Anfang Dateinamen
07734  e8fb
07735  e8fb
07736  e8fb  ===> Filenamen aus Kassettenpuffer auf akt. Kanal ausgeben <===
07737  e8fb
```

```
07738  e8fb  b1 b2      fah55  lda (tape1),y    addrl Zeiger Kassettenpuffer
07739  e8fd  20 d2 ff         jsr kbsout       Ausgabe ac üb. Vector auf akt. Kanal
                                                (Sprungv.)
07740  e900
07741  e900  c8               iny
07742  e901  c0 15            cpy #5+16        Ende Filenamen
07743  e903  d0 f6            bne fah55        Filenamen aus Kassettenpuffer auf
                                                akt. Kanal ausgeben
07744  e905
07745  e905  a5 a1            lda time+1       Uhrzeit in 1/60 s (jiffies)
07746  e907  69 02            adc #2
07747  e909
07748  e909
07749  e909  ===> 8.5 sec auf CBM-Taste warten <===
07750  e909
07751  e909  a4 91      fah56  ldy stkey       Abfrage Tast.matrix Reihen Spalte 7
07752  e90b  c8               iny              %11111111 -> %00000000: keine Taste
                                                gedrückt
07753  e90c  d0 04            bne fah57        Taste gedrückt
07754  e90e
07755  e90e  c5 a1            cmp time+1       Uhrzeit in 1/60 s (jiffies)
07756  e910  d0 f7            bne fah56        8.5 sec auf CBM-Taste warten
07757  e912
07758  e912  c0 f0      fah57  cpy #%11101111+%1      Matrix-Wert SPACE-Taste
07759  e914  f0 ba            beq fah          SPACE-Taste gedrückt
07760  e916
07761  e916
07762  e916  ===> rts mit rückgesetztem Equal-Flag <===
07763  e916
07764  e916  18        fah45  clc
07765  e917  88        fah46  dey
07766  e918
07767  e918
07768  e918  ===> rts mit gesetztem Equal-Flag <===
07769  e918
07770  e918  60        fah40  rts
07771  e919
07772  e919
07773  e919  ===> PRG-Header/Ende-Block auf Kassette schreiben <===
07774  e919
07775  e919  85 9e      tapeh  sta ptr1         Block-Marke
07776  e91b
07777  e91b  20 80 e9         jsr tapadr       Zeiger akt. Kass.puffer einrichten
07778  e91e  90 5f            bcc th40
07779  e920
07780  e920  a5 c2            lda stah         addrh Startadresse save
07781  e922  48               pha
07782  e923
07783  e923  a5 c1            lda stal         addrl Startadresse save
07784  e925  48               pha
07785  e926
07786  e926  a5 af            lda eah          addrh Ende load/save
07787  e928  48               pha
07788  e929
07789  e929  a5 ae            lda eal          addrl Ende load/save
07790  e92b  48               pha
07791  e92c
```

```
07792  e92c  ===> Kassettenpuffer mit Spaces füllen <===
07793  e92c
07794  e92c  a0 bf             ldy #191
07795  e92e  a9 20             lda #' '
07796  e930  91 b2      blnk2  sta (tape1),y    addrl Zeiger Kassettenpuffer
07797  e932  88                dey
07798  e933  d0 fb             bne blnk2
07799  e935
07800  e935  ===> Block-Marke in Kassettenpuffer <===
07801  e935
07802  e935  a5 9e             lda ptr1
07803  e937  91 b2             sta (tape1),y    addrl Zeiger Kassettenpuffer
07804  e939
07805  e939
07806  e939  ===> Startadresse in Kassettenpuffer <===
07807  e939
07808  e939  c8                iny
07809  e93a  a5 c1             lda stal         addrl Startadresse save
07810  e93c  91 b2             sta (tape1),y    addrl Zeiger Kassettenpuffer
07811  e93e
07812  e93e  c8                iny
07813  e93f  a5 c2             lda stah         addrh Startadresse save
07814  e941  91 b2             sta (tape1),y    addrl Zeiger Kassettenpuffer
07815  e943
07816  e943  ===> Endadresse in Kassettenpuffer <===
07817  e943
07818  e943  c8                iny
07819  e944  a5 ae             lda eal          addrl Ende load/save
07820  e946  91 b2             sta (tape1),y    addrl Zeiger Kassettenpuffer
07821  e948
07822  e948  c8                iny
07823  e949  a5 af             lda eah          addrh Ende load/save
07824  e94b  91 b2             sta (tape1),y    addrl Zeiger Kassettenpuffer
07825  e94d
07826  e94d  c8                iny              Zeiger in Kassettenpuffer
07827  e94e  84 9f             sty ptr2
07828  e950
07829  e950  ===> Filenamen in Kassettenpuffer <===
07830  e950
07831  e950  a0 00             ldy #0
07832  e952  84 9e             sty ptr1
07833  e954
07834  e954
07835  e954  ===> Filenamen in Puffer <===
07836  e954
07837  e954  a4 9e      th20   ldy ptr1
07838  e956  c4 b7             cpy fnlen        Länge akt. Filename
07839  e958  f0 0d             beq th30
07840  e95a
07841  e95a  20 ae f7          jsr fnadry       LDA (FNADR),y
07842  e95d  a4 9f             ldy ptr2
07843  e95f  91 b2             sta (tape1),y    addrl Zeiger Kassettenpuffer
07844  e961
07845  e961  e6 9e             inc ptr1
07846  e963  e6 9f             inc ptr2
07847  e965  d0 ed             bne th20         immer
07848  e967
07849  e967
```

```
07850  e967  20 87 e9   th30   jsr ldad1        Zeiger auf Anfang & Ende
                                                 Kassettenpuffer einrichten
07851  e96a  a9 69             lda #%01101001
07852  e96c  85 ab             sta riprty       RS 232 Eingabe-Parität
07853  e96e
07854  e96e  20 1c ea           jsr twrt2        Schreiben auf Kassette
07855  e971  a8                 tay
07856  e972  68                 pla
07857  e973  85 ae             sta eal          addrl Ende load/save
07858  e975
07859  e975  68                 pla
07860  e976  85 af             sta eah          addrh Ende load/save
07861  e978
07862  e978  68                 pla
07863  e979  85 c1             sta stal         addrl Startadresse save
07864  e97b
07865  e97b  68                 pla
07866  e97c  85 c2             sta stah         addrh Startadresse save
07867  e97e
07868  e97e  98                 tya
07869  e97f  60          th40   rts
07870  e980
07871  e980
07872  e980  ===> Zeiger akt. Kass.puffer einrichten <===
07873  e980
07874  e980  a6 b2       tapadr ldx tape1        addrl Zeiger Kassettenpuffer
07875  e982  a4 b3              ldy tape1+1      addrh Zeiger Kassettenpuffer
07876  e984  c0 02              cpy #>buf        Kassettenpuffer
07877  e986  60                 rts
07878  e987
07879  e987
07880  e987  ===> Zeiger auf Anfang & Ende Kassettenpuffer einrichten <===
07881  e987
07882  e987  20 80 e9   ldad1  jsr tapadr       Zeiger akt. Kass.puffer einrichten
07883  e98a  8a                 txa
07884  e98b  85 c1             sta stal         addrl Startadresse save
07885  e98d
07886  e98d  18                 clc
07887  e98e  69 c0             adc #192         Länge Kassettenpuffer
07888  e990  85 ae             sta eal          addrl Ende load/save
07889  e992
07890  e992  98                 tya
07891  e993  85 c2             sta stah         addrh Startadresse save
07892  e995
07893  e995  69 00             adc #0
07894  e997  85 af             sta eah          addrh Ende load/save
07895  e999
07896  e999  60                 rts
07897  e99a
07898  e99a
07899  e99a  ===> File auf Kassette suchen/anzeigen <===
07900  e99a
07901  e99a  20 d0 e8   faf    jsr fah          Header-Block auf Kassette
                                                 suchen/anzeigen
07902  e99d  b0 1e             bcs faf40
07903  e99f
07904  e99f  a0 05             ldy #5           Zeiger auf Filenamen
07905  e9a1  84 9f             sty ptr2
```

```
07906  e9a3
07907  e9a3  a0 00           ldy #0          Zähler Länge Filenamen
07908  e9a5  84 9e           sty ptr1
07909  e9a7
07910  e9a7  c4 b7     faf20 cpy fnlen       Länge akt. Filename
07911  e9a9  f0 11           beq faf30       gleich, ok
07912  e9ab
07913  e9ab  20 ae f7        jsr fnadry      LDA (FNADR),y
07914  e9ae  a4 9f           ldy ptr2
07915  e9b0  d1 b2           cmp (tape1),y   addrl Zeiger Kassettenpuffer
07916  e9b2  d0 e6           bne faf         weitersuchen
07917  e9b4
07918  e9b4  e6 9e           inc ptr1
07919  e9b6  e6 9f           inc ptr2
07920  e9b8  a4 9e           ldy ptr1
07921  e9ba  d0 eb           bne faf20       immer
07922  e9bc
07923  e9bc
07924  e9bc  18        faf30 clc
07925  e9bd  60        faf40 rts
07926  e9be
07927  e9be
07928  e9be  ===> Test auf Kassettenpuffer voll <===
07929  e9be
07930  e9be  20 80 e9  jtp20 jsr tapadr      Zeiger akt. Kass.puffer einrichten
07931  e9c1  e6 a6           inc bufpt       Zeiger in Kassettenpuffer
07932  e9c3  a4 a6           ldy bufpt       Zeiger in Kassettenpuffer
07933  e9c5  c0 c0           cpy #192        Länge Kassettenpuffer
07934  e9c7  60              rts
07935  e9c8
07936  e9c8
07937  e9c8  ===> auf Taste warten, Ausgabe 'press play on tape' <===
07938  e9c8
07939  e9c8  20 df e9  cste1 jsr cs10        gedrückte Taste abfragen
07940  e9cb  f0 1a           beq cs25
07941  e9cd
07942  e9cd  a0 1b           ldy #27         Zeiger auf 'press play on tape'
07943  e9cf  20 22 f7  cs30  jsr msg         Ausgabe Systemmeldungen, yr=offset
07944  e9d2
07945  e9d2
07946  e9d2  ===> Warteschleife, bis 'play' oder stop gedrückt <===
07947  e9d2
07948  e9d2  20 8f ea  cs40  jsr tstop       Test auf STOPP-Taste (Motor aus, IRQ
                                             setzen)
07949  e9d5  20 df e9        jsr cs10        gedrückte Taste abfragen
07950  e9d8  d0 f8           bne cs40        Warteschleife, bis 'play' oder stop
                                             gedrückt
07951  e9da
07952  e9da  a0 6a           ldy #106        Zeiger auf 'ok'
07953  e9dc  4c 22 f7        jmp msg         Ausgabe Systemmeldungen, yr=offset
07954  e9df
07955  e9df
07956  e9df  ===> gedrückte Taste abfragen <===
07957  e9df
07958  e9df  a9 10     cs10  lda #%00010000 Bit4: Kass.rek. Taste
07959  e9e1  24 01           bit r8502       8502 CPU-Port Datenregister
07960  e9e3  d0 02           bne cs25        nicht gedrückt
07961  e9e5
```

```
07962  e9e5  24 01            bit r8502       8502 CPU-Port Datenregister
07963  e9e7  18        cs25   clc
07964  e9e8  60               rts
07965  e9e9
07966  e9e9
07967  e9e9  ===> prüfen, Meldungen ausgeben, warten auf 'play & record' <===
07968  e9e9
07969  e9e9  20 df e9   cste2  jsr cs10        gedrückte Taste abfragen
07970  e9ec  f0 f9            beq cs25        gedrückt
07971  e9ee
07972  e9ee  a0 2e            ldy #46         Zeiger auf 'press record & play'
07973  e9f0  d0 dd            bne cs30        immer
07974  e9f2
07975  e9f2
07976  e9f2
07977  e9f2  ===> Block von Kassette in Puffer einlesen <===
07978  e9f2
07979  e9f2  a9 00      rblk  lda #%00000000  Vorbesetzen
07980  e9f4  85 90            sta status      I/O-Status 'st'
07981  e9f6  85 93            sta verck       Flag für load/verify
07982  e9f8
07983  e9f8  20 87 e9         jsr ldad1       Zeiger auf Anfang & Ende
                                               Kassettenpuffer einrichten
07984  e9fb
07985  e9fb
07986  e9fb  ===> Lesen von Kassette <===
07987  e9fb
07988  e9fb  20 c8 e9   trd   jsr cste1       auf Taste warten, Ausgabe 'press
                                               play on tape'
07989  e9fe  b0 1f            bcs twrt3       Stopp-Taste gedrückt, Abbruch
07990  ea00
07991  ea00  78               sei
07992  ea01  a9 00            lda #%00000000  Vorbesetzen
07993  ea03  85 aa            sta ridata      RS 232 Eingabe Byte
07994  ea05  85 b4            sta snsw1       RS 232 Ausgabe Bitzähler
07995  ea07  85 b0            sta cmpo        Timing-Wert für Kassette
07996  ea09  85 9e            sta ptr1        Fehler Paß 1
07997  ea0b  85 9f            sta ptr2        Fehler Paß 2
07998  ea0d  85 9c            sta dpsw        Flag für Byte-Ausgabe Kassette
07999  ea0f
08000  ea0f  a9 90            lda #%10010000  Bitmuster für CIA1 IRQ
08001  ea11  a2 0e            ldx #14         Zeiger IRQ-Vector
08002  ea13  d0 11            bne tape        immer
08003  ea15
08004  ea15
08005  ea15  ===> Kassettenpuffer auf Kassette schreiben <===
08006  ea15
08007  ea15  20 87 e9   wblk  jsr ldad1       Zeiger auf Anfang & Ende
                                               Kassettenpuffer einrichten
08008  ea18
08009  ea18
08010  ea18  ===> PRG-file auf Kassette schreiben <===
08011  ea18
08012  ea18  a9 14      twrt  lda #%00010100
08013  ea1a  85 ab            sta riprty      RS 232 Eingabe-Parität
08014  ea1c
08015  ea1c
```

```
08016  ea1c  ===> Schreiben auf Kassette <===
08017  ea1c
08018  ea1c  20 e9 e9   twrt2  jsr cste2       prüfen, Meldungen ausgeben, warten
                                               auf 'play & record'
08019  ea1f  b0 7a      twrt3  bcs stop3       Stopp-Taste gedrückt
08020  ea21
08021  ea21  78                sei
08022  ea22  a9 82             lda #%10000010  Bitmuster für CIA1 IRQ
08023  ea24  a2 08             ldx #8          Zeiger IRQ-Vector
08024  ea26  a0 00      tape   ldy #%00000000
08025  ea28  8c 1a d0          sty vicier      VIC 8564 Interrupt Enable Register
08026  ea2b
08027  ea2b  88                dey             %11111111
08028  ea2c  8c 19 d0          sty vicifr      VIC 8564 Interrupt Flag-Register
08029  ea2f
08030  ea2f  8d 0d dc          sta d1icr       CIA1 Interrupt Control-Register
08031  ea32
08032  ea32  ad 0e dc          lda d1cra       CIA1 Control-Register a
08033  ea35  09 19             ora #%00011001
08034  ea37  8d 0f dc          sta d1crb       CIA1 Control-Register b
08035  ea3a
08036  ea3a  29 91             and #%10010001
08037  ea3c  8d 0b 0a          sta caston      Zeitvergleich für Kass.-Operat.
08038  ea3f
08039  ea3f  20 ec e7          jsr rsp232      Test / Warten auf Ende RS 232 I/O
08040  ea42
08041  ea42  ad 11 d0          lda viccr1      VIC 8564 Control-Register 1
08042  ea45  a8                tay
08043  ea46  29 10             and #%00010000
08044  ea48  8d 39 0a          sta blnkng
08045  ea4b
08046  ea4b  98                tya
08047  ea4c  29 6f             and #%01101111
08048  ea4e  8d 11 d0          sta viccr1      VIC 8564 Control-Register 1
08049  ea51
08050  ea51  20 74 e5          jsr getsp       Test auf / Sprites ausschalten
08051  ea54  ad 14 03          lda iirq        Hardware-IRQ ($fa65)
08052  ea57  8d 09 0a          sta irqtmp      Puffer für addrl IRQ Kassette
08053  ea5a
08054  ea5a  ad 15 03          lda iirq+1      Hardware-IRQ ($fa65)
08055  ea5d  8d 0a 0a          sta irqtmp+1    Puffer für addrh IRQ Kassette
08056  ea60
08057  ea60  20 9b ee          jsr bsiv        IRQ-Vector x-indiziert aus Tab.
                                               setzen
08058  ea63  a9 02             lda #2
08059  ea65  85 be             sta fsblk       Blockzähler Kassette
08060  ea67
08061  ea67  20 5a ed          jsr newch       Register für ser. Lesen/Schreiben
                                               initialisieren
08062  ea6a
08063  ea6a  a5 01             lda r8502       8502 CPU-Port Datenregister
08064  ea6c  29 1f             and #%00011111
08065  ea6e  85 01             sta r8502       8502 CPU-Port Datenregister
08066  ea70
08067  ea70  85 c0             sta cas1        Flag für Schalter Kass.taste
08068  ea72
```

```
08069  ea72  ===> Warteschleife 0.3 s für Bandanlauf <===
08070  ea72
08071  ea72  a2 ff            ldx #$ff
08072  ea74  a0 ff      tp32  ldy #$ff
08073  ea76  88         tp35  dey
08074  ea77  d0 fd            bne tp35
08075  ea79  ca               dex
08076  ea7a  d0 f8            bne tp32
08077  ea7c
08078  ea7c  58               cli
08079  ea7d
08080  ea7d
08081  ea7d  ===> I/O-Abschluss abwarten <===
08082  ea7d
08083  ea7d  ad 0a 0a   tp40  lda irqtmp+1     Puffer für addrh IRQ Kassette
08084  ea80  cd 15 03         cmp iirq+1      Hardware-IRQ ($fa65)
08085  ea83  18               clc
08086  ea84  f0 15            beq stop3
08087  ea86
08088  ea86  20 8f ea         jsr tstop       Test auf STOPP-Taste (Motor aus, IRQ
                                              setzen)
08089  ea89  20 3d f6         jsr ud60        Abfrage Stopp/Shift-Taste/Spalte7
08090  ea8c  4c 7d ea         jmp tp40        I/O-Abschluss abwarten
08091  ea8f
08092  ea8f
08093  ea8f  ===> Test auf STOPP-Taste (Motor aus, IRQ setzen) <===
08094  ea8f
08095  ea8f  20 e1 ff   tstop  jsr kstop       Stopp-Taste üb. Vector abfragen
                                              (Sprungv.)
08096  ea92  18               clc
08097  ea93  d0 0b            bne stop4
08098  ea95
08099  ea95  ===> Motor aus, IRQ normal <===
08100  ea95
08101  ea95  20 57 ee         jsr tnif        Kassettenmotor aus, IRQ rücksetzen
08102  ea98  38               sec             Abbruch-Flag
08103  ea99  68               pla             Rücksprungadresse
08104  ea9a  68               pla             verwerfen
08105  ea9b  a9 00      stop3 lda #0
08106  ea9d  8d 0a 0a         sta irqtmp+1    Puffer für addrh IRQ Kassette
08107  eaa0
08108  eaa0  60         stop4 rts
08109  eaa1
08110  eaa1
08111  eaa1  ===> Kassette für Lesen vorbereiten <===
08112  eaa1
08113  eaa1  86 b1      stt1  stx temp
08114  eaa3
08115  eaa3  a5 b0            lda cmpo        Timing-Wert für Kassette
08116  eaa5  0a               asl a
08117  eaa6  0a               asl a
08118  eaa7  18               clc
08119  eaa8  65 b0            adc cmpo        Timing-Wert für Kassette
08120  eaaa  18               clc
08121  eaab  65 b1            adc temp
08122  eaad  85 b1            sta temp
08123  eaaf
08124  eaaf  a9 00            lda #%00000000
```

```
08125  eab1  24 b0           bit cmpo        Timing-Wert für Kassette
08126  eab3  30 01           bmi stt2
08127  eab5
08128  eab5  2a              rol a
08129  eab6  06 b1     stt2  asl temp
08130  eab8
08131  eab8  2a              rol a
08132  eab9  06 b1           asl temp
08133  eabb
08134  eabb  2a              rol a
08135  eabc  aa              tax
08136  eabd  ad 06 dc   stt3  lda d1t2l       CIA1 TIMER B low
08137  eac0  c9 16           cmp #22
08138  eac2  90 f9           bcc stt3
08139  eac4
08140  eac4  65 b1           adc temp
08141  eac6  8d 04 dc        sta d1t1l       CIA1 TIMER A low
08142  eac9
08143  eac9  8a              txa
08144  eaca  6d 07 dc        adc d1t2h       CIA1 TIMER B high
08145  eacd  8d 05 dc        sta d1t1h       CIA1 TIMER A high
08146  ead0
08147  ead0  ad 0b 0a        lda caston      Zeitvergleich für Kass.-Operat.
08148  ead3  8d 0e dc        sta d1cra       CIA1 Control-Register a
08149  ead6  8d 0d 0a        sta stupid      temporäres Register Kassette
08150  ead9
08151  ead9  ad 0d dc        lda d1icr       CIA1 Interrupt Control-Register
08152  eadc  29 10           and #%00010000
08153  eade  f0 09           beq stt4
08154  eae0
08155  eae0  a9 ea           lda #>stt4
08156  eae2  48              pha
08157  eae3
08158  eae3  a9 e9           lda #<stt4
08159  eae5  48              pha
08160  eae6
08161  eae6  4c c8 ee        jmp simirq      Interrupt simulieren
08162  eae9
08163  eae9
08164  eae9  58        stt4  cli
08165  eaea  60              rts
08166  eaeb
08167  eaeb
08168  eaeb  ===> Seriell von Kassette lesen (IRQ) <===
08169  eaeb
08170  eaeb  ae 07 dc   read  ldx d1t2h       CIA1 TIMER B high
08171  eaee  a0 ff           ldy #$ff
08172  eaf0  98              tya
08173  eaf1  ed 06 dc        sbc d1t2l       CIA1 TIMER B low
08174  eaf4  ec 07 dc        cpx d1t2h       CIA1 TIMER B high
08175  eaf7  d0 f2           bne read        Seriell von Kassette lesen (IRQ)
08176  eaf9
08177  eaf9  86 b1           stx temp
08178  eafb
08179  eafb  aa              tax
08180  eafc  8c 06 dc        sty d1t2l       CIA1 TIMER B low
08181  eaff  8c 07 dc        sty d1t2h       CIA1 TIMER B high
08182  eb02
```

```
08183  eb02  a9 19          lda #%00011001
08184  eb04  8d 0f dc       sta d1crb       CIA1 Control-Register b
08185  eb07
08186  eb07  ad 0d dc       lda d1icr       CIA1 Interrupt Control-Register
08187  eb0a  8d 0c 0a       sta kika26      temporäres Register Kassette
08188  eb0d
08189  eb0d  98             tya
08190  eb0e  e5 b1          sbc temp
08191  eb10  86 b1          stx temp
08192  eb12
08193  eb12  4a             lsr a
08194  eb13  66 b1          ror temp
08195  eb15  4a             lsr a
08196  eb16  66 b1          ror temp
08197  eb18  a5 b0          lda cmpo        Timing-Wert für Kassette
08198  eb1a  18             clc
08199  eb1b  69 3c          adc #$3c
08200  eb1d  c5 b1          cmp temp
08201  eb1f  b0 4a          bcs rdbk
08202  eb21
08203  eb21  a6 9c          ldx dpsw        Flag für Byte-Ausgabe Kassette
08204  eb23  f0 03          beq rjdj
08205  eb25
08206  eb25  4c 1f ec       jmp radj        Bytes abspeichern
08207  eb28
08208  eb28
08209  eb28  a6 a3    rjdj  ldx pcntr
08210  eb2a  30 1b          bmi jrad2
08211  eb2c
08212  eb2c  a2 00          ldx #$00
08213  eb2e  69 30          adc #$30
08214  eb30  65 b0          adc cmpo        Timing-Wert für Kassette
08215  eb32  c5 b1          cmp temp
08216  eb34  b0 1c          bcs radx2
08217  eb36
08218  eb36  e8             inx
08219  eb37  69 26          adc #$26
08220  eb39  65 b0          adc cmpo        Timing-Wert für Kassette
08221  eb3b  c5 b1          cmp temp
08222  eb3d  b0 17          bcs radl
08223  eb3f
08224  eb3f  69 2c          adc #$2c
08225  eb41  65 b0          adc cmpo        Timing-Wert für Kassette
08226  eb43  c5 b1          cmp temp
08227  eb45  90 03          bcc srer
08228  eb47
08229  eb47  4c cf eb  jrad2 jmp rad2
08230  eb4a
08231  eb4a
08232  eb4a  a5 b4    srer  lda snsw1       RS 232 Ausgabe Bitzähler
08233  eb4c  f0 1d          beq rdbk
08234  eb4e
08235  eb4e  85 a8          sta bitci       RS 232 Bitzähler
08236  eb50  d0 19          bne rdbk
08237  eb52
08238  eb52
08239  eb52  e6 a9    radx2 inc rinone      RS 232 Flag für Startbit
08240  eb54  b0 02          bcs rad5
```

```
08241  eb56
08242  eb56  c6 a9      radl   dec rinone      RS 232 Flag für Startbit
08243  eb58  38         rad5   sec
08244  eb59  e9 13             sbc #$13
08245  eb5b  e5 b1             sbc temp
08246  eb5d  65 92             adc svxt        Kassette Timing-Konstante
08247  eb5f  85 92             sta svxt        Kassette Timing-Konstante
08248  eb61
08249  eb61  a5 a4             lda bsour1
08250  eb63  49 01             eor #%00000001
08251  eb65  85 a4             sta bsour1
08252  eb67
08253  eb67  f0 2b             beq rad3
08254  eb69  86 c5             stx data        Kassetten Lese- und Schreibdaten
08255  eb6b
08256  eb6b  a5 b4      rdbk   lda snsw1       RS 232 Ausgabe Bitzähler
08257  eb6d  f0 22             beq radbk
08258  eb6f
08259  eb6f  ad 0c 0a          lda kika26      temporäres Register Kassette
08260  eb72  29 01             and #%00000001
08261  eb74  d0 05             bne radkx
08262  eb76
08263  eb76  ad 0d 0a          lda stupid      temporäres Register Kassette
08264  eb79  d0 16             bne radbk
08265  eb7b
08266  eb7b  a9 00      radkx  lda #0
08267  eb7d  85 a4             sta bsour1
08268  eb7f  8d 0d 0a          sta stupid      temporäres Register Kassette
08269  eb82
08270  eb82  a5 a3             lda pcntr
08271  eb84  10 30             bpl rad4
08272  eb86  30 bf             bmi jrad2
08273  eb88
08274  eb88
08275  eb88  a2 a6      eadp   ldx #166
08276  eb8a  20 a1 ea          jsr stt1        Kassette für Lesen vorbereiten
08277  eb8d
08278  eb8d  a5 9b             lda prty        Paritäts-Bit Kassettenoperation
08279  eb8f  d0 b9             bne srer
08280  eb91
08281  eb91  4c 33 ff  radbk   jmp prend
08282  eb94
08283  eb94
08284  eb94  a5 92      rad3   lda svxt        Kassette Timing-Konstante
08285  eb96  f0 07             beq rout1
08286  eb98  30 03             bmi rout2
08287  eb9a
08288  eb9a  c6 b0             dec cmpo        Timing-Wert für Kassette
08289  eb9c  2c                .byte skip2
08290  eb9d
08291  eb9d  e6 b0      rout2  inc cmpo        Timing-Wert für Kassette
08292  eb9f  a9 00      rout1  lda #0
08293  eba1  85 92             sta svxt        Kassette Timing-Konstante
08294  eba3
08295  eba3  e4 c5             cpx data        Kassetten Lese- und Schreibdaten
08296  eba5  d0 0f             bne rad4
08297  eba7
08298  eba7  8a                txa
```

```
08299  eba8  d0 a0           bne srer
08300  ebaa
08301  ebaa  a5 a9           lda rinone      RS 232 Flag für Startbit
08302  ebac  30 bd           bmi rdbk
08303  ebae
08304  ebae  c9 10           cmp #$10
08305  ebb0  90 b9           bcc rdbk
08306  ebb2
08307  ebb2  85 96           sta syno        Kassette Synchronisations-Wert
08308  ebb4  b0 b5           bcs rdbk
08309  ebb6
08310  ebb6  8a        rad4  txa
08311  ebb7  45 9b           eor prty        Paritäts-Bit Kassettenoperation
08312  ebb9  85 9b           sta prty        Paritäts-Bit Kassettenoperation
08313  ebbb
08314  ebbb  a5 b4           lda snsw1       RS 232 Ausgabe Bitzähler
08315  ebbd  f0 d2           beq radbk
08316  ebbf
08317  ebbf  c6 a3           dec pcntr
08318  ebc1  30 c5           bmi eadp
08319  ebc3
08320  ebc3  46 c5           lsr data        Kassetten Lese- und Schreibdaten
08321  ebc5  66 bf           ror mych        Puffer für serielles Zeichen
08322  ebc7  a2 da           ldx #$da
08323  ebc9  20 a1 ea        jsr stt1        Kassette für Lesen vorbereiten
08324  ebcc  4c 33 ff        jmp prend
08325  ebcf
08326  ebcf
08327  ebcf  a5 96     rad2  lda syno        Kassette Synchronisations-Wert
08328  ebd1  f0 04           beq rad2y
08329  ebd3
08330  ebd3  a5 b4           lda snsw1       RS 232 Ausgabe Bitzähler
08331  ebd5  f0 07           beq rad2x
08332  ebd7
08333  ebd7  a5 a3     rad2y lda pcntr
08334  ebd9  30 03           bmi rad2x
08335  ebdb
08336  ebdb  4c 56 eb        jmp radl
08337  ebde
08338  ebde
08339  ebde  46 b1     rad2x lsr temp
08340  ebe0  a9 93           lda #$93
08341  ebe2  38              sec
08342  ebe3  e5 b1           sbc temp
08343  ebe5  65 b0           adc cmpo        Timing-Wert für Kassette
08344  ebe7  0a              asl a
08345  ebe8  aa              tax
08346  ebe9  20 a1 ea        jsr stt1        Kassette für Lesen vorbereiten
08347  ebec  e6 9c           inc dpsw        Flag für Byte-Ausgabe Kassette
08348  ebee  a5 b4           lda snsw1       RS 232 Ausgabe Bitzähler
08349  ebf0  d0 11           bne radq2
08350  ebf2
08351  ebf2  a5 96           lda syno        Kassette Synchronisations-Wert
08352  ebf4  f0 26           beq rdbk2
08353  ebf6
08354  ebf6  85 a8           sta bitci       RS 232 Bitzähler
08355  ebf8  a9 00           lda #0
08356  ebfa  85 96           sta syno        Kassette Synchronisations-Wert
```

```
08357  ebfc
08358  ebfc  a9 81           lda #%10000001
08359  ebfe  8d 0d dc        sta d1icr        CIA1 Interrupt Control-Register
08360  ec01  85 b4           sta snsw1        RS 232 Ausgabe Bitzähler
08361  ec03
08362  ec03  a5 96     radq2 lda syno         Kassette Synchronisations-Wert
08363  ec05  85 b5           sta nxtbit       RS 232 nächstes Ausgabe-Bit
08364  ec07  f0 09           beq radk
08365  ec09
08366  ec09  a9 00           lda #%00000000
08367  ec0b  85 b4           sta snsw1        RS 232 Ausgabe Bitzähler
08368  ec0d
08369  ec0d  a9 01           lda #%00000001
08370  ec0f  8d 0d dc        sta d1icr        CIA1 Interrupt Control-Register
08371  ec12
08372  ec12  a5 bf     radk  lda mych         Puffer für serielles Zeichen
08373  ec14  85 bd           sta ochar        RS 232 Ausgabe-Parität
08374  ec16
08375  ec16  a5 a8           lda bitci        RS 232 Bitzähler
08376  ec18  05 a9           ora rinone       RS 232 Flag für Startbit
08377  ec1a  85 b6           sta rodata       RS 232 Ausgabe Byte-Puffer
08378  ec1c
08379  ec1c  4c 33 ff  rdbk2 jmp prend
08380  ec1f
08381  ec1f
08382  ec1f                  .end
08383  ec1f                  .lib c128.tape2
08384  ec1f
08386  ec1f
08387  ec1f
08388  ec1f  ===> Bytes abspeichern <===
08389  ec1f
08390  ec1f  20 5a ed  radj  jsr newch        Register für ser. Lesen/Schreiben
                             initialisieren
08391  ec22  85 9c           sta dpsw         Flag für Byte-Ausgabe Kassette
08392  ec24
08393  ec24  a2 da           ldx #$da
08394  ec26  20 a1 ea        jsr stt1         Kassette für Lesen vorbereiten
08395  ec29  a5 be           lda fsblk        Blockzähler Kassette
08396  ec2b  f0 02           beq rd15
08397  ec2d
08398  ec2d  85 a7           sta inbit
08399  ec2f
08400  ec2f  a9 0f     rd15  lda #%00001111
08401  ec31  24 aa           bit ridata       RS 232 Eingabe Byte
08402  ec33  10 17           bpl rd20
08403  ec35
08404  ec35  a5 b5           lda nxtbit       RS 232 nächstes Ausgabe-Bit
08405  ec37  d0 0c           bne rd12
08406  ec39
08407  ec39  a6 be           ldx fsblk        Blockzähler Kassette
08408  ec3b  ca              dex
08409  ec3c  d0 0b           bne rd10
08410  ec3e
08411  ec3e  a9 08           lda #%00001000
08412  ec40  20 57 f7        jsr udst         ac-Bits in Status setzen
08413  ec43  d0 04           bne rd10
08414  ec45
```

```
08415  ec45  a9 00      rd12   lda #%00000000
08416  ec47  85 aa             sta ridata       RS 232 Eingabe Byte
08417  ec49
08418  ec49  4c 33 ff  rd10   jmp prend
08419  ec4c
08420  ec4c
08421  ec4c  70 31      rd20   bvs rd60
08422  ec4e  d0 18             bne rd200
08423  ec50
08424  ec50  a5 b5             lda nxtbit       RS 232 nächstes Ausgabe-Bit
08425  ec52  d0 f5             bne rd10
08426  ec54
08427  ec54  a5 b6             lda rodata       RS 232 Ausgabe Byte-Puffer
08428  ec56  d0 f1             bne rd10
08429  ec58
08430  ec58  a5 a7             lda inbit
08431  ec5a  4a                lsr a
08432  ec5b  a5 bd             lda ochar        RS 232 Ausgabe-Parität
08433  ec5d  30 03             bmi rd22
08434  ec5f  90 18             bcc rd40
08435  ec61
08436  ec61  18                clc
08437  ec62  b0 15      rd22   bcs rd40
08438  ec64
08439  ec64  29 0f             and #%00001111
08440  ec66  85 aa             sta ridata       RS 232 Eingabe Byte
08441  ec68
08442  ec68  c6 aa      rd200  dec ridata       RS 232 Eingabe Byte
08443  ec6a  d0 dd             bne rd10
08444  ec6c
08445  ec6c  a9 40             lda #%01000000
08446  ec6e  85 aa             sta ridata       RS 232 Eingabe Byte
08447  ec70
08448  ec70  20 51 ed          jsr rd300        stah swap sah
08449  ec73  a9 00             lda #%00000000
08450  ec75  85 ab             sta riprty       RS 232 Eingabe-Parität
08451  ec77  f0 d0             beq rd10
08452  ec79
08453  ec79  a9 80      rd40   lda #%10000000
08454  ec7b  85 aa             sta ridata       RS 232 Eingabe Byte
08455  ec7d  d0 ca             bne rd10
08456  ec7f
08457  ec7f  a5 b5      rd60   lda nxtbit       RS 232 nächstes Ausgabe-Bit
08458  ec81  f0 0a             beq rd70
08459  ec83
08460  ec83  a9 04             lda #%00000100
08461  ec85  20 57 f7          jsr udst         ac-Bits in Status setzen
08462  ec88  a9 00             lda #0
08463  ec8a  4c 0c ed          jmp rd161
08464  ec8d
08465  ec8d
08466  ec8d  20 b7 ee  rd70    jsr cmpste       Adresse 'akt./ende' vergleichen
08467  ec90  90 03             bcc rd71
08468  ec92
08469  ec92  4c 0a ed          jmp rd160
08470  ec95
08471  ec95
08472  ec95  a6 a7      rd71   ldx inbit
```

```
08473  ec97  ca              dex
08474  ec98  f0 2e           beq rd58
08475  ec9a
08476  ec9a  a5 93           lda verck       Flag für load/verify
08477  ec9c  f0 0d           beq rd80
08478  ec9e
08479  ec9e  a0 00           ldy #0
08480  eca0  20 cc f7        jsr sally       Byte indirekt über 'sal' lesen
08481  eca3  c5 bd           cmp ochar       RS 232 Ausgabe-Parität
08482  eca5  f0 04           beq rd80
08483  eca7
08484  eca7  a9 01           lda #%00000001
08485  eca9  85 b6           sta rodata      RS 232 Ausgabe Byte-Puffer
08486  ecab
08487  ecab  a5 b6     rd80  lda rodata      RS 232 Ausgabe Byte-Puffer
08488  ecad  f0 4c           beq rd59
08489  ecaf
08490  ecaf  a2 3d           ldx #$3d
08491  ecb1  e4 9e           cpx ptr1
08492  ecb3  90 3f           bcc rd55
08493  ecb5
08494  ecb5  a6 9e           ldx ptr1
08495  ecb7  a5 ad           lda sah         addrh für load/save
08496  ecb9  9d 01 01        sta stack+1,x   8502 CPU-Stapel
08497  ecbc
08498  ecbc  a5 ac           lda sal         addrl für load/save
08499  ecbe  9d 00 01        sta stack,x     8502 CPU-Stapel
08500  ecc1
08501  ecc1  e8              inx
08502  ecc2  e8              inx
08503  ecc3  86 9e           stx ptr1
08504  ecc5
08505  ecc5  4c fb ec        jmp rd59
08506  ecc8
08507  ecc8
08508  ecc8  a6 9f     rd58  ldx ptr2
08509  ecca  e4 9e           cpx ptr1
08510  eccc  f0 37           beq rd90
08511  ecce
08512  ecce  a5 ac           lda sal         addrl für load/save
08513  ecd0  dd 00 01        cmp stack,x     8502 CPU-Stapel
08514  ecd3  d0 30           bne rd90
08515  ecd5
08516  ecd5  a5 ad           lda sah         addrh für load/save
08517  ecd7  dd 01 01        cmp stack+1,x   8502 CPU-Stapel
08518  ecda  d0 29           bne rd90
08519  ecdc
08520  ecdc  e6 9f           inc ptr2
08521  ecde  e6 9f           inc ptr2
08522  ece0  a5 93           lda verck       Flag für load/verify
08523  ece2  f0 0c           beq rd52
08524  ece4
08525  ece4  a0 00           ldy #0
08526  ece6  20 cc f7        jsr sally       Byte indirekt über 'sal' lesen
08527  ece9  c5 bd           cmp ochar       RS 232 Ausgabe-Parität
08528  eceb  f0 18           beq rd90
08529  eced
08530  eced  c8              iny
```

```
08531  ecee  84 b6          sty rodata      RS 232 Ausgabe Byte-Puffer
08532  ecf0
08533  ecf0  a5 b6    rd52  lda rodata      RS 232 Ausgabe Byte-Puffer
08534  ecf2  f0 07          beq rd59
08535  ecf4
08536  ecf4  a9 10    rd55  lda #%00010000
08537  ecf6  20 57 f7       jsr udst        ac-Bits in Status setzen
08538  ecf9  d0 0a          bne rd90
08539  ecfb
08540  ecfb  a5 93    rd59  lda verck      Flag für load/verify
08541  ecfd  d0 06          bne rd90
08542  ecff
08543  ecff  a8             tay
08544  ed00  a5 bd          lda ochar       RS 232 Ausgabe-Parität
08545  ed02  20 bc f7       jsr salsy       Byte indirekt über 'sal' speichern
08546  ed05  20 c1 ee rd90  jsr incsal      'load/store-Adresse' erhöhen
08547  ed08  d0 44          bne rd180
08548  ed0a
08549  ed0a  a9 80    rd160 lda #%10000000
08550  ed0c  85 aa    rd161 sta ridata      RS 232 Eingabe Byte
08551  ed0e
08552  ed0e  78             sei
08553  ed0f  a2 01          ldx #%00000001
08554  ed11  8e 0d dc       stx d1icr       CIA1 Interrupt Control-Register
08555  ed14
08556  ed14  ae 0d dc       ldx d1icr       CIA1 Interrupt Control-Register
08557  ed17  a6 be          ldx fsblk       Blockzähler Kassette
08558  ed19  ca             dex
08559  ed1a  30 02          bmi rd167
08560  ed1c
08561  ed1c  86 be          stx fsblk       Blockzähler Kassette
08562  ed1e
08563  ed1e  c6 a7    rd167 dec inbit
08564  ed20  f0 08          beq rd175
08565  ed22
08566  ed22  a5 9e          lda ptr1
08567  ed24  d0 28          bne rd180
08568  ed26
08569  ed26  85 be          sta fsblk       Blockzähler Kassette
08570  ed28  f0 24          beq rd180
08571  ed2a
08572  ed2a
08573  ed2a  20 57 ee rd175 jsr tnif        Kassettenmotor aus, IRQ rücksetzen
08574  ed2d  20 51 ed       jsr rd300       stah swap sah
08575  ed30  a0 00          ldy #%00000000
08576  ed32  84 ab          sty riprty      RS 232 Eingabe-Parität
08577  ed34
08578  ed34  20 cc f7 vprty jsr sally       Byte indirekt über 'sal' lesen
08579  ed37  45 ab          eor riprty      RS 232 Eingabe-Parität
08580  ed39  85 ab          sta riprty      RS 232 Eingabe-Parität
08581  ed3b
08582  ed3b  20 c1 ee       jsr incsal      'load/store-Adresse' erhöhen
08583  ed3e  20 b7 ee       jsr cmpste      Adresse 'akt./ende' vergleichen
08584  ed41  90 f1          bcc vprty
08585  ed43
08586  ed43  a5 ab          lda riprty      RS 232 Eingabe-Parität
08587  ed45  45 bd          eor ochar       RS 232 Ausgabe-Parität
08588  ed47  f0 05          beq rd180
```

```
08589  ed49
08590  ed49  a9 20           lda #%00100000
08591  ed4b  20 57 f7        jsr udst          ac-Bits in Status setzen
08592  ed4e  4c 33 ff  rd180 jmp prend
08593  ed51
08594  ed51
08595  ed51  ===> stah swap sah <===
08596  ed51
08597  ed51  a5 c2     rd300 lda stah          addrh Startadresse save
08598  ed53  85 ad           sta sah           addrh für load/save
08599  ed55
08600  ed55  a5 c1           lda stal          addrl Startadresse save
08601  ed57  85 ac           sta sal           addrl für load/save
08602  ed59
08603  ed59  60              rts
08604  ed5a
08605  ed5a
08606  ed5a  ===> Register für ser. Lesen/Schreiben initialisieren <===
08607  ed5a
08608  ed5a  a9 08     newch lda #$08
08609  ed5c  85 a3           sta pcntr
08610  ed5e
08611  ed5e  a9 00           lda #0
08612  ed60  85 a4           sta bsour1
08613  ed62  85 a8           sta bitci         RS 232 Bitzähler
08614  ed64  85 9b           sta prty          Paritäts-Bit Kassettenoperation
08615  ed66  85 a9           sta rinone        RS 232 Flag für Startbit
08616  ed68
08617  ed68  60              rts
08618  ed69
08619  ed69
08620  ed69
08621  ed69  ===> ein Bit auf Kassette ausgeben <===
08622  ed69
08623  ed69  a5 bd     write lda ochar         RS 232 Ausgabe-Parität
08624  ed6b  4a              lsr a
08625  ed6c  a9 60           lda #%01100000
08626  ed6e  90 02           bcc wrt1
08627  ed70
08628  ed70  a9 b0     wrtw  lda #%10110000
08629  ed72  a2 00     wrt1  ldx #%00000000
08630  ed74  8d 06 dc  wrtx  sta d1t2l         CIA1 TIMER B low
08631  ed77  8e 07 dc        stx d1t2h         CIA1 TIMER B high
08632  ed7a
08633  ed7a  ad 0d dc        lda d1icr         CIA1 Interrupt Control-Register
08634  ed7d
08635  ed7d  a9 19           lda #%00011001
08636  ed7f  8d 0f dc        sta d1crb         CIA1 Control-Register b
08637  ed82
08638  ed82  a5 01           lda r8502         8502 CPU-Port Datenregister
08639  ed84  49 08           eor #%00001000
08640  ed86  85 01           sta r8502         8502 CPU-Port Datenregister
08641  ed88
08642  ed88  29 08           and #%00001000
08643  ed8a  60              rts
08644  ed8b
08645  ed8b
```

```
08646  ed8b  ===> Daten schreiben <===
08647  ed8b
08648  ed8b  38          wrtl3  sec
08649  ed8c  66 b6              ror rodata      RS 232 Ausgabe Byte-Puffer
08650  ed8e  30 3c              bmi wrt3
08651  ed90
08652  ed90
08653  ed90  ===> auf Kassette schreiben <===
08654  ed90
08655  ed90  a5 a8       wrtn   lda bitci       RS 232 Bitzähler
08656  ed92  d0 12              bne wrtn1
08657  ed94
08658  ed94  a9 10              lda #%00010000
08659  ed96  a2 01              ldx #%00000001
08660  ed98  20 74 ed           jsr wrtx
08661  ed9b  d0 2f              bne wrt3
08662  ed9d
08663  ed9d  e6 a8              inc bitci       RS 232 Bitzähler
08664  ed9f  a5 b6              lda rodata      RS 232 Ausgabe Byte-Puffer
08665  eda1  10 29              bpl wrt3
08666  eda3
08667  eda3  4c 1b ee           jmp wrnc        nächsten Block schreiben
08668  eda6
08669  eda6
08670  eda6  a5 a9       wrtn1  lda rinone      RS 232 Flag für Startbit
08671  eda8  d0 09              bne wrtn2
08672  edaa
08673  edaa  20 70 ed           jsr wrtw
08674  edad  d0 1d              bne wrt3
08675  edaf
08676  edaf  e6 a9              inc rinone      RS 232 Flag für Startbit
08677  edb1  d0 19              bne wrt3
08678  edb3
08679  edb3  20 69 ed    wrtn2  jsr write       ein Bit auf Kassette ausgeben
08680  edb6  d0 14              bne wrt3
08681  edb8
08682  edb8  a5 a4              lda bsour1
08683  edba  49 01              eor #%00000001
08684  edbc  85 a4              sta bsour1
08685  edbe  f0 0f              beq wrt2
08686  edc0
08687  edc0  a5 bd              lda ochar       RS 232 Ausgabe-Parität
08688  edc2  49 01              eor #%00000001
08689  edc4  85 bd              sta ochar       RS 232 Ausgabe-Parität
08690  edc6
08691  edc6  29 01              and #%00000001
08692  edc8  45 9b              eor prty        Paritäts-Bit Kassettenoperation
08693  edca  85 9b              sta prty        Paritäts-Bit Kassettenoperation
08694  edcc
08695  edcc  4c 33 ff    wrt3   jmp prend
08696  edcf
08697  edcf
08698  edcf  46 bd       wrt2   lsr ochar       RS 232 Ausgabe-Parität
08699  edd1  c6 a3              dec pcntr
08700  edd3
08701  edd3  a5 a3              lda pcntr
08702  edd5  f0 3b              beq wrt4
08703  edd7  10 f3              bpl wrt3
```

```
08704  edd9
08705  edd9  20 5a ed   wrts   jsr newch        Register für ser. Lesen/Schreiben
                                                 initialisieren
08706  eddc  58                 cli
08707  eddd
08708  eddd  a5 a5              lda cntdn
08709  eddf  f0 12              beq wrt6
08710  ede1
08711  ede1  a2 00              ldx #0
08712  ede3  86 c5              stx data         Kassetten Lese- und Schreibdaten
08713  ede5
08714  ede5  c6 a5              dec cntdn
08715  ede7  a6 be              ldx fsblk        Blockzähler Kassette
08716  ede9  e0 02              cpx #2
08717  edeb  d0 02              bne wrt61
08718  eded
08719  eded  09 80              ora #%10000000
08720  edef  85 bd       wrt61  sta ochar        RS 232 Ausgabe-Parität
08721  edf1  d0 d9              bne wrt3
08722  edf3
08723  edf3  20 b7 ee   wrt6   jsr cmpste       Adresse 'akt./ende' vergleichen
08724  edf6  90 0a              bcc wrt7
08725  edf8  d0 91              bne wrtl3        Daten schreiben
08726  edfa
08727  edfa  e6 ad              inc sah          addrh für load/save
08728  edfc  a5 c5              lda data         Kassetten Lese- und Schreibdaten
08729  edfe  85 bd              sta ochar        RS 232 Ausgabe-Parität
08730  ee00  b0 ca              bcs wrt3
08731  ee02
08732  ee02  a0 00       wrt7   ldy #0
08733  ee04  20 cc f7           jsr sally        Byte indirekt über 'sal' lesen
08734  ee07  85 bd              sta ochar        RS 232 Ausgabe-Parität
08735  ee09
08736  ee09  45 c5              eor data         Kassetten Lese- und Schreibdaten
08737  ee0b  85 c5              sta data         Kassetten Lese- und Schreibdaten
08738  ee0d
08739  ee0d  20 c1 ee           jsr incsal       'load/store-Adresse' erhöhen
08740  ee10  d0 ba              bne wrt3
08741  ee12
08742  ee12  a5 9b       wrt4   lda prty         Paritäts-Bit Kassettenoperation
08743  ee14  49 01              eor #%00000001
08744  ee16  85 bd              sta ochar        RS 232 Ausgabe-Parität
08745  ee18
08746  ee18  4c 33 ff   wrtbk  jmp prend
08747  ee1b
08748  ee1b
08749  ee1b  ===> nächsten Block schreiben <===
08750  ee1b
08751  ee1b  c6 be       wrnc   dec fsblk        Blockzähler Kassette
08752  ee1d  d0 03              bne wrend
08753  ee1f
08754  ee1f  20 b0 ee           jsr tnof         Kassettenmotor ausschalten
08755  ee22  a9 50       wrend  lda #80
08756  ee24  85 a7              sta inbit
08757  ee26
08758  ee26  a2 08              ldx #8
08759  ee28  78                 sei
```

```
08760  ee29  20 9b ee          jsr bsiv          IRQ-Vector x-indiziert aus Tab.
                                                 setzen
08761  ee2c  d0 ea             bne wrtbk
08762  ee2e
08763  ee2e
08764  ee2e  ===> 'Null' seriell auf Kassette schreiben <===
08765  ee2e
08766  ee2e  a9 78       wrtz   lda #$78
08767  ee30  20 72 ed          jsr wrt1
08768  ee33  d0 e3             bne wrtbk
08769  ee35
08770  ee35  c6 a7             dec inbit
08771  ee37  d0 df             bne wrtbk
08772  ee39
08773  ee39  20 5a ed          jsr newch         Register für ser. Lesen/Schreiben
                                                 initialisieren
08774  ee3c  c6 ab             dec riprty        RS 232 Eingabe-Parität
08775  ee3e  10 d8             bpl wrtbk
08776  ee40
08777  ee40  a2 0a             ldx #10
08778  ee42  20 9b ee          jsr bsiv          IRQ-Vector x-indiziert aus Tab.
                                                 setzen
08779  ee45  58                cli
08780  ee46  e6 ab             inc riprty        RS 232 Eingabe-Parität
08781  ee48  a5 be             lda fsblk         Blockzähler Kassette
08782  ee4a  f0 49             beq stky          Standard-IRQ-Vector setzen
08783  ee4c
08784  ee4c  20 51 ed          jsr rd300         stah swap sah
08785  ee4f
08786  ee4f  a2 09             ldx #9
08787  ee51  86 a5             stx cntdn
08788  ee53  86 b6             stx rodata        RS 232 Ausgabe Byte-Puffer
08789  ee55  d0 82             bne wrts
08790  ee57
08791  ee57
08792  ee57
08793  ee57  ===> Kassettenmotor aus, IRQ rücksetzen <===
08794  ee57
08795  ee57  08          tnif   php
08796  ee58  78                 sei
08797  ee59
08798  ee59  ad 11 d0           lda viccr1        VIC 8564 Control-Register 1
08799  ee5c  0d 39 0a           ora blnkng
08800  ee5f  29 7f              and #%01111111
08801  ee61  8d 11 d0           sta viccr1        VIC 8564 Control-Register 1
08802  ee64
08803  ee64  2c 3a 0a           bit holdof        Flag 'Sprites abschalten'
08804  ee67  30 16              bmi tnif1
08805  ee69
08806  ee69  2c 37 0a           bit speed         Speicher für akt. CPU-Takt
08807  ee6c  10 11              bpl tnif1
08808  ee6e
08809  ee6e  ad 38 0a           lda sprits        Speicher Einschaltzustand Sprites
08810  ee71  8d 15 d0           sta vicmoe        VIC 8564 Sprite Enable Register
08811  ee74
08812  ee74  ad 37 0a           lda speed         Speicher für akt. CPU-Takt
08813  ee77  8d 30 d0           sta vicspd        VIC 8564 Speed-Register
08814  ee7a
```

```
08815  ee7a  a9 00           lda #0
08816  ee7c  8d 37 0a        sta speed        Speicher für akt. CPU-Takt
08817  ee7f
08818  ee7f  20 b0 ee  tnif1 jsr tnof         Kassettenmotor ausschalten
08819  ee82  20 b8 e1       jsr iokeys       Test auf FSD
08820  ee85  ad 0a 0a       lda irqtmp+1     Puffer für addrh IRQ Kassette
08821  ee88  f0 09          beq tniq
08822  ee8a
08823  ee8a  8d 15 03        sta iirq+1       Hardware-IRQ ($fa65)
08824  ee8d
08825  ee8d  ad 09 0a        lda irqtmp       Puffer für addrl IRQ Kassette
08826  ee90  8d 14 03        sta iirq         Hardware-IRQ ($fa65)
08827  ee93
08828  ee93  28        tniq  plp
08829  ee94  60              rts
08830  ee95
08831  ee95
08832  ee95  ===> Standard-IRQ-Vector setzen <===
08833  ee95
08834  ee95  20 57 ee  stky  jsr tnif         Kassettenmotor aus, IRQ rücksetzen
08835  ee98  4c 33 ff        jmp prend
08836  ee9b
08837  ee9b
08838  ee9b
08839  ee9b  ===> IRQ-Vector x-indiziert aus Tab. setzen <===
08840  ee9b
08841  ee9b  bd a0 ee  bsiv  lda bsit-8,x     Tab. der Kassetten-IRQ-Vectoren
08842  ee9e  8d 14 03        sta iirq         Hardware-IRQ ($fa65)
08843  eea1
08844  eea1  bd a1 ee        lda bsit-7,x     Tab. der Kassetten-IRQ-Vectoren
08845  eea4  8d 15 03        sta iirq+1       Hardware-IRQ ($fa65)
08846  eea7  60              rts
08847  eea8
08848  eea8
08849  eea8
08850  eea8  ===> Tab. der Kassetten-IRQ-Vectoren <===
08851  eea8
08852  eea8  2e ee     bsit  .word wrtz       'Null' seriell auf Kassette
                                              schreiben
08853  eeaa  90 ed           .word wrtn       auf Kassette schreiben
08854  eeac  65 fa           .word nirq       IRQ-Routine
08855  eeae  eb ea           .word read       Seriell von Kassette lesen (IRQ)
08856  eeb0
08857  eeb0
08858  eeb0  ===> Kassettenmotor ausschalten <===
08859  eeb0
08860  eeb0  a5 01     tnof  lda r8502        8502 CPU-Port Datenregister
08861  eeb2  09 20           ora #%00100000
08862  eeb4  85 01           sta r8502        8502 CPU-Port Datenregister
08863  eeb6
08864  eeb6  60              rts
08865  eeb7
08866  eeb7
08867  eeb7  ===> Adresse 'akt./ende' vergleichen <===
08868  eeb7
08869  eeb7  38        cmpste sec
08870  eeb8  a5 ac           lda sal          addrl für load/save
08871  eeba  e5 ae           sbc eal          addrl Ende load/save
```

```
08872  eebc  a5 ad            lda sah          addrh für load/save
08873  eebe  e5 af            sbc eah          addrh Ende load/save
08874  eec0  60               rts
08875  eec1
08876  eec1
08877  eec1  ===> 'load/store-Adresse' erhöhen <===
08878  eec1
08879  eec1  e6 ac      incsal inc sal          addrl für load/save
08880  eec3  d0 02             bne incr
08881  eec5
08882  eec5  e6 ad             inc sah          addrh für load/save
08883  eec7  60         incr   rts
08884  eec8
08885  eec8
08886  eec8  ===> Interrupt simulieren <===
08887  eec8
08888  eec8  08         simirq php
08889  eec9
08890  eec9  68               pla
08891  eeca  29 ef            and #%11101111
08892  eecc  48               pha
08893  eecd
08894  eecd  4c 17 ff         jmp irq          System IRQ-Routine
08895  eed0
08896  eed0
08897  eed0  ===> Abfrage gedrückte Kassetten-Taste <===
08898  eed0
08899  eed0  a5 01      tapirq lda r8502        8502 CPU-Port Datenregister
08900  eed2  29 10             and #%00010000
08901  eed4  f0 0a             beq puls6        Kassettenmotor ausschalten ?
08902  eed6
08903  eed6  a0 00             ldy #0
08904  eed8  84 c0             sty cas1         Flag für Schalter Kass.taste
08905  eeda
08906  eeda  a5 01             lda r8502        8502 CPU-Port Datenregister
08907  eedc  09 20             ora #%00100000
08908  eede  d0 08             bne puls7        Motor ein-/ausschalten
08909  eee0
08910  eee0
08911  eee0  ===> Kassettenmotor ausschalten ? <===
08912  eee0
08913  eee0  a5 c0      puls6  lda cas1         Flag für Schalter Kass.taste
08914  eee2  d0 06             bne puls7r
08915  eee4
08916  eee4  a5 01             lda r8502        8502 CPU-Port Datenregister
08917  eee6  29 df             and #%11011111
08918  eee8
08919  eee8
08920  eee8  ===> Motor ein-/ausschalten <===
08921  eee8
08922  eee8  85 01      puls7  sta r8502        8502 CPU-Port Datenregister
08923  eeea
08924  eeea  60         puls7r rts
08925  eeeb
08926  eeeb
08927  eeeb                    .end
08928  eeeb                    .lib c128.kernal1
08929  eeeb
```

4.1.15 Kernal

```
08931  eeeb
08932  eeeb
08933  eeeb  ===> Zeichen vom aktiven Kanal nach ac <===
08934  eeeb
08935  eeeb  a5 99      ngetin lda dfltn        Vorgabe Eingabe-Gerätenummer (0)
08936  eeed  d0 0a             bne gn10        nicht Tastatur
08937  eeef
08938  eeef  ===> Get aus Tastaturpuffer <===
08939  eeef
08940  eeef  a5 d0             lda ndx         Bytes im Tast.puffer
08941  eef1  05 d1             ora kyndx       Bytes im Fkt.tasten-Puffer
08942  eef3  f0 0f             beq getrts      Puffer leer
08943  eef5
08944  eef5  78                sei
08945  eef6  4c 06 c0          jmp jlp2        Zeichen Tast.puffer nach ac
08946  eef9
08947  eef9
08948  eef9  ===> Test auf Geräteadresse 2 (RS 232) <===
08949  eef9
08950  eef9  c9 02      gn10   cmp #2          Geräteadresse RS 232
08951  eefb  d0 18             bne bn10        nicht RS 232
08952  eefd
08953  eefd
08954  eefd  ===> Zeichen von RS 232 nach ac <===
08955  eefd
08956  eefd  84 97      get232 sty xsav        yr retten
08957  eeff  20 ce e7          jsr bsi232      get von RS 232
08958  ef02  a4 97             ldy xsav        yr zurück
08959  ef04
08960  ef04  18         getrts clc
08961  ef05  60                rts
08962  ef06
08963  ef06
08964  ef06  ===> Zeichen vom aktiven Kanal nach ac <===
08965  ef06
08966  ef06  a5 99      nbasin lda dfltn        Vorgabe Eingabe-Gerätenummer (0)
08967  ef08  d0 0b             bne bn10        nicht Tastatur
08968  ef0a
08969  ef0a  ===> Input Tastatur über BS-Editor <===
08970  ef0a
08971  ef0a  a5 ec             lda pntr        akt. Cursor-Spalte
08972  ef0c  85 e9             sta lstp        Cursor-Spalte vor input
08973  ef0e
08974  ef0e  a5 eb             lda tblx        akt. Cursor-Zeile
08975  ef10  85 e8             sta lsxp        Cursor-Zeile vor input
08976  ef12
08977  ef12  4c 09 c0          jmp jloop5      Zeichen von BS nach ac
08978  ef15
08979  ef15
08980  ef15  ===> Test auf Geräteadresse 3 (Bildschirm) <===
08981  ef15
08982  ef15  c9 03      bn10   cmp #3          Geräteadresse Bildschirm
08983  ef17  d0 09             bne bn20        nicht Bildschirm
08984  ef19
```

```
08985  ef19  ===> Input von Bildschirm <===
08986  ef19
08987  ef19  85 d6            sta crsw        Flag Warten auf CR
08988  ef1b
08989  ef1b  a5 e7            lda scrt        BS-Fenster: Spalte rechter Rand
08990  ef1d  85 ea            sta indx        Spalte letztes Zeichen in Zeile
08991  ef1f
08992  ef1f  4c 09 c0         jmp jloop5      Zeichen von BS nach ac
08993  ef22
08994  ef22
08995  ef22  b0 38      bn20  bcs bn30        Geräteadresse >3 (IEC)
08996  ef24
08997  ef24  c9 02            cmp #2          Geräteadresse RS 232
08998  ef26  f0 3f            beq bn50        Zeichen RS 232-Puffer nach ac
08999  ef28
09000  ef28  ===> Eingabe von Kassettenrekorder <===
09001  ef28
09002  ef28  86 97            stx xsav        xr retten
09003  ef2a
09004  ef2a  20 48 ef         jsr jtget       nächst. Byte aus Kass.puffer nach ac
09005  ef2d  b0 16            bcs jtg37       Fehler, Abschluß
09006  ef2f
09007  ef2f  48               pha             Zeichen auf Stapel
09008  ef30
09009  ef30  20 48 ef         jsr jtget       nächst. Byte aus Kass.puffer nach ac
09010  ef33  b0 0d            bcs jtg36       Fehler
09011  ef35  d0 05            bne jtg35       ok
09012  ef37
09013  ef37  ===> EOF Kassette <===
09014  ef37
09015  ef37  a9 40            lda #%01000000
09016  ef39
09017  ef39   x1xxxxxx  END-OF-FILE
09018  ef39
09019  ef39  20 57 f7         jsr udst        ac-Bits in Status setzen
09020  ef3c
09021  ef3c
09022  ef3c  ===> Abschluß ok <===
09023  ef3c
09024  ef3c  c6 a6      jtg35 dec bufpt       Pufferzeiger erniedrigen
09025  ef3e  a6 97            ldx xsav        xr zurückladen
09026  ef40  68               pla             Zeichen von Stapel
09027  ef41  60               rts
09028  ef42
09029  ef42
09030  ef42  ===> Abschluß Fehler <===
09031  ef42
09032  ef42  aa         jtg36 tax
09033  ef43  68               pla
09034  ef44  8a               txa
09035  ef45
09036  ef45
09037  ef45  ===> xr zurück, Abschluß <===
09038  ef45
09039  ef45  a6 97      jtg37 ldx xsav
09040  ef47  60               rts
09041  ef48
09042  ef48
```

```
09043  ef48  ===> nächst. Byte aus Kass.puffer nach ac <===
09044  ef48
09045  ef48  20 be e9   jtget  jsr jtp20        Test auf Kassettenpuffer voll
09046  ef4b  d0 0b             bne jtg10        Puffer noch nicht ausgelesen
09047  ef4d
09048  ef4d
09049  ef4d  ===> nächsten Block von Kassette lesen <===
09050  ef4d
09051  ef4d  20 f2 e9          jsr rblk         Block von Kassette in Puffer
                                                einlesen
09052  ef50  b0 09             bcs jtg11        Abbruch durch Stopp-Taste
09053  ef52
09054  ef52  a9 00             lda #0           Zeiger in Puffer
09055  ef54  85 a6             sta bufpt        auf erstes Byte
09056  ef56  f0 f0             beq jtget        immer
09057  ef58
09058  ef58
09059  ef58  ===> akt. Byte aus Kass.puffer nach ac <===
09060  ef58
09061  ef58  b1 b2      jtg10  lda (tape1),y    addrl Zeiger Kassettenpuffer
09062  ef5a  18                clc
09063  ef5b  60         jtg11  rts
09064  ef5c
09065  ef5c
09066  ef5c  ===> Eingabe vom IEC-Bus <===
09067  ef5c
09068  ef5c  a5 90      bn30   lda status       I/O-Status 'st'
09069  ef5e  d0 03             bne bn40         Ersatzzeichen cr nach ac
09070  ef60
09071  ef60  4c 3e e4          jmp acptr        Zeichen vom IEC-Bus nach ac
09072  ef63
09073  ef63
09074  ef63  ===> Ersatzzeichen cr nach ac <===
09075  ef63
09076  ef63  a9 0d      bn40   lda #cr          Carriage Return
09077  ef65  18         bn41   clc
09078  ef66  60         bn42   rts
09079  ef67
09080  ef67
09081  ef67  ===> Zeichen RS 232-Puffer nach ac <===
09082  ef67
09083  ef67  20 fd ee   bn50   jsr get232       Zeichen von RS 232 nach ac
09084  ef6a  b0 f9             bcs bn41         ?? nie
09085  ef6c
09086  ef6c  c9 00             cmp #0
09087  ef6e  d0 f6             bne bn42
09088  ef70
09089  ef70  ===> Ersatzzeichen 0 geladen <===
09090  ef70
09091  ef70  ad 14 0a          lda rsstat       RS 232 Status-Byte
09092  ef73  29 60             and #%01100000
09093  ef75
09094  ef75   x1xxxxxx  DSR  DATA SET READY fehlt
09095  ef75   xx1xxxxx  nicht benutzt
09096  ef75
09097  ef75  d0 ec             bne bn40         fehlt, Ersatzzeichen CR
09098  ef77  f0 ee             beq bn50         DSR vorhanden, warten
09099  ef79
```

```
09100  ef79
09101  ef79  ===> ac auf aktivem Kanal ausgeben <===
09102  ef79
09103  ef79  48          nbsout pha              Ausgabezeichen auf Stapel
09104  ef7a
09105  ef7a  a5 9a              lda dflto         Vorgabe Ausgabe-Gerätenummer (3)
09106  ef7c  c9 03              cmp #3            Geräteadresse Bildschirm
09107  ef7e  d0 04              bne bo10          nicht BS
09108  ef80
09109  ef80  ===> Ausgabe auf Bildschirm <===
09110  ef80
09111  ef80  68                 pla
09112  ef81  4c 0c c0           jmp jprint        BS-Ausgabe ac
09113  ef84
09114  ef84
09115  ef84  90 04       bo10   bcc bo20          Ausgabe auf Gerätenummer < 3
09116  ef86
09117  ef86  ===> Ausgabe auf IEC-Bus <===
09118  ef86
09119  ef86  68                 pla               Ausgabezeichen vom Stapel
09120  ef87  4c 03 e5           jmp ciout         Ausgabe ac auf IEC-Bus
09121  ef8a
09122  ef8a
09123  ef8a  ===> Ausgabe auf Gerätenummer < 3 <===
09124  ef8a
09125  ef8a  4a          bo20   lsr a             Bit0 ins Carry: (1=Kass., 0=RS)
09126  ef8b  68                 pla               Ausgabezeichen vom Stapel
09127  ef8c
09128  ef8c  85 9e       casout sta ptr1          Ausgabezeichen
09129  ef8e
09130  ef8e  8a                 txa
09131  ef8f  48                 pha
09132  ef90
09133  ef90  98                 tya
09134  ef91  48                 pha
09135  ef92
09136  ef92  90 23              bcc out232        Ausgabe auf RS 232
09137  ef94
09138  ef94  ===> Ausgabe auf Kassette <===
09139  ef94
09140  ef94  20 be e9           jsr jtp20         Test auf Kassettenpuffer voll
09141  ef97  d0 0e              bne jtp10         Zeichen in Kassettenpuffer schreiben
09142  ef99
09143  ef99  ===> Kassetten-Puffer voll, auf Kassette schreiben <===
09144  ef99
09145  ef99  20 15 ea           jsr wblk          Kassettenpuffer auf Kassette
                                                  schreiben
09146  ef9c  b0 0e              bcs rstor         Abbruch durch Stopp-Taste
09147  ef9e
09148  ef9e  a9 02              lda #2            Headermarke Folge-Datenblock
09149  efa0  a0 00              ldy #0
09150  efa2  91 b2              sta (tape1),y     addrl Zeiger Kassettenpuffer
09151  efa4
09152  efa4  c8                 iny
09153  efa5  84 a6              sty bufpt         Zeiger in Kassettenpuffer
09154  efa7
09155  efa7
```

```
09156  efa7  ===> Zeichen in Kassettenpuffer schreiben <===
09157  efa7
09158  efa7  a5 9e      jtp10  lda ptr1          Ausgabezeichen
09159  efa9  91 b2             sta (tape1),y     in Kassettenpuffer
09160  efab
09161  efab
09162  efab
09163  efab  ===> CPU-Reg. vom Stapel, ok-Flag <===
09164  efab
09165  efab  18         rstoa  clc
09166  efac
09167  efac
09168  efac  ===> CPU-Register vom Stapel, rts <===
09169  efac
09170  efac  68         rstor  pla
09171  efad  a8                tay
09172  efae
09173  efae  68                pla
09174  efaf  aa                tax
09175  efb0
09176  efb0  a5 9e             lda ptr1          Ausgabezeichen
09177  efb2  90 02             bcc rstor1
09178  efb4
09179  efb4  a9 00             lda #0            Ersatzwert bei Fehler
09180  efb6  60         rstor1 rts
09181  efb7
09182  efb7
09183  efb7  ===> Zeichen aus ac nach RS 232 <===
09184  efb7
09185  efb7  20 5f e7   out232 jsr bso232        Ausgabe in RS 232 Puffer
09186  efba  4c ab ef          jmp rstoa         CPU-Reg. vom Stapel, ok-Flag
09187  efbd
09188  efbd
09189  efbd  ===> log. File öffnen, Ausgabe fn <===
09190  efbd
09191  efbd  a6 b8      nopen  ldx la            akt. log. Filenummer
09192  efbf  20 02 f2          jsr lookup        log. Filenummer ac in Tab. suchen
09193  efc2  f0 2f             beq err2          File vorhanden, Fehler
09194  efc4
09195  efc4  a6 98             ldx ldtnd         Anzahl offene Files
09196  efc6  e0 0a             cpx #10           max. Anzahl offener Files
09197  efc8  b0 26             bcs err1          zu viele Files, Fehler
09198  efca
09199  efca  ===> la, fa, sa in Tabellen eintragen <===
09200  efca
09201  efca  e6 98             inc ldtnd         Anzahl offene Files
09202  efcc  a5 b8             lda la            akt. log. Filenummer
09203  efce  9d 62 03          sta lat,x         Tab. der log. Filenummern
09204  efd1
09205  efd1  a5 b9             lda sa            akt. Sekundäradresse
09206  efd3  09 60             ora #%01100000
09207  efd5  85 b9             sta sa            akt. Sekundäradresse
09208  efd7
09209  efd7  9d 76 03          sta sat,x         Tab. der Sekundär-Adr.
09210  efda
09211  efda  a5 ba             lda fa            akt. Gerätenummer
09212  efdc  9d 6c 03          sta fat,x         Tab. der Gerätenummern
09213  efdf  f0 0d             beq opnrts        Abschluß mit ok-Flag
```

```
09214  efe1
09215  efe1  ===> Test auf Kassette, RS 232, IEC <===
09216  efe1
09217  efe1  c9 02           cmp #2          Geräteadresse RS 232
09218  efe3  f0 5b           beq opn232      Open RS 232
09219  efe5  90 0f           bcc opncas      Open Kassettenfile
09220  efe7
09221  efe7  c9 03           cmp #3          Geräteadresse Bildschirm
09222  efe9  f0 03           beq opnrts      Bildschirm öffnen, fertig
09223  efeb
09224  efeb  ===> IEC - File öffnen <===
09225  efeb
09226  efeb  20 cb f0        jsr openi       Open IEC-Kanal, Name senden
09227  efee
09228  efee
09229  efee  ===> Abschluß mit ok-Flag <===
09230  efee
09231  efee  18        opnrts clc
09232  efef  60               rts
09233  eff0
09234  eff0
09235  eff0  ===> I/O-Fehler #1: too many files <===
09236  eff0
09237  eff0  4c 7c f6   err1  jmp error1       I/O-Fehler #1: too many Files
09238  eff3
09239  eff3
09240  eff3  ===> I/O-Fehler #2: file open <===
09241  eff3
09242  eff3  4c 7f f6   err2  jmp error2       I/O-Fehler #2: File Open
09243  eff6
09244  eff6
09245  eff6  ===> logisches Kassetten-File öffnen <===
09246  eff6
09247  eff6  20 80 e9   opncas jsr tapadr      Zeiger akt. Kass.puffer einrichten
09248  eff9  b0 03            bcs op155
09249  effb
09250  effb  4c 94 f6         jmp error9      I/O-Fehler #9: illegal device number
09251  effe
09252  effe
09253  effe  a5 b9      op155 lda sa          akt. Sekundäradresse
09254  f000  29 0f            and #%00001111  0 = lesen
09255  f002  d0 1f            bne op200       Open Kassette zum Schreiben
09256  f004
09257  f004  ===> Open Kassette zum Lesen <===
09258  f004
09259  f004  20 c8 e9         jsr cste1       auf Taste warten, Ausgabe 'press
                                               play on tape'
09260  f007  b0 36            bcs op180       Abbruch durch Stopp-Taste
09261  f009
09262  f009  20 0f f5         jsr luking      Direktmodus Ausgabe 'searching (for
                                               fn)'
09263  f00c
09264  f00c  a5 b7            lda fnlen       Länge akt. Filename
09265  f00e  f0 0a            beq op170       nächstbestes File Kass. öffnen
09266  f010
09267  f010  20 9a e9         jsr faf         File auf Kassette suchen/anzeigen
09268  f013  90 18            bcc op171       gefunden
09269  f015  f0 28            beq op180       Abbruch durch Stopp-Taste
```

```
09270  f017
09271  f017  ===> I/O-Fehler #4: file not found <===
09272  f017
09273  f017  4c 85 f6   op160  jmp error4       I/O-Fehler #4: File not found
09274  f01a
09275  f01a
09276  f01a
09277  f01a  ===> nächstbestes File Kass. öffnen <===
09278  f01a
09279  f01a  20 d0 e8   op170  jsr fah          Header-Block auf Kassette
                                               suchen/anzeigen
09280  f01d  90 0e             bcc op171       gefunden
09281  f01f  f0 1e             beq op180       Abbruch durch Stopp-Taste
09282  f021  b0 f4             bcs op160       nicht gefunden
09283  f023
09284  f023
09285  f023  ===> Open Kassette zum Schreiben <===
09286  f023
09287  f023  20 e9 e9   op200  jsr cste2        prüfen, Meldungen ausgeben, warten
                                               auf 'play & record'
09288  f026  b0 17             bcs op180       Abbruch durch Stopp-Taste
09289  f028
09290  f028  a9 04             lda #4          Headermarke 1. Block Datenfile
09291  f02a  20 19 e9          jsr tapeh       PRG-Header/Ende-Block auf Kassette
                                               schreiben
09292  f02d  a9 bf      op171  lda #191         Zeiger auf Ende Kass.puffer
09293  f02f
09294  f02f  a4 b9             ldy sa          akt. Sekundäradresse
09295  f031  c0 60             cpy #%01100000
09296  f033  f0 07             beq op172       PRG-File
09297  f035
09298  f035  a0 00             ldy #0
09299  f037  a9 02             lda #2          Headermarke Folgeblock Datenfile
09300  f039  91 b2             sta (tape1),y   in Kassetten-Puffer
09301  f03b
09302  f03b  98                tya
09303  f03c  85 a6      op172  sta bufpt        Zeiger in Kassettenpuffer
09304  f03e
09305  f03e  18                clc             ok
09306  f03f  60         op180  rts
09307  f040
09308  f040
09309  f040  ===> Open RS 232 <===
09310  f040
09311  f040  20 b0 f0   opn232 jsr cln232       CIA's RS 232 initialisieren
09312  f043  8c 14 0a          sty rsstat      RS Status-Byte rücksetzen
09313  f046
09314  f046  c4 b7      opn020 cpy fnlen        Länge akt. Filename
09315  f048  f0 0b             beq opn025      Parameter übernehmen
09316  f04a
09317  f04a  ===> RS 232 Parameter eintragen <===
09318  f04a
09319  f04a  20 ae f7          jsr fnadry      RS 232 - Parameter aus fnam lesen
09320  f04d  99 10 0a          sta m51ctr,y    und abspeichern
09321  f050
09322  f050  c8                iny
09323  f051  c0 04             cpy #4          max. 4 Bytes
09324  f053  d0 f1             bne opn020
```

```
09325  f055
09326  f055  20 8e e6   opn025 jsr bitcnt       Anzahl Datenbits festlegen
09327  f058  8e 15 0a          stx bitnum       RS 232 Anzahl Datenbits
09328  f05b
09329  f05b  ad 10 0a          lda m51ctr       RS 232 CTRL-Register
09330  f05e  29 0f             and #%00001111
09331  f060
09332  f060   ....3210  Nummer der Baud-Rate Konstanten
09333  f060
09334  f060  f0 1c             beq opn028       User-Baud-Rate
09335  f062
09336  f062  ===> Baud-Rate Konstante aus Tabelle entnehmen <===
09337  f062
09338  f062  0a                asl a
09339  f063  aa                tax              Zeiger in Tabelle
09340  f064
09341  f064  ad 03 0a          lda palnts       Flag PAL/NTSC
09342  f067  d0 09             bne opn026       Konstanten PAL-Tabelle
09343  f069
09344  f069  ===> Baud-Rate Konstante aus NTSC-Tabelle <===
09345  f069
09346  f069  bc 4f e8          ldy baudo-1,x    Tab. RS 232 Baud-Rate Konst. NTSC
09347  f06c  bd 4e e8          lda baudo-2,x    Tab. RS 232 Baud-Rate Konst. NTSC
09348  f06f  4c 78 f0          jmp opn027
09349  f072
09350  f072
09351  f072  ===> Baud Rate Konst. aus PAL-Tab. <===
09352  f072
09353  f072  bc 63 e8   opn026 ldy baudop-1,x   Tab. RS 232 Baud-Rate Konst. PAL
09354  f075  bd 62 e8          lda baudop-2,x   Tab. RS 232 Baud-Rate Konst. PAL
09355  f078
09356  f078  8c 13 0a   opn027 sty m51ajb+1     RS 232 BAUD-Rate Konst. hi
09357  f07b  8d 12 0a          sta m51ajb       RS 232 BAUD-Rate Konst. lo
09358  f07e
09359  f07e
09360  f07e  ===> Timer Konst. berechnen <===
09361  f07e
09362  f07e  ad 12 0a   opn028 lda m51ajb       RS 232 BAUD-Rate Konst. lo
09363  f081  0a                asl a            mal zwei
09364  f082  aa                tax
09365  f083
09366  f083  ad 13 0a          lda m51ajb+1     RS 232 BAUD-Rate Konst. hi
09367  f086  2a                rol a            mal zwei
09368  f087  a8                tay
09369  f088
09370  f088  8a                txa
09371  f089  69 c8             adc #200         plus 200
09372  f08b  8d 16 0a          sta baudof       RS 232 Timer-Konst. lo BAUD-Rate
09373  f08e
09374  f08e  98                tya
09375  f08f  69 00             adc #$00
09376  f091  8d 17 0a          sta baudof+1     RS 232 Timer-Konst. hi BAUD-Rate
09377  f094
09378  f094  ad 11 0a          lda m51cdr       RS 232 Command-Register
09379  f097
09380  f097   xxxxxxx0  3-Draht Handshake
09381  f097   xxxxxxx1  X-Draht Handshake
09382  f097
```

```
09383  f097  4a               lsr a
09384  f098  90 09            bcc opn050       3-Draht Handshake
09385  f09a
09386  f09a  ad 01 dd         lda d2prb        CIA2 Datenport B
09387  f09d
09388  f09d   7.......  DSR DATA SET READY
09389  f09d
09390  f09d  0a               asl a
09391  f09e  b0 03            bcs opn050       DSR gesetzt
09392  f0a0
09393  f0a0  20 55 e7         jsr ckdsrx       DSR fehlt, Status setzen
09394  f0a3
09395  f0a3
09396  f0a3  ===> Zeiger in RS 232 Puffer einrichten <===
09397  f0a3
09398  f0a3  ad 18 0a   opn050 lda ridbe        RS 232 Zeiger auf eingeles. Byte
09399  f0a6  8d 19 0a         sta ridbs        RS 232 Zeiger auf auszules. Byte
09400  f0a9
09401  f0a9  ad 1b 0a         lda rodbe        RS 232 Zeiger auf auszugeb. Byte
09402  f0ac  8d 1a 0a         sta rodbs        RS 232 Zeiger auf einzules. Byte
09403  f0af
09404  f0af  60               rts
09405  f0b0
09406  f0b0
09407  f0b0  ===> CIA's RS 232 initialisieren <===
09408  f0b0
09409  f0b0  a9 7f      cln232 lda #%01111111
09410  f0b2
09411  f0b2   0xxxxxxx  CLEAR IRQ ENABLE Bit
09412  f0b2   x11xxxxx  unbenutzt
09413  f0b2   xxx1xxxx  FLAG IRQ (RS in)
09414  f0b2   xxxx1xxx  SDR IRQ
09415  f0b2   xxxxx1xx  ALARM TOD IRQ
09416  f0b2   xxxxxx1x  TIMER B UNDERFLOW IRQ (RS Empfangen)
09417  f0b2   xxxxxxx1  TIMER A UNDERFLOW IRQ (RS Senden)
09418  f0b2
09419  f0b2  8d 0d dd         sta d2icr        CIA2 Interrupt Control-Register
09420  f0b5
09421  f0b5  a9 06            lda #%00000110
09422  f0b7
09423  f0b7   0xxxxxxx  DSR DATA SET READY:      Input
09424  f0b7   x0xxxxxx  CTS CLEAR TO SEND:       Input
09425  f0b7   xx0xxxxx  DCD DATA CARRIER DETECT: Input
09426  f0b7   xxx0xxxx  unbenutzt:               Input
09427  f0b7   xxxx0xxx  RI  RING INDICATOR:      Input
09428  f0b7   xxxxx1xx  DTR DATA TERMINAL READY: Output
09429  f0b7   xxxxxx1x  RTS REQUEST TO SEND:     Output
09430  f0b7   xxxxxxx0  RXD RECEIVED DATA:       Input
09431  f0b7
09432  f0b7  8d 03 dd         sta d2ddrb       CIA2 Datenrichtungs-Reg. B
09433  f0ba
09434  f0ba   xxxxx1xx  DTR DATA TERMINAL READY: 1 (hi)
09435  f0ba   xxxxxx1x  RTS REQUEST TO SEND:     1 (hi)
09436  f0ba
09437  f0ba  8d 01 dd         sta d2prb        CIA2 Datenport B
09438  f0bd
09439  f0bd  a9 04            lda #%00000100
09440  f0bf
```

```
09441  f0bf   xxxxx1xx  TXD TRANSMIT DATA:       1 (hi)
09442  f0bf
09443  f0bf  0d 00 dd          ora d2pra        CIA2 Datenport A
09444  f0c2  8d 00 dd          sta d2pra        CIA2 Datenport A
09445  f0c5
09446  f0c5  a0 00             ldy #%00000000   alle RS 232 IRQ disabled
09447  f0c7  8c 0f 0a          sty enabl        CIA2 NMI-Flag
09448  f0ca
09449  f0ca  60                rts
09450  f0cb
09451  f0cb
09452  f0cb  ===> Open IEC-Kanal, Name senden <===
09453  f0cb
09454  f0cb  a5 b9       openi  lda sa           akt. Sekundäradresse
09455  f0cd  30 04              bmi opnier
09456  f0cf
09457  f0cf  a4 b7              ldy fnlen        Länge akt. Filename
09458  f0d1  d0 02              bne opniok       Ausgabe Filename / Befehl IEC Bus
09459  f0d3
09460  f0d3  18          opnier clc
09461  f0d4  60                 rts
09462  f0d5
09463  f0d5
09464  f0d5  ===> Ausgabe Filename / Befehl IEC Bus <===
09465  f0d5
09466  f0d5  a9 00       opniok lda #%00000000   Vorbesetzen
09467  f0d7  85 90              sta status       I/O-Status 'st'
09468  f0d9
09469  f0d9  a5 ba              lda fa           akt. Gerätenummer
09470  f0db  20 3e e3           jsr listn        Ausgabe LISTEN auf IEC-Bus
09471  f0de
09472  f0de  24 90              bit status       I/O-Status 'st'
09473  f0e0  30 0b              bmi opnir2       I/O-Fehler #5: device not present
09474  f0e2
09475  f0e2  a5 b9              lda sa           akt. Sekundäradresse
09476  f0e4  09 f0              ora #%11110000
09477  f0e6  20 d2 e4           jsr secnd        Ausgabe Sek.Adr. nach LISTEN auf
                                                 IEC-Bus
09478  f0e9
09479  f0e9  a5 90              lda status       I/O-Status 'st'
09480  f0eb  10 05              bpl op35         Test / Filename auf IEC ausgeben
09481  f0ed
09482  f0ed
09483  f0ed  ===> I/O-Fehler #5: device not present <===
09484  f0ed
09485  f0ed  68          opnir2 pla
09486  f0ee  68                 pla
09487  f0ef  4c 88 f6           jmp error5       I/O-Fehler #5: device not present
09488  f0f2
09489  f0f2
09490  f0f2  ===> Test / Filename auf IEC ausgeben <===
09491  f0f2
09492  f0f2  a5 b7       op35   lda fnlen        Länge akt. Filename
09493  f0f4  f0 0d              beq op45         kein Filename/Kommando
09494  f0f6
09495  f0f6  ===> Filename/Kommando auf IEC ausgeben <===
09496  f0f6
09497  f0f6  a0 00              ldy #0
```

```
09498  f0f8  20 ae f7   op40   jsr fnadry       LDA (FNADR),y
09499  f0fb  20 03 e5          jsr ciout        Ausgabe ac auf IEC-Bus
09500  f0fe  c8                iny
09501  f0ff  c4 b7             cpy fnlen        Länge akt. Filename
09502  f101  d0 f5             bne op40
09503  f103
09504  f103  4c b0 f5   op45   jmp cunlsn       Ausgabe UNLISTEN auf IEC-Bus (BASIC)
09505  f106
09506  f106
09507  f106
09508  f106  ===> Eingabekanal (la in xr) öffnen <===
09509  f106
09510  f106  20 02 f2   nchkin jsr lookup       log. Filenummer ac in Tab. suchen
09511  f109  d0 3e             bne err3         I/O-Fehler #3: File not Open
09512  f10b
09513  f10b  20 12 f2          jsr getlfs       la, fa, sa x-ind. aus Filetab.
09514  f10e  f0 13             beq ckirts       Eingabe von Tastatur
09515  f110
09516  f110  c9 03             cmp #3           Geräteadresse Bildschirm
09517  f112  f0 0f             beq ckirts       Eingabe von Bildschirm
09518  f114  b0 11             bcs ckiser       Eingabe von IEC-Bus
09519  f116
09520  f116  c9 02             cmp #2           Geräteadresse RS 232
09521  f118  d0 03             bne ckitap       Eingabe von Kassette
09522  f11a
09523  f11a  ===> Eingabekanal für RS 232 öffnen <===
09524  f11a
09525  f11a  4c 95 e7          jmp cki232       Eingabe von RS 232
09526  f11d
09527  f11d
09528  f11d  ===> Eingabekanal für Kassette öffnen <===
09529  f11d
09530  f11d  a6 b9      ckitap ldx sa           akt. Sekundäradresse
09531  f11f  e0 60             cpx #%01100000
09532  f121  d0 20             bne err6         I/O-Fehler #6: not input File
09533  f123
09534  f123
09535  f123  ===> ac als akt. Eingabe-Gerätenummer Speichern <===
09536  f123
09537  f123  85 99      ckirts sta dfltn        Vorgabe Eingabe-Gerätenummer (0)
09538  f125  18               clc
09539  f126  60               rts
09540  f127
09541  f127
09542  f127  ===> IEC-Eingabekanal öffnen <===
09543  f127
09544  f127  aa         ckiser tax              Gerätenummer
09545  f128  20 3b e3          jsr talk         Ausgabe TALK auf IEC-Bus
09546  f12b
09547  f12b  24 90             bit status       I/O-Status 'st'
09548  f12d  30 11             bmi err5         I/O-Fehler #5: device not present
09549  f12f
09550  f12f  a5 b9             lda sa           akt. Sekundäradresse
09551  f131  10 05             bpl jx340
09552  f133
09553  f133  20 e9 e4          jsr tkatn        NOT ATN HI nach Ausgabe TALK auf
                                                 IEC-Bus
09554  f136  10 03             bpl jx350
```

```
09555  f138
09556  f138  20 e0 e4   jx340  jsr tksa        Ausgabe Sek.Adr. nach TALK auf
                                                IEC-Bus
09557  f13b  8a         jx350  txa
09558  f13c
09559  f13c  24 90             bit status      I/O-Status 'st'
09560  f13e  10 e3             bpl ckirts      ac als akt. Eingabe-Gerätenummer
                                                Speichern
09561  f140
09562  f140
09563  f140  ===> I/O-Fehler #5: device not present <===
09564  f140
09565  f140  4c 88 f6   err5   jmp error5       I/O-Fehler #5: device not present
09566  f143
09567  f143  ===> I/O-Fehler #6: not input file <===
09568  f143
09569  f143  4c 8b f6   err6   jmp error6       I/O-Fehler #6: not input File
09570  f146
09571  f146  ===> I/O-Fehler #7: not output file <===
09572  f146
09573  f146  4c 8e f6   err7   jmp error7       I/O-Fehler #7: not output File
09574  f149
09575  f149  ===> I/O-Fehler #3: file not open <===
09576  f149
09577  f149  4c 82 f6   err3   jmp error3       I/O-Fehler #3: File not Open
09578  f14c
09579  f14c
09580  f14c  ===> Ausgabe-Kanal (la in xr) öffnen <===
09581  f14c
09582  f14c  20 02 f2   nckout jsr lookup       log. Filenummer ac in Tab. suchen
09583  f14f  d0 f8             bne err3        I/O-Fehler #3: File not Open
09584  f151
09585  f151  20 12 f2          jsr getlfs      la, fa, sa x-ind. aus Filetab.
09586  f154  f0 f0             beq err7        I/O-Fehler #7: not output File
09587  f156
09588  f156  c9 03             cmp #3          Geräteadresse Bildschirm
09589  f158  f0 0f             beq ckorts      Ausgabe auf Bildschirm
09590  f15a  b0 11             bcs ckoser      Ausgabe auf IEC-Bus
09591  f15c
09592  f15c  c9 02             cmp #2          Geräteadresse RS 232
09593  f15e  d0 03             bne ckotap      Ausgabe auf Kassette
09594  f160
09595  f160  4c 29 e7          jmp cko232      Ausgabe auf RS 232
09596  f163
09597  f163
09598  f163  ===> Kassette als Ausgabegerät setzen <===
09599  f163
09600  f163  a6 b9      ckotap ldx sa          akt. Sekundäradresse
09601  f165  e0 60             cpx #%01100000
09602  f167  f0 dd             beq err7        I/O-Fehler #7: not output File
09603  f169
09604  f169
09605  f169  ===> ac als akt. Ausgabe-Gerätenummer Speichern <===
09606  f169
09607  f169  85 9a      ckorts sta dflto       Vorgabe Ausgabe-Gerätenummer (3)
09608  f16b
09609  f16b  18         clrts  clc
09610  f16c  60               rts
```

```
09611  f16d
09612  f16d
09613  f16d  ===> IEC-Bus als Ausgabegerät setzen <===
09614  f16d
09615  f16d  aa          ckoser tax              Gerätenummer
09616  f16e  20 3e e3           jsr listn        Ausgabe LISTEN auf IEC-Bus
09617  f171
09618  f171  24 90              bit status       I/O-Status 'st'
09619  f173  30 cb              bmi err5         I/O-Fehler #5: device not present
09620  f175
09621  f175  a5 b9              lda sa           akt. Sekundäradresse
09622  f177  10 05              bpl ck50
09623  f179
09624  f179  20 d7 e4           jsr scatn        NOT ATN HI nach Ausgabe Sek.Adr. auf
                                                  IEC-Bus
09625  f17c  d0 03              bne ck60
09626  f17e
09627  f17e  20 d2 e4    ck50   jsr secnd        Ausgabe Sek.Adr. nach LISTEN auf
                                                  IEC-Bus
09628  f181  8a          ck60   txa
09629  f182
09630  f182  24 90              bit status       I/O-Status 'st'
09631  f184  10 e3              bpl ckorts       ac als akt. Ausgabe-Gerätenummer
                                                  Speichern
09632  f186  30 b8              bmi err5         I/O-Fehler #5: device not present
09633  f188
09634  f188
09635  f188                     .end
09636  f188                     .lib c128.kernal2
09637  f188
09639  f188
09640  f188
09641  f188  ===> log. File schließen <===
09642  f188
09643  f188  66 92       nclose ror svxt         Carry in Bit7 schieben
09644  f18a
09645  f18a  20 07 f2           jsr jltlk        log. Filenummer xr in Tab. suchen
09646  f18d  d0 dc              bne clrts        log. Filenummer nicht gefunden
09647  f18f
09648  f18f  20 12 f2           jsr getlfs       la, fa, sa x-ind. aus Filetab.
09649  f192
09650  f192  8a                 txa              Zeiger in Tabellen
09651  f193  48                 pha              auf Stapel
09652  f194
09653  f194  a5 ba              lda fa           akt. Gerätenummer
09654  f196  f0 4c              beq jx150        File aus Filetabellen entfernen
09655  f198
09656  f198  c9 03              cmp #3           Gerätenummer Bildschirm
09657  f19a  f0 48              beq jx150        Bildschirm aus Tab. löschen
09658  f19c  b0 31              bcs jx120        IEC-File aus Tab. löschen
09659  f19e
09660  f19e  c9 02              cmp #2           Gerätenummer RS 232
09661  f1a0  d0 07              bne jx115        Kassetten-File aus Tab. löschen
09662  f1a2
09663  f1a2  ===> RS 232 - File aus Tabellen entfernen <===
09664  f1a2
09665  f1a2  68                 pla              Zeiger in Tabellen vom Stapel
09666  f1a3  20 e5 f1           jsr jxrmv        File aus Tabellen entfernen
```

```
09667  f1a6  4c b0 f0          jmp cln232       CIA's RS 232 initialisieren
09668  f1a9
09669  f1a9
09670  f1a9  ===> File für Kassette schließen <===
09671  f1a9
09672  f1a9  a5 b9       jx115 lda sa           akt. Sekundäradresse
09673  f1ab  29 0f             and #%00001111
09674  f1ad  f0 35             beq jx150        Lesefile aus Tab. entfernen
09675  f1af
09676  f1af  ===> letzten Block auf Kassette schreiben <===
09677  f1af
09678  f1af  20 80 e9          jsr tapadr       Zeiger akt. Kass.puffer einrichten
09679  f1b2
09680  f1b2  a9 00             lda #0           Endezeichen
09681  f1b4  38                sec
09682  f1b5  20 8c ef          jsr casout
09683  f1b8  20 15 ea          jsr wblk         Kassettenpuffer auf Kassette
                                                schreiben
09684  f1bb  90 04             bcc jx117
09685  f1bd
09686  f1bd  68                pla              Zeiger verwerfen
09687  f1be  a9 00             lda #0
09688  f1c0  60                rts
09689  f1c1
09690  f1c1
09691  f1c1  a5 b9       jx117 lda sa           akt. Sekundäradresse
09692  f1c3  c9 62             cmp #%01100010   Sekundäradresse 2 ?
09693  f1c5  d0 1d             bne jx150        nein
09694  f1c7
09695  f1c7
09696  f1c7  ===> END-OF-TAPE Markierung auf Kassette ausgeben <===
09697  f1c7
09698  f1c7  a9 05             lda #5           Headermarke EOT-Block
09699  f1c9  20 19 e9          jsr tapeh        PRG-Header/Ende-Block auf Kassette
                                                schreiben
09700  f1cc  4c e4 f1          jmp jx150        File aus Filetabellen entfernen
09701  f1cf
09702  f1cf
09703  f1cf  ===> IEC File aus Tab. entfernen <===
09704  f1cf
09705  f1cf  24 92       jx120 bit svxt         Kassette Timing-Konstante
09706  f1d1  10 0e             bpl jx130        IEC-PRG-file schließen
09707  f1d3
09708  f1d3  a5 ba             lda fa           akt. Gerätenummer
09709  f1d5  c9 08             cmp #8
09710  f1d7  90 08             bcc jx130        IEC-PRG-file schließen
09711  f1d9
09712  f1d9  a5 b9             lda sa           akt. Sekundäradresse
09713  f1db  29 0f             and #15
09714  f1dd  c9 0f             cmp #15
09715  f1df  f0 03             beq jx150        File aus Filetabellen entfernen
09716  f1e1
09717  f1e1
09718  f1e1  ===> IEC-PRG-file schließen <===
09719  f1e1
09720  f1e1  20 9e f5    jx130 jsr clsei        IEC-Programmmfile schließen
09721  f1e4
09722  f1e4
```

```
09723  f1e4  ===> File aus Filetabellen entfernen <===
09724  f1e4
09725  f1e4  68         jx150  pla             Zeiger in Tabellen vom Stapel
09726  f1e5  aa         jxrmv  tax
09727  f1e6  c6 98             dec ldtnd       Anzahl offene Files
09728  f1e8  e4 98             cpx ldtnd       Anzahl offene Files
09729  f1ea  f0 14             beq jx170       letztes File, Verschieben unnötig
09730  f1ec
09731  f1ec  ===> Verschiebeschleife File-Tabellen <===
09732  f1ec
09733  f1ec  a4 98             ldy ldtnd       Anzahl offene Files
09734  f1ee  b9 62 03          lda lat,y       Tab. der log. Filenummern
09735  f1f1  9d 62 03          sta lat,x       Tab. der log. Filenummern
09736  f1f4
09737  f1f4  b9 6c 03          lda fat,y       Tab. der Gerätenummern
09738  f1f7  9d 6c 03          sta fat,x       Tab. der Gerätenummern
09739  f1fa
09740  f1fa  b9 76 03          lda sat,y       Tab. der Sekundär-Adr.
09741  f1fd  9d 76 03          sta sat,x       Tab. der Sekundär-Adr.
09742  f200
09743  f200  18         jx170  clc
09744  f201  60               rts
09745  f202
09746  f202
09747  f202  ===> log. Filenummer ac in Tab. suchen <===
09748  f202
09749  f202  a9 00      lookup lda #%00000000
09750  f204  85 90             sta status      I/O-Status 'st'
09751  f206
09752  f206  8a               txa
09753  f207
09754  f207
09755  f207  ===> log. Filenummer xr in Tab. suchen <===
09756  f207
09757  f207  a6 98      jltlk  ldx ldtnd       Anzahl offene Files
09758  f209  ca         jx600  dex             Zeiger in Tab. erniedrigen
09759  f20a  30 05             bmi jx601       fertig
09760  f20c
09761  f20c  dd 62 03          cmp lat,x       Tab. der log. Filenummern
09762  f20f  d0 f8             bne jx600
09763  f211
09764  f211  60         jx601  rts
09765  f212
09766  f212
09767  f212  ===> la, fa, sa x-ind. aus Filetab. <===
09768  f212
09769  f212  bd 62 03   getlfs lda lat,x       Tab. der log. Filenummern
09770  f215  85 b8             sta la          akt. log. Filenummer
09771  f217
09772  f217  bd 76 03          lda sat,x       Tab. der Sekundär-Adr.
09773  f21a  85 b9             sta sa          akt. Sekundäradresse
09774  f21c
09775  f21c  bd 6c 03          lda fat,x       Tab. der Gerätenummern
09776  f21f  85 ba             sta fa          akt. Gerätenummer
09777  f221
09778  f221  60               rts
09779  f222
09780  f222
```

```
09781  f222  ===> offene Files = 0, Ersatz-I/O-Kanäle <===
09782  f222
09783  f222  a9 00      nclall lda #0          Anzahl offene Files
09784  f224  85 98             sta ldtnd       rücksetzen (Files sind noch offen)
09785  f226
09786  f226
09787  f226  ===> Ein-/Ausgabe-Kanal schließen <===
09788  f226
09789  f226  a2 03      nclrch ldx #3
09790  f228  e4 9a             cpx dflto       Vorgabe Ausgabe-Gerätenummer (3)
09791  f22a  b0 03             bcs jx750       UNTALK, wenn akt.
                                                Eingabegerätenummer > 3
09792  f22c
09793  f22c  20 26 e5          jsr nunlsn      Ausgabe UNLISTEN auf IEC-Bus
09794  f22f
09795  f22f
09796  f22f  ===> UNTALK, wenn akt. Eingabegerätenummer > 3 <===
09797  f22f
09798  f22f  e4 99      jx750  cpx dfltn       Vorgabe Eingabe-Gerätenummer (0)
09799  f231  b0 03             bcs clall2
09800  f233
09801  f233  20 15 e5          jsr untlk       Ausgabe UNTALK auf IEC-Bus
09802  f236
09803  f236  86 9a      clall2 stx dflto       Vorgabe Ausgabe-Gerätenummer (3)
09804  f238
09805  f238  a9 00             lda #0          Gerätenummer Tastatur
09806  f23a  85 99             sta dfltn       Vorgabe Eingabe-Gerätenummer (0)
09807  f23c
09808  f23c  60                rts
09809  f23d
09810  f23d
09811  f23d
09812  f23d  ===> alle Files von Gerät in ac schließen <===
09813  f23d
09814  f23d  85 ba      closal sta fa          akt. Gerätenummer
09815  f23f
09816  f23f  c5 9a             cmp dflto       Vorgabe Ausgabe-Gerätenummer (3)
09817  f241  d0 05             bne closa1
09818  f243
09819  f243  a9 03             lda #3          Gerätenummer Bildschirm
09820  f245  85 9a             sta dflto       Vorgabe Ausgabe-Gerätenummer (3)
09821  f247
09822  f247  2c                .byte skip2
09823  f248  c5 99      closa1 cmp dfltn       Vorgabe Eingabe-Gerätenummer (0)
09824  f24a  d0 04             bne closa2
09825  f24c
09826  f24c  a9 00             lda #0          Gerätenummer Tastatur
09827  f24e  85 99             sta dfltn       Vorgabe Eingabe-Gerätenummer (0)
09828  f250
09829  f250  a5 ba      closa2 lda fa          akt. Gerätenummer
09830  f252  a6 98             ldx ldtnd       Anzahl offene Files
09831  f254  ca        closa3 dex
09832  f255  30 0d             bmi closax
09833  f257
09834  f257  dd 6c 03          cmp fat,x       Tab. der Gerätenummern
09835  f25a  d0 f8             bne closa3
09836  f25c
09837  f25c  bd 62 03          lda lat,x       Tab. der log. Filenummern
```

```
09838  f25f  20 c3 ff          jsr kclose       log. File üb. Vector schließen
                                                (Sprungv.)
09839  f262  90 ec             bcc closa2
09840  f264
09841  f264  60          closax rts
09842  f265
09843  f265
09844  f265  ===> load/verify (ac) ab Adresse in xr/yr <===
09845  f265
09846  f265  86 c3       loadsp stx tmp2          addrl Ladeadresse
09847  f267  84 c4              sty tmp2+1        addrh Ladeadresse
09848  f269
09849  f269  6c 30 03    load   jmp (iload)       Einlesen vom log. File ($f26c)
09850  f26c
09851  f26c
09852  f26c  ===> Lesen vom logischen File <===
09853  f26c
09854  f26c  85 93       nload  sta verck         Flag für load/verify
09855  f26e
09856  f26e  a9 00              lda #%00000000
09857  f270  85 90              sta status       I/O-Status 'st'
09858  f272
09859  f272  a5 ba              lda fa           akt. Gerätenummer
09860  f274  c9 04              cmp #4
09861  f276  b0 03              bcs lddisk       Lesen von IEC-Bus
09862  f278
09863  f278  4c 26 f3           jmp ldtape       load/verify auf Kassette
09864  f27b
09865  f27b
09866  f27b
09867  f27b  ===> Lesen von IEC-Bus <===
09868  f27b
09869  f27b  ad 1c 0a    lddisk lda serial        Flag für Fast-Serial-Device
09870  f27e  29 be              and #%10111110   FSD-Flag rücksetzen
09871  f280  8d 1c 0a           sta serial       Flag für Fast-Serial-Device
09872  f283
09873  f283  a6 b9              ldx sa           akt. Sekundäradresse
09874  f285  86 9e              stx ptr1         akt. sa
09875  f287
09876  f287  a4 b7              ldy fnlen        Länge akt. Filename
09877  f289  d0 03              bne ld26
09878  f28b
09879  f28b  4c 1a f3           jmp err8         I/O-Fehler #8: missing file name
09880  f28e
09881  f28e
09882  f28e  84 9f       ld26   sty ptr2         fnlen
09883  f290
09884  f290  20 0f f5           jsr luking       Direktmodus Ausgabe 'searching (for
                                                fn)'
09885  f293  20 a1 f3           jsr burst        Test / load - verify vom FSD
09886  f296  b0 03              bcs ld27         FSD nicht vorhanden
09887  f298
09888  f298  4c 9b f3           jmp lddone       Lade-Endadresse xr/yr, ok
09889  f29b
09890  f29b
09891  f29b  ===> load / verify über ser. IEC-Bus <===
09892  f29b
09893  f29b  a4 9f       ld27   ldy ptr2         fnlen
```

```
09894  f29d  84 b7            sty fnlen       Länge akt. Filename
09895  f29f
09896  f29f  a9 60            lda #%01100000
09897  f2a1  85 b9            sta sa          akt. Sekundäradresse
09898  f2a3
09899  f2a3  ===> TALK mit Sekundäradresse <===
09900  f2a3
09901  f2a3  20 cb f0         jsr openi       Open IEC-Kanal, Name senden
09902  f2a6  a5 ba            lda fa          akt. Gerätenummer
09903  f2a8  20 3b e3         jsr talk        Ausgabe TALK auf IEC-Bus
09904  f2ab  a5 b9            lda sa          akt. Sekundäradresse
09905  f2ad  20 e0 e4         jsr tksa        Ausgabe Sek.Adr. nach TALK auf
                                              IEC-Bus
09906  f2b0
09907  f2b0  ===> Startadresse einlesen <===
09908  f2b0
09909  f2b0  20 3e e4         jsr acptr       Zeichen vom IEC-Bus nach ac
09910  f2b3  85 ae            sta eal         addrl Ende load/save
09911  f2b5
09912  f2b5  20 3e e4         jsr acptr       Zeichen vom IEC-Bus nach ac
09913  f2b8  85 af            sta eah         addrh Ende load/save
09914  f2ba
09915  f2ba  ===> Test auf Timeout <===
09916  f2ba
09917  f2ba  a5 90            lda status      I/O-Status 'st'
09918  f2bc  4a               lsr a           Timeout-Bit
09919  f2bd  4a               lsr a           ins Carry
09920  f2be  b0 57            bcs err4        I/O-Fehler #4: file not found
09921  f2c0
09922  f2c0  a5 9e            lda ptr1        akt. sa
09923  f2c2  d0 08            bne ld30        nicht = 0
09924  f2c4
09925  f2c4  ===> gewünschte Ladeadresse abspeichern <===
09926  f2c4
09927  f2c4  a5 c3            lda tmp2
09928  f2c6  85 ae            sta eal         addrl Ende load/save
09929  f2c8
09930  f2c8  a5 c4            lda tmp2+1
09931  f2ca  85 af            sta eah         addrh Ende load/save
09932  f2cc
09933  f2cc  20 33 f5   ld30  jsr loding      Direktmodus Ausgabe 'loading ...'
09934  f2cf
09935  f2cf
09936  f2cf  ===> Warteschleife bei IEC-Timeout <===
09937  f2cf
09938  f2cf  a9 fd      ld40  lda #%11111101  Timeout-Bit
09939  f2d1  25 90            and status      rücksetzen
09940  f2d3  85 90            sta status      I/O-Status 'st'
09941  f2d5
09942  f2d5  20 e1 ff         jsr kstop       Stopp-Taste üb. Vector abfragen
                                              (Sprungv.)
09943  f2d8  f0 49            beq brkerr      Abbruch durch Stopp-Taste
09944  f2da
09945  f2da  ===> Zeichen lesen, Test auf Timeout <===
09946  f2da
09947  f2da  20 3e e4         jsr acptr       Zeichen vom IEC-Bus nach ac
09948  f2dd  aa               tax
09949  f2de  a5 90            lda status      I/O-Status 'st'
```

```
09950  f2e0  4a                lsr a
09951  f2e1  4a                lsr a
09952  f2e2  b0 eb             bcs ld40          Warteschleife bei IEC-Timeout
09953  f2e4
09954  f2e4  8a                txa               akt. Zeichen
09955  f2e5  a4 93             ldy verck         Flag für load/verify
09956  f2e7  f0 12             beq ld50          load
09957  f2e9
09958  f2e9  ===> Verify <===
09959  f2e9
09960  f2e9  85 bd             sta ochar         akt. Zeichen
09961  f2eb
09962  f2eb  a0 00             ldy #0
09963  f2ed  20 c9 f7          jsr eally         Byte indirekt über 'eal' lesen
09964  f2f0  c5 bd             cmp ochar         akt. Zeichen
09965  f2f2  f0 0a             beq ld60          Verify ok
09966  f2f4
09967  f2f4  ===> Verify Fehler <===
09968  f2f4
09969  f2f4  a9 10             lda #%00010000    Verify-Fehler Bit setzen
09970  f2f6  20 57 f7          jsr udst          ac-Bits in Status setzen
09971  f2f9  d0 03             bne ld60          immer
09972  f2fb
09973  f2fb
09974  f2fb  ===> Load <===
09975  f2fb
09976  f2fb  20 bf f7   ld50   jsr ealsy         Byte indirekt über 'eal' speichern
09977  f2fe
09978  f2fe
09979  f2fe
09980  f2fe  ===> Ladezeiger erhöhen <===
09981  f2fe
09982  f2fe  e6 ae      ld60   inc eal           addrl Ende load/save
09983  f300  d0 08             bne ld64
09984  f302
09985  f302  e6 af             inc eah           addrh Ende load/save
09986  f304  a5 af             lda eah           addrh Ende load/save
09987  f306
09988  f306  ===> Test auf Überschreiben MMU-Register <===
09989  f306
09990  f306  c9 ff             cmp #$ff
09991  f308  f0 16             beq err16         I/O-Fehler #0: out of memory
09992  f30a
09993  f30a  24 90      ld64   bit status        I/O-Status 'st'
09994  f30c
09995  f30c   x1xxxxxx  EOF
09996  f30c
09997  f30c  50 c1             bvc ld40          kein EOF
09998  f30e
09999  f30e  ===> END-OF-FILE <===
10000  f30e
10001  f30e  20 15 e5          jsr untlk         Ausgabe UNTALK auf IEC-Bus
10002  f311  20 9e f5          jsr clsei         IEC-Programmmfile schließen
10003  f314  4c 9b f3          jmp lddone        Lade-Endadresse xr/yr, ok
10004  f317
10005  f317
10006  f317  ===> I/O-Fehler #4: file not found <===
10007  f317
```

```
10008  f317  4c 85 f6    err4   jmp error4        I/O-Fehler #4: file not found
10009  f31a
10010  f31a  ===> I/O-Fehler #8: missing file name <===
10011  f31a
10012  f31a  4c 91 f6    err8   jmp error8        I/O-Fehler #8: missing file name
10013  f31d
10014  f31d  ===> I/O-Fehler #9: illegal device number <===
10015  f31d
10016  f31d  4c 94 f6    err9   jmp error9        I/O-Fehler #9: illegal device number
10017  f320
10018  f320  ===> I/O-Fehler #0: out of memory <===
10019  f320
10020  f320  4c 97 f6    err16  jmp errora        I/O-Fehler #0: out of memory
10021  f323
10022  f323  ===> Stopp-Taste gedrückt, IEC-File schließen <===
10023  f323
10024  f323  4c b5 f5    brkerr jmp break         Abbruch durch Stopp-Taste, clsei
10025  f326
10026  f326
10027  f326  ===> load/verify auf Kassette <===
10028  f326
10029  f326  c9 01       ldtape cmp #1            Gerätenummer Kassette
10030  f328  d0 f3              bne err9          I/O-Fehler #9: illegal device number
10031  f32a
10032  f32a  20 80 e9           jsr tapadr        Zeiger akt. Kass.puffer einrichten
10033  f32d  90 ee              bcc err9          I/O-Fehler #9: illegal device number
10034  f32f
10035  f32f  20 c8 e9           jsr cste1         auf Taste warten, Ausgabe 'press
                                                  play on tape'
10036  f332  b0 6c              bcs ldrts         Abbruch durch Stopp-Taste
10037  f334
10038  f334  20 0f f5           jsr luking        Direktmodus Ausgabe 'searching (for
                                                  fn)'
10039  f337
10040  f337
10041  f337  ===> weiter suchen <===
10042  f337
10043  f337  a5 b7       ld112  lda fnlen         Länge akt. Filename
10044  f339  f0 09              beq ld150         nächstbestes File von Kassette laden
10045  f33b
10046  f33b  20 9a e9           jsr faf           File auf Kassette suchen/anzeigen
10047  f33e  90 0b              bcc ld170         gefunden
10048  f340  f0 5e              beq ldrts         Abbruch durch Stopp-Taste
10049  f342  b0 d3              bcs err4          nicht gefunden
10050  f344
10051  f344
10052  f344  ===> nächstbestes File von Kassette laden <===
10053  f344
10054  f344  20 d0 e8    ld150  jsr fah           Header-Block auf Kassette
                                                  suchen/anzeigen
10055  f347  f0 57              beq ldrts         Abbruch durch Stopp-Taste
10056  f349  b0 cc              bcs err4          nicht gefunden
10057  f34b
10058  f34b  38          ld170  sec
10059  f34c
10060  f34c  a5 90              lda status        I/O-Status 'st'
10061  f34e  29 10              and #%00010000    Test auf Verify-Fehler
10062  f350  d0 4e              bne ldrts         Verify-Fehler
```

```
10063  f352
10064  f352  e0 01           cpx #1          Headermarke BASIC-Programm
10065  f354  f0 11           beq ld178       BASIC-Programm
10066  f356
10067  f356  e0 03           cpx #3          Headermarke Maschinen-Programm
10068  f358  d0 dd           bne ld112       weiter suchen
10069  f35a
10070  f35a
10071  f35a  ===> Maschinen-Programm laden <===
10072  f35a
10073  f35a  a0 01     ld177 ldy #1
10074  f35c  b1 b2           lda (tape1),y   addrl orig. Ladeadresse
10075  f35e  85 c3           sta tmp2
10076  f360
10077  f360  c8              iny
10078  f361  b1 b2           lda (tape1),y   addrh orig. Ladeadresse
10079  f363  85 c4           sta tmp2+1
10080  f365  b0 04           bcs ld179       immer
10081  f367
10082  f367
10083  f367  ===> BASIC-Programm laden <===
10084  f367
10085  f367  a5 b9     ld178 lda sa          akt. Sekundäradresse
10086  f369  d0 ef           bne ld177       Maschinen-Programm laden
10087  f36b
10088  f36b
10089  f36b  ===> Test auf out of memory <===
10090  f36b
10091  f36b  a0 03     ld179 ldy #3
10092  f36d  b1 b2           lda (tape1),y   addrl orig. Endadresse
10093  f36f
10094  f36f  a0 01           ldy #1
10095  f371  f1 b2           sbc (tape1),y   addrl orig. Anfangsadresse
10096  f373  aa              tax             addrl Differenz
10097  f374
10098  f374  a0 04           ldy #4
10099  f376  b1 b2           lda (tape1),y   addrh orig. Endadresse
10100  f378
10101  f378  a0 02           ldy #2
10102  f37a  f1 b2           sbc (tape1),y   addrh orig. Anfangsadresse
10103  f37c  a8              tay             addrh Differenz
10104  f37d
10105  f37d  18              clc
10106  f37e  8a              txa             addrl Differenz
10107  f37f  65 c3           adc tmp2        addrl gewünschte Ladeadresse
10108  f381  85 ae           sta eal         addrl berechnete Endadresse
10109  f383
10110  f383  98              tya             addrh Differenz
10111  f384  65 c4           adc tmp2+1      addrh gewünschte Ladeadresse
10112  f386  85 af           sta eah         addrh berechnete Endadresse
10113  f388
10114  f388  c9 ff           cmp #>mmucr     MMU 8722 Config. Reg. (Kopie)
10115  f38a  f0 94           beq err16       I/O-Fehler #0: out of memory
10116  f38c
10117  f38c  ===> gewünschte Ladeadresse bereitstellen <===
10118  f38c
10119  f38c  a5 c3           lda tmp2
10120  f38e  85 c1           sta stal        addrl Startadresse save
```

```
10121  f390
10122  f390  a5 c4            lda tmp2+1
10123  f392  85 c2            sta stah          addrh Startadresse save
10124  f394
10125  f394  20 33 f5         jsr loding        Direktmodus Ausgabe 'loading ...'
10126  f397  20 fb e9         jsr trd           Lesen von Kassette
10127  f39a  24               .byte skip1
10128  f39b
10129  f39b
10130  f39b  ===> Lade-Endadresse xr/yr, ok <===
10131  f39b
10132  f39b  18         lddone clc
10133  f39c  a6 ae             ldx eal           addrl Ende load/save
10134  f39e  a4 af             ldy eah           addrh Ende load/save
10135  f3a0  60         ldrts  rts
10136  f3a1
10137  f3a1
10138  f3a1  ===> Test / load - verify vom FSD <===
10139  f3a1
10140  f3a1  a0 00      burst  ldy #0
10141  f3a3  20 ae f7          jsr fnadry        LDA (FNADR),y
10142  f3a6
10143  f3a6  ===> Abbruch bei Filename auf '$' <===
10144  f3a6
10145  f3a6  c9 24             cmp #'$'
10146  f3a8  f0 f6             beq ldrts         Abbruch mit carry =1
10147  f3aa
10148  f3aa  ===> Kommandokanal Floppy öffnen (open 0, fa, 15) <===
10149  f3aa
10150  f3aa  a6 ba             ldx fa            akt. Gerätenummer
10151  f3ac  a0 0f             ldy #15           akt. Sekundäradresse
10152  f3ae  a9 00             lda #0            akt. log. Filenummer
10153  f3b0  20 38 f7          jsr setlfs        la, fa, sa aus ac, xr, yr setzen
10154  f3b3  85 b7             sta fnlen         Länge akt. Filename
10155  f3b5
10156  f3b5  20 c0 ff          jsr kopen         log. File üb. Vector öffnen, Ausgabe
                                                 fn (sprungv.)
10157  f3b8
10158  f3b8  a6 b8             ldx la            akt. log. Filenummer
10159  f3ba  20 c9 ff          jsr kckout        Ausgabe-Kanal üb. Vector öffnen
                                                 (Sprungv.)
10160  f3bd  90 08             bcc burst1        ok
10161  f3bf
10162  f3bf  ===> Abbruch durch device not present <===
10163  f3bf
10164  f3bf  20 8c f4          jsr brtend        akt. File schließen
10165  f3c2  68                pla               Rücksprungadresse
10166  f3c3  68                pla               verwerfen
10167  f3c4  4c 88 f6          jmp error5        I/O-Fehler #5: device not present
10168  f3c7
10169  f3c7
10170  f3c7  ===> Aufruf FSD senden <===
10171  f3c7
10172  f3c7  a0 03      burst1 ldy #3
10173  f3c9  b9 0b f5   burst2 lda brtcmd-1,y   Floppy-Befehl für
                                                 Fast-Serial-Interface
10174  f3cc  20 d2 ff          jsr kbsout        Ausgabe ac üb. Vector auf akt. Kanal
                                                 (Sprungv.)
```

```
10175  f3cf  88              dey
10176  f3d0  d0 f7           bne burst2
10177  f3d2
10178  f3d2
10179  f3d2  ===> akt. Filenamen ausgeben <===
10180  f3d2
10181  f3d2  20 ae f7   btfnam jsr fnadry     LDA (FNADR),y
10182  f3d5  20 d2 ff         jsr kbsout     Ausgabe ac üb. Vector auf akt. Kanal
                                             (Sprungv.)
10183  f3d8  c8              iny
10184  f3d9  c4 9f           cpy ptr2
10185  f3db  d0 f5           bne btfnam     akt. Filenamen ausgeben
10186  f3dd
10187  f3dd  20 cc ff         jsr kclrch     Ein-/Ausgabe-Kanäle üb. Vector
                                             schließen (Sprungv.)
10188  f3e0
10189  f3e0  2c 1c 0a         bit serial     Flag für Fast-Serial-Device
10190  f3e3
10191  f3e3   x1xxxxxx  FSD vorhanden
10192  f3e3
10193  f3e3  70 05            bvs btopen     akt. File öffnen
10194  f3e5
10195  f3e5  ===> Abbruch durch FSD nicht vorhanden <===
10196  f3e5
10197  f3e5  20 8c f4         jsr brtend     akt. File schließen
10198  f3e8  38               sec            Fehler-Flag
10199  f3e9  60               rts
10200  f3ea
10201  f3ea
10202  f3ea  ===> akt. File öffnen <===
10203  f3ea
10204  f3ea  a5 9f      btopen lda ptr2
10205  f3ec  85 b7            sta fnlen      Länge akt. Filename
10206  f3ee
10207  f3ee  78               sei
10208  f3ef  20 45 e5         jsr clkhi      CLOCK-Leitung hi
10209  f3f2  20 c3 e5         jsr spinp      Serial Port INPUT
10210  f3f5
10211  f3f5  2c 0d dc         bit d1icr      CIA1 Interrupt Control-Register
10212  f3f8  20 03 f5         jsr wiggle     CLOCK OUT flippen
10213  f3fb  20 ba f4         jsr brtbyt     Zeichen vom FSD lesen
10214  f3fe
10215  f3fe  c9 02            cmp #2
10216  f400  d0 08            bne btfnd      File gefunden
10217  f402
10218  f402  ===> Abbruch durch file not found <===
10219  f402
10220  f402  20 8c f4         jsr brtend     akt. File schließen
10221  f405  68               pla            Rücksprungadresse
10222  f406  68               pla            verwerfen
10223  f407
10224  f407  4c 85 f6         jmp error4     I/O-Fehler #4: File not found
10225  f40a
10226  f40a
10227  f40a  ===> File gefunden <===
10228  f40a
10229  f40a  48         btfnd  pha            Transfer-Status
10230  f40b
```

```
10231  f40b  c9 1f           cmp #31          Status: letzter Block folgt
10232  f40d  d0 0b           bne btckst       Test Transfer-Status
10233  f40f
10234  f40f  ===> Länge letzter Block einlesen <===
10235  f40f
10236  f40f  20 03 f5        jsr wiggle       CLOCK OUT flippen
10237  f412  20 ba f4        jsr brtbyt       Zeichen vom FSD lesen
10238  f415  85 a5           sta cntdn        Länge letzter Block
10239  f417
10240  f417  4c 21 f4        jmp btldad       Ladeadresse von File einlesen
10241  f41a
10242  f41a
10243  f41a  ===> Test Transfer-Status <===
10244  f41a
10245  f41a  c9 02     btckst cmp #2           Transfer-Status
10246  f41c  90 03           bcc btldad       1 = ok
10247  f41e
10248  f41e  ===> Abbruch durch Ladefehler <===
10249  f41e
10250  f41e  68              pla              Status verwerfen
10251  f41f  b0 77           bcs btlode       Ladefehler
10252  f421
10253  f421
10254  f421  ===> Ladeadresse von File einlesen <===
10255  f421
10256  f421  20 33 f5   btldad jsr loding     Direktmodus Ausgabe 'loading ...'
10257  f424
10258  f424  ===> orig. Ladeadresse einlesen <===
10259  f424
10260  f424  20 03 f5        jsr wiggle       CLOCK OUT flippen
10261  f427  20 ba f4        jsr brtbyt       Zeichen vom FSD lesen
10262  f42a  85 ae           sta eal          addrl Ende load/save
10263  f42c
10264  f42c  20 03 f5        jsr wiggle       CLOCK OUT flippen
10265  f42f  20 ba f4        jsr brtbyt       Zeichen vom FSD lesen
10266  f432  85 af           sta eah          addrh Ende load/save
10267  f434
10268  f434  a6 9e           ldx ptr1         akt. Sekundäradresse
10269  f436  d0 08           bne btldab       absolut laden
10270  f438
10271  f438  ===> Programm über tmp2 laden <===
10272  f438
10273  f438  a5 c3           lda tmp2
10274  f43a  a6 c4           ldx tmp2+1
10275  f43c  85 ae           sta eal          addrl Ende load/save
10276  f43e  86 af           stx eah          addrh Ende load/save
10277  f440
10278  f440
10279  f440  ===> Programm absolut laden <===
10280  f440
10281  f440  a5 ae     btldab lda eal          addrl Ende load/save
10282  f442  a6 af           ldx eah          addrh Ende load/save
10283  f444  85 ac           sta sal          addrl für load/save
10284  f446  86 ad           stx sah          addrh für load/save
10285  f448
10286  f448  68              pla              Transfer-Status
10287  f449  c9 1f           cmp #31          Status letzter Block folgt
10288  f44b  f0 32           beq brteoi       letzten Datenblock lesen, EOI-Flag
```

```
10289  f44d
10290  f44d  20 03 f5          jsr wiggle       CLOCK OUT flippen
10291  f450
10292  f450  ===> ersten Datenblock lesen <===
10293  f450
10294  f450  a9 fc             lda #252         erster Datenblock 252 Bytes
10295  f452  85 a5             sta cntdn        Länge akt. Datenblock
10296  f454
10297  f454
10298  f454  ===> nächsten Datenblock von FSD lesen <===
10299  f454
10300  f454  20 3d f6   btnxbk jsr ud60         Abfrage Stopp/Shift-Taste/Spalte7
10301  f457  20 e1 ff          jsr kstop        Stopp-Taste üb. Vector abfragen
                                                (Sprungv.)
10302  f45a  f0 4a             beq btbrke       Abbruch durch Stopp-Taste
10303  f45c
10304  f45c  20 c5 f4          jsr brtblk       Datenblock vom FSD lesen
10305  f45f  b0 51             bcs btomer       Abbruch durch out of memory
10306  f461
10307  f461  ===> Transfer-Status lesen <===
10308  f461
10309  f461  20 ba f4          jsr brtbyt       Zeichen vom FSD lesen
10310  f464  c9 02             cmp #2
10311  f466  90 06             bcc btblck       normalen 254-Byte Block lesen
10312  f468
10313  f468  c9 1f             cmp #31
10314  f46a  f0 0b             beq btlnls       Länge letzten Datenblock von FSD
                                                lesen
10315  f46c  d0 2a             bne btlode       Abbruch durch Lade- (Timeout-)
                                                Fehler
10316  f46e
10317  f46e
10318  f46e
10319  f46e  ===> normalen 254-Byte Block lesen <===
10320  f46e
10321  f46e  20 03 f5   btblck jsr wiggle       CLOCK OUT flippen
10322  f471  a9 fe             lda #254         normaler Datenblock 254 Bytes
10323  f473  85 a5             sta cntdn        Länge akt. Datenblock
10324  f475  d0 dd             bne btnxbk       nächsten Datenblock von FSD lesen
                                                (immer)
10325  f477
10326  f477
10327  f477  ===> Länge letzten Datenblock von FSD lesen <===
10328  f477
10329  f477  20 03 f5   btlnls jsr wiggle       CLOCK OUT flippen
10330  f47a  20 ba f4          jsr brtbyt       Zeichen vom FSD lesen
10331  f47d  85 a5             sta cntdn        Länge akt. Datenblock
10332  f47f
10333  f47f
10334  f47f  ===> letzten Datenblock lesen, EOI-Flag <===
10335  f47f
10336  f47f  20 03 f5   brteoi jsr wiggle       CLOCK OUT flippen
10337  f482  20 c5 f4          jsr brtblk       Datenblock vom FSD lesen
10338  f485  b0 2b             bcs btomer       Abbruch durch out of memory
10339  f487
10340  f487  a9 40             lda #%01000000   EOF-Bit setzen
10341  f489  20 57 f7          jsr udst         ac-Bits in Status setzen
10342  f48c
```

```
10343  f48c
10344  f48c  ===> akt. File schließen <===
10345  f48c
10346  f48c  20 45 e5   brtend jsr clkhi        CLOCK-Leitung hi
10347  f48f  58                cli
10348  f490
10349  f490  a5 b8             lda la           akt. log. Filenummer
10350  f492  38                sec
10351  f493  20 c3 ff          jsr kclose       log. File üb. Vector schließen
                                                (Sprungv.)
10352  f496
10353  f496  18                clc              ok
10354  f497  60                rts
10355  f498
10356  f498
10357  f498  ===> Abbruch durch Lade- (Timeout-) Fehler <===
10358  f498
10359  f498  a9 02      btlode lda #%00000010   Timeout-Bit
10360  f49a  20 57 f7          jsr udst         ac-Bits in Status setzen
10361  f49d
10362  f49d  20 8c f4          jsr brtend       akt. File schließen
10363  f4a0  68                pla              Rücksprungadresse
10364  f4a1  68                pla              verwerfen
10365  f4a2
10366  f4a2  a9 29             lda #$29         BASIC-Fehler #29: load error
10367  f4a4  38                sec              Fehler
10368  f4a5  60                rts
10369  f4a6
10370  f4a6
10371  f4a6  ===> Abbruch durch Stopp-Taste <===
10372  f4a6
10373  f4a6  20 8c f4   btbrke jsr brtend       akt. File schließen
10374  f4a9
10375  f4a9  a9 00             lda #0
10376  f4ab  85 b9             sta sa           akt. Sekundäradresse
10377  f4ad
10378  f4ad  68                pla              Rücksprungadresse
10379  f4ae  68                pla              verwerfen
10380  f4af
10381  f4af  4c b5 f5          jmp break        Abbruch I/O IEC-Bus
10382  f4b2
10383  f4b2
10384  f4b2  ===> Abbruch durch out of memory <===
10385  f4b2
10386  f4b2  20 8c f4   btomer jsr brtend       akt. File schließen
10387  f4b5  68                pla
10388  f4b6  68                pla
10389  f4b7  4c 97 f6          jmp errora
10390  f4ba
10391  f4ba
10392  f4ba  ===> Zeichen vom FSD lesen <===
10393  f4ba
10394  f4ba  a9 08      brtbyt lda #%00001000
10395  f4bc
10396  f4bc   ....3...  IRQ Maskenbit Serial Port
10397  f4bc
```

```
10398  f4bc  ===> Warten auf SDR IRQ <===
10399  f4bc
10400  f4bc  2c 0d dc   brtbt1 bit d1icr        CIA1 Interrupt Control-Register
10401  f4bf  f0 fb             beq brtbt1       warten
10402  f4c1
10403  f4c1  ad 0c dc          lda d1sdr        akt. Zeichen aus SDR
10404  f4c4  60                rts
10405  f4c5
10406  f4c5
10407  f4c5  ===> Datenblock vom FSD lesen <===
10408  f4c5
10409  f4c5  a9 08      brtblk lda #%00001000
10410  f4c7
10411  f4c7   ....3...  IRQ Maskenbit Serial Port
10412  f4c7
10413  f4c7  ===> Warten auf SDR IRQ <===
10414  f4c7
10415  f4c7  2c 0d dc   brtbl1 bit d1icr        CIA1 Interrupt Control-Register
10416  f4ca  f0 fb             beq brtbl1       warten
10417  f4cc
10418  f4cc  ac 0c dc          ldy d1sdr        akt. Zeichen aus SDR
10419  f4cf
10420  f4cf  ad 00 dd          lda d2pra        CIA2 Datenport A
10421  f4d2  49 10             eor #%00010000   CLOCK OUT flippen
10422  f4d4  8d 00 dd          sta d2pra        CIA2 Datenport A
10423  f4d7
10424  f4d7  98                tya              akt. Zeichen
10425  f4d8
10426  f4d8  ===> Test load / verify <===
10427  f4d8
10428  f4d8  a4 93             ldy verck        Flag für load/verify
10429  f4da  f0 12             beq brtbl2       load
10430  f4dc
10431  f4dc
10432  f4dc  ===> verify <===
10433  f4dc
10434  f4dc  85 bd             sta ochar        akt. Zeichen
10435  f4de
10436  f4de  a0 00             ldy #$00
10437  f4e0  20 c9 f7          jsr eally        Byte indirekt über 'eal' lesen
10438  f4e3  c5 bd             cmp ochar        akt. Zeichen
10439  f4e5  f0 0a             beq brtbl3       verify ok
10440  f4e7
10441  f4e7  ===> verify error <===
10442  f4e7
10443  f4e7  a9 10             lda #%00010000
10444  f4e9  20 57 f7          jsr udst         ac-Bits in Status setzen
10445  f4ec  d0 03             bne brtbl3       ,weiter verify
10446  f4ee
10447  f4ee  20 bf f7   brtbl2 jsr ealsy        Byte indirekt über 'eal' speichern
10448  f4f1  e6 ae      brtbl3 inc eal          addrl Ende load/save
10449  f4f3  d0 08             bne brtbl4
10450  f4f5
10451  f4f5  e6 af             inc eah          addrh Ende load/save
10452  f4f7  a5 af             lda eah          addrh Ende load/save
10453  f4f9
```

```
10454  f4f9  ===> Test auf Überschreiben MMU-Register <===
10455  f4f9
10456  f4f9  c9 ff            cmp #$ff
10457  f4fb  f0 05            beq brtbl5
10458  f4fd
10459  f4fd  c6 a5     brtbl4 dec cntdn        Länge Datenblock
10460  f4ff  d0 c4            bne brtblk       Datenblock vom FSD lesen
10461  f501
10462  f501  18               clc              ok
10463  f502  60        brtbl5 rts
10464  f503
10465  f503
10466  f503  ===> CLOCK OUT flippen <===
10467  f503
10468  f503  ad 00 dd  wiggle lda d2pra        CIA2 Datenport A
10469  f506  49 10            eor #%00010000   CLOCK OUT flippen
10470  f508  8d 00 dd         sta d2pra        CIA2 Datenport A
10471  f50b
10472  f50b  60               rts
10473  f50c
10474  f50c
10475  f50c  ===> Floppy-Befehl für Fast-Serial-Interface <===
10476  f50c
10477  f50c  1f 30 55  brtcmd .byte $1f,'0u'
10478  f50f
10479  f50f
10480  f50f  ===> Direktmodus Ausgabe 'searching (for fn)' <===
10481  f50f
10482  f50f  a5 9d     luking lda msgflg       Flag disable Ausgabe OPSYS-Meldungen
10483  f511  10 1f            bpl lurts
10484  f513
10485  f513  a0 0c            ldy #12          Zeiger auf 'searching'
10486  f515  20 22 f7         jsr msg          Ausgabe Systemmeldungen, yr=offset
10487  f518
10488  f518  a5 b7            lda fnlen        Länge akt. Filename
10489  f51a  f0 16            beq lurts        kein Filenamen
10490  f51c
10491  f51c  a0 17            ldy #23          Zeiger auf 'for'
10492  f51e  20 22 f7         jsr msg          Ausgabe Systemmeldungen, yr=offset
10493  f521
10494  f521
10495  f521  ===> Ausgabe akt. Filename <===
10496  f521
10497  f521  a4 b7     outfn  ldy fnlen         Länge akt. Filename
10498  f523  f0 0d            beq lurts        kein Filenamen
10499  f525
10500  f525  a0 00            ldy #0
10501  f527  20 ae f7  outfn1 jsr fnadry       LDA (FNADR),y
10502  f52a  20 d2 ff         jsr kbsout       Ausgabe ac üb. Vector auf akt. Kanal
                                               (Sprungv.)
10503  f52d  c8               iny
10504  f52e  c4 b7            cpy fnlen        Länge akt. Filename
10505  f530  d0 f5            bne outfn1
10506  f532
10507  f532  60        lurts  rts
10508  f533
10509  f533
```

```
10510  f533  ===> Direktmodus Ausgabe 'loading ...' <===
10511  f533
10512  f533  a0 49      loding ldy #73          Zeiger auf 'loading'
10513  f535  a5 93             lda verck        Flag für load/verify
10514  f537  f0 02             beq lodng1       load
10515  f539
10516  f539  a0 59             ldy #89          Zeiger auf 'verifying'
10517  f53b  4c 1e f7   lodng1 jmp spmsg        Test errorflag, Ausgabe
                                                Systemmeldung
10518  f53e
10519  f53e
10520  f53e  ===> save ab Adr. ZP (ac) bis xr, yr <===
10521  f53e
10522  f53e  86 ae      savesp stx eal          addrl Ende load/save
10523  f540  84 af             sty eah          addrh Ende load/save
10524  f542
10525  f542  aa                tax
10526  f543  b5 00             lda 0,x
10527  f545  85 c1             sta stal         addrl Startadresse save
10528  f547
10529  f547  b5 01             lda 1,x
10530  f549  85 c2             sta stah         addrh Startadresse save
10531  f54b
10532  f54b  6c 32 03          jmp (isave)      Abspeichern auf log. File ($f54e)
10533  f54e
10534  f54e
10535  f54e  ===> Abspeichern auf logisches File <===
10536  f54e
10537  f54e  a5 ba      nsave  lda fa           akt. Gerätenummer
10538  f550  c9 01             cmp #1           Gerätenummer Kassette
10539  f552  f0 74             beq svtape       schreiben auf Kassette
10540  f554
10541  f554  c9 04             cmp #4
10542  f556  b0 09             bcs svdisk       Ausgabe Speicherbereich auf IEC-Bus
10543  f558
10544  f558
10545  f558  ===> I/O-Fehler #9: illegal device number <===
10546  f558
10547  f558  4c 94 f6   erro9  jmp error9       I/O-Fehler #9: illegal device number
10548  f55b
10549  f55b  ===> I/O-Fehler #8: missing file name <===
10550  f55b
10551  f55b  4c 91 f6   erro8  jmp error8       I/O-Fehler #8: missing File name
10552  f55e
10553  f55e  ===> I/O-Fehler #4: file not found <===
10554  f55e
10555  f55e  4c 85 f6   erro4  jmp error4       I/O-Fehler #4: File not found
10556  f561
10557  f561
10558  f561  ===> Ausgabe Speicherbereich auf IEC-Bus <===
10559  f561
10560  f561  a4 b7      svdisk ldy fnlen         Länge akt. Filename
10561  f563  f0 f6             beq erro8        I/O-Fehler #8: missing File name
10562  f565
10563  f565  a9 61             lda #%01100001
10564  f567  85 b9             sta sa           akt. Sekundäradresse
10565  f569
10566  f569  20 cb f0          jsr openi        Open IEC-Kanal, Name senden
```

```
10567  f56c  20 bc f5          jsr saving       im Direktmodus Ausgabe 'saving ...'
10568  f56f
10569  f56f  a5 ba             lda fa           akt. Gerätenummer
10570  f571  20 3e e3          jsr listn        Ausgabe LISTEN auf IEC-Bus
10571  f574
10572  f574  a5 b9             lda sa           akt. Sekundäradresse
10573  f576  20 d2 e4          jsr secnd        Ausgabe Sek.Adr. nach LISTEN auf
                                                 IEC-Bus
10574  f579
10575  f579  a0 00             ldy #0
10576  f57b  20 51 ed          jsr rd300        stah swap sah
10577  f57e  20 03 e5          jsr ciout        addrl Start Speicherbereich auf IEC
10578  f581
10579  f581  a5 ad             lda sah          addrh Start Speicherbereich
10580  f583  20 03 e5          jsr ciout        Ausgabe ac auf IEC-Bus
10581  f586
10582  f586  20 b7 ee   sv30    jsr cmpste       Adresse 'akt./ende' vergleichen
10583  f589  b0 10              bcs sv31         Endadresse Speicherbereich, fertig
10584  f58b
10585  f58b  ===> Schleife save Speicherbereich IEC <===
10586  f58b
10587  f58b  20 cc f7          jsr sally        Byte indirekt über 'sal' lesen
10588  f58e  20 03 e5          jsr ciout        Ausgabe ac auf IEC-Bus
10589  f591
10590  f591  20 e1 ff          jsr kstop        Stopp-Taste üb. Vector abfragen
                                                 (Sprungv.)
10591  f594  f0 1f             beq break        Abbruch durch Stopp-Taste
10592  f596
10593  f596  20 c1 ee          jsr incsal       'load/store-Adresse' erhöhen
10594  f599  d0 eb             bne sv30
10595  f59b
10596  f59b
10597  f59b  ===> Endadresse Speicherbereich, fertig <===
10598  f59b
10599  f59b  20 26 e5   sv31    jsr nunlsn       Ausgabe UNLISTEN auf IEC-Bus
10600  f59e
10601  f59e
10602  f59e  ===> IEC-Programmmfile schließen <===
10603  f59e
10604  f59e  24 b9      clsei   bit sa           akt. Sekundäradresse
10605  f5a0  30 11              bmi clsrts
10606  f5a2
10607  f5a2  a5 ba              lda fa           akt. Gerätenummer
10608  f5a4  20 3e e3           jsr listn        Ausgabe LISTEN auf IEC-Bus
10609  f5a7
10610  f5a7  a5 b9              lda sa           akt. Sekundäradresse
10611  f5a9  29 ef              and #%11101111
10612  f5ab  09 e0              ora #%11100000
10613  f5ad  20 d2 e4           jsr secnd        Ausgabe Sek.Adr. nach LISTEN auf
                                                 IEC-Bus
10614  f5b0
10615  f5b0
10616  f5b0  ===> Ausgabe UNLISTEN auf IEC-Bus (BASIC) <===
10617  f5b0
10618  f5b0  20 26 e5   cunlsn jsr nunlsn       Ausgabe UNLISTEN auf IEC-Bus
10619  f5b3  18         clsrts clc              ok
10620  f5b4  60                rts
10621  f5b5
```

```
10622  f5b5
10623  f5b5                    .end
10624  f5b5                    .lib c128.kernal3
10625  f5b5
10627  f5b5
10628  f5b5
10629  f5b5  ===> Abbruch I/O IEC-Bus <===
10630  f5b5
10631  f5b5  20 9e f5   break  jsr clsei        IEC-Programmmfile schließen
10632  f5b8  a9 00             lda #0
10633  f5ba  38                sec              Fehler
10634  f5bb  60                rts
10635  f5bc
10636  f5bc
10637  f5bc  ===> im Direktmodus Ausgabe 'saving ...' <===
10638  f5bc
10639  f5bc  a5 9d      saving lda msgflg       Flag disable Ausgabe OPSYS-Meldungen
10640  f5be  10 37             bpl svrts        nicht ausgeben
10641  f5c0
10642  f5c0  a0 51             ldy #81          Zeiger auf 'saving'
10643  f5c2  20 22 f7          jsr msg          Ausgabe Systemmeldungen, yr=offset
10644  f5c5  4c 21 f5          jmp outfn        Ausgabe akt. Filename
10645  f5c8
10646  f5c8
10647  f5c8  ===> schreiben auf Kassette <===
10648  f5c8
10649  f5c8  20 80 e9   svtape jsr tapadr       Zeiger akt. Kass.puffer einrichten
10650  f5cb  90 8b             bcc erro9        I/O-Fehler #9: illegal device number
10651  f5cd
10652  f5cd  20 e9 e9          jsr cste2        prüfen, Meldungen ausgeben, warten
                                                auf 'play & record'
10653  f5d0  b0 25             bcs svrts        Abbruch durch Stopp-Taste
10654  f5d2
10655  f5d2  20 bc f5          jsr saving       im Direktmodus Ausgabe 'saving ...'
10656  f5d5
10657  f5d5  a2 03             ldx #3           Vorgabe Headermarke
                                                Maschinenprogramm
10658  f5d7
10659  f5d7  a5 b9             lda sa           akt. Sekundäradresse
10660  f5d9  29 01             and #%00000001
10661  f5db  d0 02             bne sv106
10662  f5dd
10663  f5dd  a2 01             ldx #1           Headermarke BASIC-Programm
10664  f5df  8a         sv106  txa
10665  f5e0  20 19 e9          jsr tapeh        PRG-Header/Ende-Block auf Kassette
                                                schreiben
10666  f5e3  b0 12             bcs svrts
10667  f5e5
10668  f5e5  20 18 ea          jsr twrt         PRG-file auf Kassette schreiben
10669  f5e8  b0 0d             bcs svrts        Abbruch durch Stopp-Taste
10670  f5ea
10671  f5ea  a5 b9             lda sa           akt. Sekundäradresse
10672  f5ec  29 02             and #%00000010   Bit für EOT
10673  f5ee  f0 06             beq sv110
10674  f5f0
10675  f5f0  ===> Ausgabe EOT-Block <===
10676  f5f0
10677  f5f0  a9 05             lda #5           Headermarke EOT-Block
```

```
10734  f645   nach der Tastaturabfrage wird mit CIA 1 Port A die Spalte 7
10735  f645   abgefragt. Die Stopp-Taste hat in der Matrix Spalte 7 / Reihe 7.
10736  f645
10737  f645  aa               tax
10738  f646  30 13            bmi ud80          Stopp-Taste nicht gedrückt
10739  f648
10740  f648
10741  f648  ===> Stopp-Taste gedrückt <===
10742  f648
10743  f648  a2 bd            ldx #%10111101    Spalten 6 und 1 abfragen
                                                (Shift-Tasten)
10744  f64a  8e 00 dc         stx d1pra
10745  f64d
10746  f64d  ae 01 dc  ud70   ldx rows
10747  f650  ec 01 dc         cpx rows          Tastaturmatrix: Reihen
10748  f653  d0 f8            bne ud70          entprellen
10749  f655
10750  f655  8d 00 dc         sta d1pra         %01111111 ;Spalte 7 abfragen
10751  f658
10752  f658  e8               inx               %11111111 = keine Taste gedrückt
10753  f659  d0 02            bne ud90          keine Shift-Taste gedrückt
10754  f65b
10755  f65b  85 91      ud80  sta stkey         Status Shift/Stopp/Spalte7
10756  f65d  60         ud90  rts
10757  f65e
10758  f65e
10759  f65e  ===> System-Uhr nach ac, xr, yr <===
10760  f65e
10761  f65e  78         rdtim sei
10762  f65f  a5 a2            lda time+2        Uhrzeit in 1/60 s (jiffies)
10763  f661  a6 a1            ldx time+1        Uhrzeit in 1/60 s (jiffies)
10764  f663  a4 a0            ldy time          Uhrzeit in 1/60 s (jiffies)
10765  f665
10766  f665
10767  f665  ===> System-Uhr ac, xr, yr setzen <===
10768  f665
10769  f665  78         settim sei
10770  f666  85 a2            sta time+2        Uhrzeit in 1/60 s (jiffies)
10771  f668  86 a1            stx time+1        Uhrzeit in 1/60 s (jiffies)
10772  f66a  84 a0            sty time          Uhrzeit in 1/60 s (jiffies)
10773  f66c
10774  f66c  58               cli
10775  f66d  60               rts
10776  f66e
10777  f66e
10778  f66e  ===> Stopp-Taste abfragen <===
10779  f66e
10780  f66e  a5 91      nstop lda stkey         Flag für Stopp- und RVS-Taste
10781  f670  c9 7f            cmp #%01111111    Spalte7, Reihe7 maskieren (Stopp-T.)
10782  f672  d0 07            bne nstopx        nicht gedrückt
10783  f674
10784  f674  ===> I/O-Kanäle schließen, Vorgabe-I/O <===
10785  f674
10786  f674  08               php
10787  f675  20 cc ff         jsr kclrch        Ein-/Ausgabe-Kanäle üb. Vector
                                                schließen (Sprungv.)
10788  f678  85 d0            sta ndx           Bytes im Tast.puffer
10789  f67a
```

```
10790  f67a  28               plp
10791  f67b  60        nstopx rts             Equal=1: Stopp, Equal=0: nicht Stopp
10792  f67c
10793  f67c
10794  f67c
10795  f67c  ===> I/O-Fehler #1: too many Files <===
10796  f67c
10797  f67c  a9 01     error1 lda #1
10798  f67e  2c               .byte skip2
10799  f67f
10800  f67f
10801  f67f  ===> I/O-Fehler #2: File Open <===
10802  f67f
10803  f67f  a9 02     error2 lda #2
10804  f681  2c               .byte skip2
10805  f682
10806  f682
10807  f682  ===> I/O-Fehler #3: File not Open <===
10808  f682
10809  f682  a9 03     error3 lda #3
10810  f684  2c               .byte skip2
10811  f685
10812  f685
10813  f685  ===> I/O-Fehler #4: File not found <===
10814  f685
10815  f685  a9 04     error4 lda #4
10816  f687  2c               .byte skip2
10817  f688
10818  f688
10819  f688  ===> I/O-Fehler #5: device not present <===
10820  f688
10821  f688  a9 05     error5 lda #5
10822  f68a  2c               .byte skip2
10823  f68b
10824  f68b
10825  f68b  ===> I/O-Fehler #6: not input File <===
10826  f68b
10827  f68b  a9 06     error6 lda #6
10828  f68d  2c               .byte skip2
10829  f68e
10830  f68e
10831  f68e  ===> I/O-Fehler #7: not output File <===
10832  f68e
10833  f68e  a9 07     error7 lda #7
10834  f690  2c               .byte skip2
10835  f691
10836  f691
10837  f691  ===> I/O-Fehler #8: missing File name <===
10838  f691
10839  f691  a9 08     error8 lda #8
10840  f693  2c               .byte skip2
10841  f694
10842  f694
10843  f694  ===> I/O-Fehler #9: illegal device number <===
10844  f694
10845  f694  a9 09     error9 lda #9
10846  f696  2c               .byte skip2
10847  f697
```

```
10848  f697  a9 10      errora lda #16
10849  f699
10850  f699  48               pha             Fehlernummer
10851  f69a
10852  f69a  20 cc ff          jsr kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                               schließen (Sprungv.)
10853  f69d
10854  f69d  a0 00             ldy #0          Zeiger auf i/o error
10855  f69f
10856  f69f  24 9d             bit msgflg      Flag disable Ausgabe OPSYS-Meldungen
10857  f6a1
10858  f6a1   x1xxxxxx  enable Ausgabe I/O-Fehler (nur Monitor)
10859  f6a1
10860  f6a1  50 0a             bvc erexit
10861  f6a3
10862  f6a3  ===> OPSYS-Meldung ausgeben <===
10863  f6a3
10864  f6a3  20 22 f7          jsr msg         Ausgabe Systemmeldungen, yr=offset
10865  f6a6  68                pla             Fehlernummer
10866  f6a7  48                pha
10867  f6a8
10868  f6a8  09 30             ora #$30        HEX nach ASCII
10869  f6aa  20 d2 ff          jsr kbsout      Fehlernummer ausgeben
10870  f6ad
10871  f6ad  68         erexit pla             Fehlernummer
10872  f6ae  38                sec             Fehlerflag
10873  f6af  60                rts
10874  f6b0
10875  f6b0
10876  f6b0  0d         msgtbl .byte cr
10877  f6b1  49 2f 4f          .byte 'i/o error ', $a3
10878  f6b4  20 45 52
10879  f6b7  52 4f 52
10880  f6ba  20 a3
10881  f6bc  0d 53 45          .byte cr, 'searching', sftspc
10882  f6bf  41 52 43
10883  f6c2  48 49 4e
10884  f6c5  47 a0
10885  f6c7  46 4f 52          .byte 'for', sftspc
10886  f6ca  a0
10887  f6cb  0d 50 52          .byte cr, 'press play on tapE'
10888  f6ce  45 53 53
10889  f6d1  20 50 4c
10890  f6d4  41 59 20
10891  f6d7  4f 4e 20
10892  f6da  54 41 50
10893  f6dd  c5
10894  f6de  50 52 45          .byte 'press record & play on tapE'
10895  f6e1  53 53 20
10896  f6e4  52 45 43
10897  f6e7  4f 52 44
10898  f6ea  20 26 20
10899  f6ed  50 4c 41
10900  f6f0  59 20 4f
10901  f6f3  4e 20 54
10902  f6f6  41 50 c5
10903  f6f9  0d 4c 4f          .byte cr, 'loadinG'
10904  f6fc  41 44 49
```

```
10905  f6ff  4e c7
10906  f701  0d 53 41          .byte cr, 'saving', sftspc
10907  f704  56 49 4e
10908  f707  47 a0
10909  f709  0d 56 45          .byte cr, 'verifyinG'
10910  f70c  52 49 46
10911  f70f  59 49 4e
10912  f712  c7
10913  f713  0d 46 4f          .byte cr, 'found', sftspc
10914  f716  55 4e 44
10915  f719  a0
10916  f71a  0d 4f 4b          .byte cr, 'ok', sftcr
10917  f71d  8d
10918  f71e
10919  f71e
10920  f71e  ===> Test errorflag, Ausgabe Systemmeldung <===
10921  f71e
10922  f71e  24 9d      spmsg  bit msgflg       Flag disable Ausgabe OPSYS-Meldungen
10923  f720
10924  f720   1xxxxxxxx  enable Ausgabe OPSYS-Meldungen (Direktmod.)
10925  f720
10926  f720  10 0d              bpl msgrts       Bit7 nicht gesetzt
10927  f722
10928  f722
10929  f722  ===> Ausgabe Systemmeldungen, yr=offset <===
10930  f722
10931  f722  b9 b0 f6   msg     lda msgtbl,y
10932  f725  08                 php              MINUS-Flag retten
10933  f726  29 7f              and #%01111111   ,Bit7 löschen
10934  f728  20 d2 ff           jsr kbsout       Ausgabe ac üb. Vector auf akt. Kanal
                                                  (Sprungv.)
10935  f72b  c8                 iny              Zeiger in Text
10936  f72c  28                 plp              MINUS-Flag
10937  f72d  10 f3              bpl msg          nicht gesetzt
10938  f72f
10939  f72f  18         msgrts  clc              ok
10940  f730  60                 rts
10941  f731
10942  f731
10943  f731  ===> fnlen, fnadr/+1 aus ac, xr, yr setzen <===
10944  f731
10945  f731  85 b7      setnam  sta fnlen        Länge akt. Filename
10946  f733  86 bb              stx fnadr        addrl akt. Filename
10947  f735  84 bc              sty fnadr+1      addrh akt. Filename
10948  f737
10949  f737  60                 rts
10950  f738
10951  f738
10952  f738  ===> la, fa, sa aus ac, xr, yr setzen <===
10953  f738
10954  f738  85 b8      setlfs  sta la           akt. log. Filenummer
10955  f73a  86 ba              stx fa           akt. Gerätenummer
10956  f73c  84 b9              sty sa           akt. Sekundäradresse
10957  f73e
10958  f73e  60                 rts
10959  f73f
10960  f73f
```

```
10961  f73f  ===> Bank für load/sav/verify setzen <===
10962  f73f
10963  f73f  85 c6      setbnk sta ba          akt. Bank für Speicher-Operationen
10964  f741  86 c7             stx fnbank      Bank des akt. Filenamens
10965  f743
10966  f743  60                rts
10967  f744
10968  f744
10969  f744
10970  f744  ===> akt. Status (RS 232 bzw. System) nach ac <===
10971  f744
10972  f744  a5 ba       readss lda fa          akt. Gerätenummer
10973  f746  c9 02              cmp #2          Geräteadresse RS 232
10974  f748  d0 0b              bne readst      System-Status nach ac
10975  f74a
10976  f74a  ===> Status RS 232 nach System-Status <===
10977  f74a
10978  f74a  ad 14 0a           lda rsstat      RS 232 Status-Byte
10979  f74d  48                 pha
10980  f74e
10981  f74e  a9 00              lda #%00000000  Rücksetzen
10982  f750  8d 14 0a           sta rsstat      RS 232 Status-Byte
10983  f753
10984  f753  68                 pla
10985  f754  60                 rts
10986  f755
10987  f755
10988  f755  ===> System-Status nach ac <===
10989  f755
10990  f755  a5 90       readst lda status      I/O-Status 'st'
10991  f757
10992  f757
10993  f757  ===> ac-Bits in Status setzen <===
10994  f757
10995  f757  05 90       udst   ora status      I/O-Status 'st'
10996  f759  85 90              sta status      I/O-Status 'st'
10997  f75b  60                 rts
10998  f75c
10999  f75c
11000  f75c  ===> Flag setzen Ausgabe/Unterdr. OPSYS-Meldung <===
11001  f75c
11002  f75c  85 9d       setmsg sta msgflg      Flag disable Ausgabe OPSYS-Meldungen
11003  f75e  60                 rts
11004  f75f
11005  f75f
11006  f75f  ===> Timeout Flag IEC-Bus setzen <===
11007  f75f
11008  f75f  8d 0e 0a    settmo sta timout      IEC-Timeout enable Flag
11009  f762  60                 rts            wird vom OPSYS nicht verwertet
11010  f763
11011  f763
11012  f763  ===> höchste Speicheradresse lesen/schreiben <===
11013  f763
11014  f763  90 06       memtop bcc settop      xr,yr in Zeiger höchste
                                                Speicheradresse
11015  f765
11016  f765  ae 07 0a           ldx memsiz      addrl Ende des Benutzerspeichers
11017  f768  ac 08 0a           ldy memsiz+1    addrh Ende des Benutzerspeichers
```

```
11018  f76b
11019  f76b
11020  f76b  ===> xr,yr in Zeiger höchste Speicheradresse <===
11021  f76b
11022  f76b  8e 07 0a   settop stx memsiz       addrl Ende des Benutzerspeichers
11023  f76e  8c 08 0a         sty memsiz+1     addrh Ende des Benutzerspeichers
11024  f771  60               rts
11025  f772
11026  f772
11027  f772
11028  f772  ===> unterste Speicheradresse lesen/schreiben <===
11029  f772
11030  f772  90 06      membot bcc setbot       xr,yr in Zeiger Anfang
                                                Benutzerspeicher
11031  f774
11032  f774  ae 05 0a         ldx memstr       addrl Anfang des Benutzerspeichers
11033  f777  ac 06 0a         ldy memstr+1     addrh Anfang des Benutzerspeichers
11034  f77a
11035  f77a
11036  f77a  ===> xr,yr in Zeiger Anfang Benutzerspeicher <===
11037  f77a
11038  f77a  8e 05 0a   setbot stx memstr       addrl Anfang des Benutzerspeichers
11039  f77d  8c 06 0a         sty memstr+1     addrh Anfang des Benutzerspeichers
11040  f780  60               rts
11041  f781
11042  f781
11043  f781  ===> Basisadresse I/O-Bausteine nach xr/yr <===
11044  f781
11045  f781  a2 00      iobase ldx #<chips      Basisadresse I/O-Chips
11046  f783  a0 d0             ldy #>chips
11047  f785  60               rts
11048  f786
11049  f786
11050  f786  ===> Geräte-Adr. über Sek.Adr. (yr) suchen <===
11051  f786
11052  f786  98         lkupsa tya
11053  f787  a6 98             ldx ldtnd        Anzahl offene Files
11054  f789  ca         lkups2 dex
11055  f78a  30 0f             bmi lkupng
11056  f78c
11057  f78c  dd 76 03          cmp sat,x        Tab. der Sekundär-Adr.
11058  f78f  d0 f8             bne lkups2
11059  f791
11060  f791  20 12 f2   lkupok jsr getlfs       la, fa, sa x-ind. aus Filetab.
11061  f794  aa                tax
11062  f795  a5 b8             lda la           akt. log. Filenummer
11063  f797  a4 b9             ldy sa           akt. Sekundäradresse
11064  f799  18                clc
11065  f79a  60                rts
11066  f79b
11067  f79b
11068  f79b  38         lkupng sec              Fehler-Flag
11069  f79c  60                rts
11070  f79d
11071  f79d
11072  f79d  ===> Geräte- & Sek.-Adr. über la (ac) suchen <===
11073  f79d
11074  f79d  aa         lkupla tax
```

```
11075  f79e  20 02 f2          jsr lookup      log. Filenummer ac in Tab. suchen
11076  f7a1  f0 ee             beq lkupok
11077  f7a3  d0 f6             bne lkupng
11078  f7a5
11079  f7a5
11080  f7a5  ===> DMA-Anforderung ext. RAM zulassen <===
11081  f7a5
11082  f7a5  bd f0 f7   dmacal lda config,x    MMU-Werte für Banknummer 0 - 15
11083  f7a8  29 fe             and #%11111110
11084  f7aa
11085  f7aa   xxxxxxx0  I/O Chips
11086  f7aa
11087  f7aa  aa                tax
11088  f7ab  4c f0 03          jmp dodma       DMA-Register setzen
11089  f7ae
11090  f7ae
11091  f7ae  ===> LDA (FNADR),y <===
11092  f7ae
11093  f7ae  8e 35 0a   fnadry stx fnadrx      xr retten
11094  f7b1
11095  f7b1  a6 c7             ldx fnbank      Bank des akt. Filenamens
11096  f7b3  a9 bb             lda #<fnadr     addrl akt. Filename
11097  f7b5  20 d0 f7          jsr indfet      LDA (FETVEC),y von Bank in xr
11098  f7b8
11099  f7b8  ae 35 0a          ldx fnadrx      xr zurück
11100  f7bb  60                rts
11101  f7bc
11102  f7bc
11103  f7bc  ===> Byte indirekt über 'sal' speichern <===
11104  f7bc
11105  f7bc  a2 ac      salsy  ldx #<sal       addrl für load/save
11106  f7be  2c                .byte skip2
11107  f7bf
11108  f7bf
11109  f7bf  ===> Byte indirekt über 'eal' speichern <===
11110  f7bf
11111  f7bf  a2 ae      ealsy  ldx #<eal       addrl Ende load/save
11112  f7c1  8e b9 02          stx stashx      STAVEC
11113  f7c4
11114  f7c4  a6 c6             ldx ba          akt. Bank für Speicher-Operationen
11115  f7c6  4c da f7          jmp indsta      STA (STAVEC),y in Bank in xr
11116  f7c9
11117  f7c9
11118  f7c9  ===> Byte indirekt über 'eal' lesen <===
11119  f7c9
11120  f7c9  a9 ae      eally  lda #<eal       addrl Ende load/save
11121  f7cb  2c                .byte skip2
11122  f7cc
11123  f7cc
11124  f7cc  ===> Byte indirekt über 'sal' lesen <===
11125  f7cc
11126  f7cc  a9 ac      sally  lda #<sal       addrl für load/save
11127  f7ce  a6 c6             ldx ba          akt. Bank für Speicher-Operationen
11128  f7d0
11129  f7d0
11130  f7d0  ===> LDA (FETVEC),y von Bank in xr <===
11131  f7d0
11132  f7d0  8d aa 02   indfet sta fetchx      LDAVEC
```

```
11133  f7d3
11134  f7d3  bd f0 f7          lda config,x     MMU-Werte für Banknummer 0 - 15
11135  f7d6  aa                tax
11136  f7d7  4c a2 02          jmp fetch        LDA (LDAVEC),y, Bank xr
11137  f7da
11138  f7da
11139  f7da  ===> STA (STAVEC),y in Bank in xr <===
11140  f7da
11141  f7da  48         indsta pha
11142  f7db
11143  f7db  bd f0 f7          lda config,x     MMU-Werte für Banknummer 0 - 15
11144  f7de  aa                tax
11145  f7df  68                pla
11146  f7e0  4c af 02          jmp stash        STA (STAVEC),y, Bank xr
11147  f7e3
11148  f7e3
11149  f7e3  ===> CMP (CMPVEC),y mit Bank in xr <===
11150  f7e3
11151  f7e3  48         indcmp pha
11152  f7e4
11153  f7e4  bd f0 f7          lda config,x     MMU-Werte für Banknummer 0 - 15
11154  f7e7  aa                tax
11155  f7e8  68                pla
11156  f7e9  4c be 02          jmp cmpare       CMP (CMPVEC),y, Bank xr
11157  f7ec
11158  f7ec
11159  f7ec  ===> MMU-Wert zu Bank in xr nach ac <===
11160  f7ec
11161  f7ec  bd f0 f7   getcfg lda config,x     MMU-Werte für Banknummer 0 - 15
11162  f7ef  60                rts
11163  f7f0
11164  f7f0
11165  f7f0  ===> MMU-Werte für Banknummer 0 - 15 <===
11166  f7f0
11167  f7f0  3f         config .byte %00111111
11168  f7f1
11169  f7f1   00xxxxxx  RAM Bank 0
11170  f7f1   xx11xxxx  RAM Bank 0  $c000-$cfff
11171  f7f1   xxxx11xx  RAM Bank 0  $8000-$bfff
11172  f7f1   xxxxxx1x  RAM Bank 0  $4000-$7fff
11173  f7f1   xxxxxxx1  RAM Bank 0  $d000-$dfff
11174  f7f1
11175  f7f1  7f                .byte %01111111  dsgl. für RAM Bank 1
11176  f7f2  bf                .byte %10111111  dsgl. für RAM Bank 2 (sonst 0)
11177  f7f3  ff                .byte %11111111  dsgl. für RAM Bank 3 (sonst 1)
11178  f7f4
11179  f7f4  16                .byte %00010110
11180  f7f5
11181  f7f5   00xxxxxx  RAM Bank 0
11182  f7f5   xx01xxxx  Internal Function ROM  $c000-$ffff
11183  f7f5   xxxx01xx  Internal Function ROM  $8000-$bfff
11184  f7f5   xxxxxx1x  RAM Bank 0             $4000-$7fff
11185  f7f5   xxxxxxx0  I/O-Chips              $d000-$dfff
11186  f7f5
11187  f7f5  56                .byte %01010110  dsgl. für RAM Bank 1
11188  f7f6  96                .byte %10010110  dsgl. für RAM Bank 2 (sonst 0)
11189  f7f7  d6                .byte %11010110  dsgl. für RAM Bank 3 (sonst 1)
11190  f7f8
```

```
11191  f7f8  2a              .byte %00101010
11192  f7f9
11193  f7f9   00xxxxxx  RAM Bank 0
11194  f7f9   xx10xxxx  External Function ROM  $c000-$ffff
11195  f7f9   xxxx10xx  External Function ROM  $8000-$bfff
11196  f7f9   xxxxxx1x  RAM Bank 0             $4000-$7fff
11197  f7f9   xxxxxxx0  I/O-Chips              $d000-$dfff
11198  f7f9
11199  f7f9  6a              .byte %01101010  dsgl. für RAM Bank 1
11200  f7fa  aa              .byte %10101010  dsgl. für RAM Bank 2 (sonst 0)
11201  f7fb  ea              .byte %11101010  dsgl. für RAM Bank 3 (sonst 1)
11202  f7fc
11203  f7fc  06              .byte %00000110
11204  f7fd
11205  f7fd   00xxxxxx  RAM Bank 0
11206  f7fd   xx00xxxx  Kernal ROM             $c000-$ffff
11207  f7fd   xxxx01xx  Internal Function ROM  $8000-$bfff
11208  f7fd   xxxxxx1x  RAM Bank 0             $4000-$7fff
11209  f7fd   xxxxxxx0  I/O-Chips              $d000-$dfff
11210  f7fd
11211  f7fd  0a              .byte %00001010
11212  f7fe
11213  f7fe   00xxxxxx  RAM Bank 0
11214  f7fe   xx00xxxx  Kernal ROM             $c000-$ffff
11215  f7fe   xxxx10xx  External Function ROM  $8000-$bfff
11216  f7fe   xxxxxx1x  RAM Bank 0             $4000-$7fff
11217  f7fe   xxxxxxx0  I/O-Chips              $d000-$dfff
11218  f7fe
11219  f7fe  01              .byte %00000001
11220  f7ff
11221  f7ff   00xxxxxx  RAM Bank 0
11222  f7ff   xx00xxxx  Kernal ROM      $c000-$ffff
11223  f7ff   xxxx00xx  Teil BASIC ROM  $8000-$bfff
11224  f7ff   xxxxxx0x  Teil BASIC ROM  $4000-$7fff
11225  f7ff   xxxxxxx1  ROM-Char.gen.   $d000-$dfff
11226  f7ff
11227  f7ff  00              .byte %00000000
11228  f800
11229  f800   00xxxxxx  RAM Bank 0
11230  f800   xx00xxxx  Kernal ROM      $c000-$ffff
11231  f800   xxxx00xx  Teil BASIC ROM  $8000-$bfff
11232  f800   xxxxxx0x  Teil BASIC ROM  $4000-$7fff
11233  f800   xxxxxxx0  I/O-Chips       $d000-$dfff
11234  f800
11235  f800
11236  f800  ===> LDA (LDAVEC),y, Bank xr (RAM $02a2) <===
11237  f800
11238  f800  ad 00 ff   fetcho lda mmucr        MMU 8722 Config. Reg. (Kopie)
11239  f803  8e 00 ff          stx mmucr        MMU 8722 Config. Reg. (Kopie)
11240  f806
11241  f806  aa                 tax
11242  f807  b1 ff              lda (dummy),y    Dummy-Adr. für ind. RAM-Routinen
11243  f809  8e 00 ff          stx mmucr        MMU 8722 Config. Reg. (Kopie)
11244  f80c
11245  f80c  60                 rts
11246  f80d
11247  f80d
```

```
11248  f80d  ===> STA (STAVEC),y, Bank xr (RAM $02af) <===
11249  f80d
11250  f80d  48         stasho pha
11251  f80e
11252  f80e  ad 00 ff          lda mmucr        MMU 8722 Config. Reg. (Kopie)
11253  f811  8e 00 ff          stx mmucr        MMU 8722 Config. Reg. (Kopie)
11254  f814
11255  f814  aa                 tax              akt. Config.
11256  f815  68                 pla
11257  f816  91 ff              sta (dummy),y    Dummy-Adr. für ind. RAM-Routinen
11258  f818  8e 00 ff          stx mmucr        ,akt. Config
11259  f81b
11260  f81b  60                 rts
11261  f81c
11262  f81c
11263  f81c  ===> CMP (CMPVEC),y, Bank xr (RAM $02be) <===
11264  f81c
11265  f81c  48         cmparo pha
11266  f81d
11267  f81d  ad 00 ff          lda mmucr        MMU 8722 Config. Reg. (Kopie)
11268  f820  8e 00 ff          stx mmucr        MMU 8722 Config. Reg. (Kopie)
11269  f823
11270  f823  aa                 tax
11271  f824  68                 pla
11272  f825  d1 ff              cmp (dummy),y    Dummy-Adr. für ind. RAM-Routinen
11273  f827  8e 00 ff          stx mmucr        MMU 8722 Config. Reg. (Kopie)
11274  f82a
11275  f82a  60                 rts
11276  f82b
11277  f82b
11278  f82b  ===> JSR, Bank xr (RAM $02cd) <===
11279  f82b
11280  f82b  20 e3 02   jsrfao  jsr jmpfar        JMP, Bank xr
11281  f82e  85 06              sta areg         8502 CPU ac
11282  f830  86 07              stx xreg         8502 CPU xr
11283  f832  84 08              sty yreg         8502 CPU yr
11284  f834
11285  f834  08                 php
11286  f835  68                 pla
11287  f836  85 05              sta sreg         8502 CPU sr
11288  f838
11289  f838  ba                 tsx
11290  f839  86 09              stx stkptr       8502 CPU sp
11291  f83b
11292  f83b  a9 00              lda #%00000000
11293  f83d  8d 00 ff          sta mmucr        MMU 8722 Config. Reg. (Kopie)
11294  f840
11295  f840  60                 rts
11296  f841
11297  f841
11298  f841  ===> JMP, Bank xr (RAM $02e3) <===
11299  f841
11300  f841  a2 00      jmpfao ldx #0
11301  f843  b5 03      jpfao1 lda pchi,x        8502 CPU PC hi
11302  f845  48                pha
11303  f846
11304  f846  e8                inx
11305  f847  e0 03              cpx #3
```

```
11306  f849  90 f8           bcc jpfao1
11307  f84b
11308  f84b  a6 02           ldx bank          Banknummer
11309  f84d  20 6b ff        jsr kgtcfg        MMU-Wert zu Bank in xr nach ac
                                               (Sprungv.)
11310  f850  8d 00 ff        sta mmucr         MMU 8722 Config. Reg. (Kopie)
11311  f853
11312  f853  a5 06           lda areg          8502 CPU ac
11313  f855  a6 07           ldx xreg          8502 CPU xr
11314  f857  a4 08           ldy yreg          8502 CPU yr
11315  f859  40              rti
11316  f85a
11317  f85a
11318  f85a
11319  f85a  ===> Abschluß DMA Routine (RAM $03f0) <===
11320  f85a
11321  f85a  ae 00 ff  dodmao ldx mmucr         MMU 8722 Config. Reg. (Kopie)
11322  f85d  8c 01 df         sty dmacmd
11323  f860  8d 00 ff         sta mmucr         MMU 8722 Config. Reg. (Kopie)
11324  f863  8e 00 ff         stx mmucr         MMU 8722 Config. Reg. (Kopie)
11325  f866
11326  f866  60               rts
11327  f867
11328  f867
11329  f867  ===> Aufruf Kaltstartroutinen Funktionskarte <===
11330  f867
11331  f867  78        phoenx sei
11332  f868  a2 03            ldx #3            Banks 3-0
11333  f86a  8e c0 0a         stx curbnk        akt. Bank Fkt.tasten
11334  f86d
11335  f86d  ae c0 0a  boot01 ldx curbnk        akt. Bank Fkt.tasten
11336  f870  bd c1 0a         lda pat,x         Tab. der Flags init. ROM-Module
11337  f873  f0 11            beq boot02
11338  f875
11339  f875  a0 00            ldy #$00
11340  f877  bd bc e2         lda fkaddr,x      Adreßtab. ROM-ID's
11341  f87a  85 03            sta pchi          8502 CPU PC hi
11342  f87c  84 04            sty pclo          8502 CPU PC lo
11343  f87e
11344  f87e  bd c0 e2         lda fkbank,x      Bank-Tab. ROM-ID's
11345  f881  85 02            sta bank          Banknummer
11346  f883
11347  f883  20 cd 02         jsr jsrfar        Bank initialisieren
11348  f886  ce c0 0a  boot02 dec curbnk        akt. Bank Fkt.tasten
11349  f889  10 e2            bpl boot01
11350  f88b
11351  f88b  58               cli
11352  f88c  a2 08            ldx #8            akt. Gerätenummer
11353  f88e  a9 30            lda #'0'
11354  f890
11355  f890
11356  f890  ===> Boot-Programm von Diskette laden <===
11357  f890
11358  f890  85 bf     bootcl sta mych          Puffer für serielles Zeichen
11359  f892  86 ba            stx fa            akt. Gerätenummer
11360  f894
11361  f894  8a               txa               akt. Gerätenummer
11362  f895  20 3d f2         jsr closal        alle Files von Gerät in ac schließen
```

```
11363  f898
11364  f898  a2 00           ldx #0
11365  f89a  86 9f           stx ptr2
11366  f89c  86 c2           stx stah        addrh Startadresse save
11367  f89e
11368  f89e  e8              inx
11369  f89f  86 c1           stx stal        addrl Startadresse save
11370  f8a1
11371  f8a1  ===> Warteschleife ca. 0.3 s <===
11372  f8a1
11373  f8a1  c8        boot04 iny
11374  f8a2  d0 fd            bne boot04
11375  f8a4  e8               inx
11376  f8a5  d0 fa            bne boot04
11377  f8a7
11378  f8a7  a2 0c            ldx #12
11379  f8a9
11380  f8a9
11381  f8a9  ===> block-read Befehl in Stapel übertragen <===
11382  f8a9
11383  f8a9  bd 08 fa   boot05 lda blkdat,x    block-read Befehl für Floppy
11384  f8ac  9d 00 01         sta stack,x     8502 CPU-Stapel
11385  f8af
11386  f8af  ca               dex
11387  f8b0  10 f7            bpl boot05      block-read Befehl in Stapel
                                              übertragen
11388  f8b2
11389  f8b2  a5 bf            lda mych        Puffer für serielles Zeichen
11390  f8b4  8d 06 01         sta stack+6     8502 CPU-Stapel
11391  f8b7
11392  f8b7  a9 00            lda #0
11393  f8b9  a2 0f            ldx #15
11394  f8bb  20 3f f7         jsr setbnk      Bank für load/sav/verify setzen
11395  f8be
11396  f8be  a9 01            lda #1
11397  f8c0  a2 15            ldx #<dskint    Floppy-Befehl 'initialisierung'
11398  f8c2  a0 fa            ldy #>dskint    Floppy-Befehl 'initialisierung'
11399  f8c4  20 31 f7         jsr setnam      fnlen, fnadr/+1 aus ac, xr, yr
                                              setzen
11400  f8c7
11401  f8c7  a9 00            lda #0
11402  f8c9  a0 0f            ldy #15
11403  f8cb  a6 ba            ldx fa          akt. Gerätenummer
11404  f8cd  20 38 f7         jsr setlfs      la, fa, sa aus ac, xr, yr setzen
11405  f8d0  20 c0 ff         jsr kopen       log. File üb. Vector öffnen, Ausgabe
                                              fn (sprungv.)
11406  f8d3  b0 16            bcs boot06
11407  f8d5
11408  f8d5  a9 01            lda #1
11409  f8d7  a2 16            ldx #<dskbfr    Floppy-Befehl 'direktkanal eröffnen'
11410  f8d9  a0 fa            ldy #>dskbfr    Floppy-Befehl 'direktkanal eröffnen'
11411  f8db  20 31 f7         jsr setnam      fnlen, fnadr/+1 aus ac, xr, yr
                                              setzen
11412  f8de
11413  f8de  a9 0d            lda #13
11414  f8e0  a8               tay
11415  f8e1  a6 ba            ldx fa          akt. Gerätenummer
11416  f8e3  20 38 f7         jsr setlfs      la, fa, sa aus ac, xr, yr setzen
```

```
11417  f8e6
11418  f8e6  20 c0 ff          jsr kopen        log. File üb. Vector öffnen, Ausgabe
                                                fn (sprungv.)
11419  f8e9  90 03             bcc boot07
11420  f8eb
11421  f8eb  4c 8b f9   boot06 jmp botcls
11422  f8ee
11423  f8ee
11424  f8ee  a9 00      boot07 lda #<tbuffr      Kassettenpuffer
11425  f8f0  a0 0b             ldy #>tbuffr     Kassettenpuffer
11426  f8f2  85 ac             sta sal          addrl für load/save
11427  f8f4  84 ad             sty sah          addrh für load/save
11428  f8f6
11429  f8f6  20 d5 f9          jsr blkred       Block von Diskette indirekt über
                                                'sal' speichern
11430  f8f9
11431  f8f9  a2 00             ldx #0
11432  f8fb
11433  f8fb
11434  f8fb  ===> Test auf Boot-Sector (CBM) <===
11435  f8fb
11436  f8fb  bd 00 0b   boot08 lda tbuffr,x     Kassettenpuffer
11437  f8fe  dd c4 e2          cmp cbmkey,x     Text ROM-ID
11438  f901  d0 e8             bne boot06
11439  f903
11440  f903  e8                inx
11441  f904  e0 03             cpx #3
11442  f906  90 f3             bcc boot08       Test auf Boot-Sector (CBM)
11443  f908
11444  f908  20 17 fa          jsr primm        folg. Text ausgeben und überspringen
11445  f90b
11446  f90b  0d 42 4f          .byte cr,'booting ', eom
11447  f90e  4f 54 49
11448  f911  4e 47 20
11449  f914  00
11450  f915
11451  f915  bd 00 0b   boot09 lda tbuffr,x     Kassettenpuffer
11452  f918  95 a9             sta rinone,x     RS 232 Flag für Startbit
11453  f91a
11454  f91a  e8                inx
11455  f91b  e0 07             cpx #7
11456  f91d  90 f6             bcc boot09
11457  f91f
11458  f91f
11459  f91f  ===> Boot-Filenamen ausgeben <===
11460  f91f
11461  f91f  bd 00 0b   boot10 lda tbuffr,x     Kassettenpuffer
11462  f922  f0 06             beq boot11
11463  f924
11464  f924  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt. Kanal
                                                (Sprungv.)
11465  f927  e8                inx
11466  f928  d0 f5             bne boot10       Boot-Filenamen ausgeben
11467  f92a
11468  f92a  86 9e      boot11 stx ptr1
11469  f92c
11470  f92c  20 17 fa          jsr primm        folg. Text ausgeben und überspringen
11471  f92f
```

```
11472  f92f  2e 2e 2e          .byte '...', cr, eom
11473  f932  0d 00
11474  f934
11475  f934  a5 ae             lda eal          addrl Ende load/save
11476  f936  85 c6             sta ba           akt. Bank für Speicher-Operationen
11477  f938
11478  f938  a5 af      boot12 lda eah          addrh Ende load/save
11479  f93a  f0 09             beq boot13
11480  f93c
11481  f93c  c6 af             dec eah          addrh Ende load/save
11482  f93e  20 b3 f9          jsr blknex
11483  f941  e6 ad             inc sah          addrh für load/save
11484  f943  d0 f3             bne boot12
11485  f945
11486  f945  20 8b f9  boot13  jsr botcls
11487  f948  a6 9e             ldx ptr1
11488  f94a  2c                .byte skip2
11489  f94b
11490  f94b  e6 9f      boot14 inc ptr2
11491  f94d  e8                inx
11492  f94e  bd 00 0b          lda tbuffr,x     Kassettenpuffer
11493  f951  d0 f8             bne boot14
11494  f953
11495  f953  e8                inx
11496  f954  86 04             stx pclo         8502 CPU PC lo
11497  f956
11498  f956  a6 9e             ldx ptr1
11499  f958  a9 3a             lda #':'
11500  f95a  9d 00 0b          sta tbuffr,x     Kassettenpuffer
11501  f95d
11502  f95d  ca                dex
11503  f95e  a5 bf             lda mych         Puffer für serielles Zeichen
11504  f960  9d 00 0b          sta tbuffr,x     Kassettenpuffer
11505  f963  86 9e             stx ptr1
11506  f965
11507  f965  a6 9f             ldx ptr2
11508  f967  f0 15             beq boot15
11509  f969
11510  f969  e8                inx
11511  f96a  e8                inx
11512  f96b  8a                txa
11513  f96c
11514  f96c  a6 9e             ldx ptr1
11515  f96e  a0 0b             ldy #>tbuffr     Kassettenpuffer
11516  f970  20 31 f7          jsr setnam       fnlen, fnadr/+1 aus ac, xr, yr
                                                 setzen
11517  f973
11518  f973  a9 00             lda #0
11519  f975  aa                tax
11520  f976  20 3f f7          jsr setbnk       Bank für load/sav/verify setzen
11521  f979
11522  f979  a9 00             lda #0
11523  f97b  20 69 f2          jsr load
11524  f97e  a9 0b      boot15 lda #>tbuffr     Kassettenpuffer
11525  f980  85 03             sta pchi         8502 CPU PC hi
11526  f982
11527  f982  a9 0f             lda #15
11528  f984  85 02             sta bank         Banknummer
```

```
11529  f986
11530  f986  20 cd 02        jsr jsrfar      JSR, Bank xr
11531  f989  18              clc
11532  f98a  60              rts
11533  f98b
11534  f98b
11535  f98b  08       botcls php
11536  f98c  48              pha
11537  f98d
11538  f98d  20 cc ff        jsr kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                             schließen (Sprungv.)
11539  f990
11540  f990  a9 0d           lda #13
11541  f992  18              clc
11542  f993  20 c3 ff        jsr kclose      log. File üb. Vector schließen
                                             (Sprungv.)
11543  f996
11544  f996  a2 00           ldx #0
11545  f998  20 c9 ff        jsr kckout      Ausgabe-Kanal üb. Vector öffnen
                                             (Sprungv.)
11546  f99b  b0 0a           bcs boot17
11547  f99d
11548  f99d  a9 55           lda #'u'
11549  f99f  20 d2 ff        jsr kbsout      Ausgabe ac üb. Vector auf akt. Kanal
                                             (Sprungv.)
11550  f9a2
11551  f9a2  a9 49           lda #'i'
11552  f9a4  20 d2 ff        jsr kbsout      Ausgabe ac üb. Vector auf akt. Kanal
                                             (Sprungv.)
11553  f9a7
11554  f9a7  20 cc ff boot17 jsr kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                             schließen (Sprungv.)
11555  f9aa  a9 00           lda #0
11556  f9ac  38              sec
11557  f9ad  20 c3 ff        jsr kclose      log. File üb. Vector schließen
                                             (Sprungv.)
11558  f9b0  68              pla
11559  f9b1  28              plp
11560  f9b2  60              rts
11561  f9b3
11562  f9b3
11563  f9b3  a6 c2    blknex ldx stah        addrh Startadresse save
11564  f9b5  e8              inx
11565  f9b6  e0 15           cpx #$15
11566  f9b8  90 04           bcc boot19
11567  f9ba
11568  f9ba  a2 00           ldx #$00
11569  f9bc  e6 c1           inc stal        addrl Startadresse save
11570  f9be  86 c2    boot19 stx stah        addrh Startadresse save
11571  f9c0
11572  f9c0  8a              txa
11573  f9c1  20 fb f9        jsr decasc      Umwandlung Hex nach ASCII, ac -> ac
                                             + xr
11574  f9c4  8d 00 01        sta stack       8502 CPU-Stapel
11575  f9c7
11576  f9c7  8e 01 01        stx stack+1     8502 CPU-Stapel
11577  f9ca
11578  f9ca  a5 c1           lda stal        addrl Startadresse save
```

```
11579  f9cc  20 fb f9          jsr decasc      Umwandlung Hex nach ASCII, ac -> ac
                                                + xr
11580  f9cf  8d 03 01          sta stack+3     8502 CPU-Stapel
11581  f9d2
11582  f9d2  8e 04 01          stx stack+4     8502 CPU-Stapel
11583  f9d5
11584  f9d5
11585  f9d5  ===> Block von Diskette indirekt über 'sal' speichern <===
11586  f9d5
11587  f9d5  a2 00       blkred ldx #0
11588  f9d7  20 c9 ff          jsr kckout      Ausgabe-Kanal üb. Vector öffnen
                                                (Sprungv.)
11589  f9da
11590  f9da  a2 0c             ldx #12
11591  f9dc  bd 00 01    redit1 lda stack,x     8502 CPU-Stapel
11592  f9df  20 d2 ff          jsr kbsout      Ausgabe ac üb. Vector auf akt. Kanal
                                                (Sprungv.)
11593  f9e2  ca                dex
11594  f9e3  10 f7             bpl redit1
11595  f9e5
11596  f9e5  20 cc ff          jsr kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                                schließen (Sprungv.)
11597  f9e8
11598  f9e8  a2 0d             ldx #13
11599  f9ea  20 c6 ff          jsr kchkin      Eingabekanal üb. Vector öffnen
                                                (Sprungv.)
11600  f9ed
11601  f9ed  a0 00             ldy #0
11602  f9ef  20 cf ff    redit2 jsr kbasin      Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
11603  f9f2  20 bc f7          jsr salsy       Byte indirekt über 'sal' speichern
11604  f9f5  c8                iny
11605  f9f6  d0 f7             bne redit2
11606  f9f8
11607  f9f8  4c cc ff          jmp kclrch      Ein-/Ausgabe-Kanäle üb. Vector
                                                schließen (Sprungv.)
11608  f9fb
11609  f9fb
11610  f9fb                    .end
11611  f9fb                    .lib c128.kernal4
11612  f9fb
11614  f9fb
11615  f9fb
11616  f9fb  ===> Umwandlung Hex nach ASCII, ac -> ac + xr <===
11617  f9fb
11618  f9fb  a2 30       decasc ldx #$30
11619  f9fd  38                sec
11620  f9fe  e9 0a       decas1 sbc #$0a
11621  fa00  90 03             bcc decas2
11622  fa02
11623  fa02  e8                inx
11624  fa03  b0 f9             bcs decas1
11625  fa05
11626  fa05  69 3a       decas2 adc #$3a
11627  fa07  60                rts
11628  fa08
11629  fa08
```

```
11630  fa08  ===> block-read Befehl für Floppy <===
11631  fa08
11632  fa08  30 30 20   blkdat .byte '00 10 0 31:1u'
11633  fa0b  31 30 20
11634  fa0e  30 20 33
11635  fa11  31 3a 31
11636  fa14  55
11637  fa15
11638  fa15
11639  fa15  ===> Floppy-Befehl 'initialisierung' <===
11640  fa15
11641  fa15  49         dskint .byte 'i'
11642  fa16
11643  fa16
11644  fa16  ===> Floppy-Befehl 'direktkanal eröffnen' <===
11645  fa16
11646  fa16  23         dskbfr .byte '#'
11647  fa17
11648  fa17
11649  fa17  ===> folg. Text ausgeben und überspringen <===
11650  fa17
11651  fa17  48         primm  pha
11652  fa18
11653  fa18  8a                txa
11654  fa19  48                pha
11655  fa1a
11656  fa1a  98                tya
11657  fa1b  48                pha
11658  fa1c
11659  fa1c  a0 00             ldy #0
11660  fa1e  ba         dirms1 tsx
11661  fa1f  fe 04 01          inc stack+4,x    8502 CPU-Stapel
11662  fa22  d0 03             bne dirms2
11663  fa24
11664  fa24  fe 05 01          inc stack+5,x    8502 CPU-Stapel
11665  fa27  bd 04 01   dirms2 lda stack+4,x    8502 CPU-Stapel
11666  fa2a  85 ce             sta imparm       Hilfsindex Ausgabe Text aus PGM-Code
11667  fa2c
11668  fa2c  bd 05 01          lda stack+5,x    8502 CPU-Stapel
11669  fa2f  85 cf             sta imparm+1     Hilfsindex Ausgabe Text aus PGM-Code
11670  fa31
11671  fa31  b1 ce             lda (imparm),y   Hilfsindex Ausgabe Text aus PGM-Code
11672  fa33  f0 05             beq dirms3
11673  fa35
11674  fa35  20 d2 ff          jsr kbsout       Ausgabe ac üb. Vector auf akt. Kanal
                                                (Sprungv.)
11675  fa38  90 e4             bcc dirms1
11676  fa3a
11677  fa3a  68         dirms3 pla
11678  fa3b  a8                tay
11679  fa3c  68                pla
11680  fa3d  aa                tax
11681  fa3e  68                pla
11682  fa3f  60                rts
11683  fa40
11684  fa40
```

```
11685  fa40  ===> NMI-Routine <===
11686  fa40
11687  fa40  d8         nnmi   cld
11688  fa41  a9 7f             lda #%01111111
11689  fa43  8d 0d dd          sta d2icr        CIA2 Interrupt Control-Register
11690  fa46
11691  fa46  ac 0d dd          ldy d2icr        CIA2 Interrupt Control-Register
11692  fa49  30 14             bmi nnmi1
11693  fa4b
11694  fa4b  20 3d f6          jsr ud60         Abfrage Stopp/Shift-Taste/Spalte7
11695  fa4e  20 e1 ff          jsr kstop        Stopp-Taste üb. Vector abfragen
                                                (Sprungv.)
11696  fa51  d0 0c             bne nnmi1
11697  fa53
11698  fa53  20 56 e0          jsr restor       Standard I/O-Vectoren einrichten
11699  fa56  20 09 e1          jsr ioinit       I/O-Bausteine initialisieren
11700  fa59  20 00 c0          jsr jpcint       Video-Chips/Editor init.
11701  fa5c  6c 00 0a          jmp (sysvec)     Vector (BASIC-) Warmstart ($4003)
11702  fa5f
11703  fa5f
11704  fa5f  20 05 e8   nnmi1  jsr nmi232       RS 232-NMI bearbeiten
11705  fa62  4c 33 ff          jmp prend
11706  fa65
11707  fa65
11708  fa65  ===> IRQ-Routine <===
11709  fa65
11710  fa65  d8         nirq   cld
11711  fa66  20 24 c0          jsr jrsirq       Raster-Interrupt bearbeiten
11712  fa69  90 12             bcc nirq1
11713  fa6b
11714  fa6b  20 f8 f5          jsr udtim        System-Uhr aktualisieren
11715  fa6e  20 d0 ee          jsr tapirq       Abfrage gedrückte Kassetten-Taste
11716  fa71  ad 0d dc          lda d1icr        CIA1 Interrupt Control-Register
11717  fa74  ad 04 0a          lda initst       Flag Status nach RESET, BASIC-IRQ
11718  fa77  4a                lsr a
11719  fa78  90 03             bcc nirq1
11720  fa7a
11721  fa7a  20 06 40          jsr birqvc       BASIC-Interrupt bearbeiten
11722  fa7d  4c 33 ff   nirq1  jmp prend
11723  fa80
11724  fa80
11725  fa80  ===> Tastaturtabelle ASCII normal <===
11726  fa80
11727  fa80  14 0d 1d   mode1  .byte $14, $0d, $1d, $88, $85, $86
11728  fa83  88 85 86
11729  fa86  87 11 33          .byte $87, $11, $33, $57, $41, $34
11730  fa89  57 41 34
11731  fa8c  5a 53 45          .byte $5a, $53, $45, $01, $35, $52
11732  fa8f  01 35 52
11733  fa92  44 36 43          .byte $44, $36, $43, $46, $54, $58
11734  fa95  46 54 58
11735  fa98  37 59 47          .byte $37, $59, $47, $38, $42, $48
11736  fa9b  38 42 48
11737  fa9e  55 56 39          .byte $55, $56, $39, $49, $4a, $30
11738  faa1  49 4a 30
11739  faa4  4d 4b 4f          .byte $4d, $4b, $4f, $4e, $2b, $50
11740  faa7  4e 2b 50
11741  faaa  4c 2d 2e          .byte $4c, $2d, $2e, $3a, $40, $2c
```

```
11742  faad  3a 40 2c
11743  fab0  5c 2a 3b          .byte $5c, $2a, $3b, $13, $01, $3d
11744  fab3  13 01 3d
11745  fab6  5e 2f 31          .byte $5e, $2f, $31, $5f, $04, $32
11746  fab9  5f 04 32
11747  fabc  20 02 51          .byte $20, $02, $51, $03, $84, $38
11748  fabf  03 84 38
11749  fac2  35 09 32          .byte $35, $09, $32, $34, $37, $31
11750  fac5  34 37 31
11751  fac8  1b 2b 2d          .byte $1b, $2b, $2d, $0a, $0d, $36
11752  facb  0a 0d 36
11753  face  39 33 08          .byte $39, $33, $08, $30, $2e, $91
11754  fad1  30 2e 91
11755  fad4  11 9d 1d          .byte $11, $9d, $1d, $ff, $ff
11756  fad7  ff ff
11757  fad9
11758  fad9
11759  fad9  ===> Tastaturtabelle ASCII mit Shift <===
11760  fad9
11761  fad9  94 8d 9d   mode2  .byte $94, $8d, $9d, $8c, $89, $8a
11762  fadc  8c 89 8a
11763  fadf  8b 91 23          .byte $8b, $91, $23, $d7, $c1, $24
11764  fae2  d7 c1 24
11765  fae5  da d3 c5          .byte $da, $d3, $c5, $01, $25, $d2
11766  fae8  01 25 d2
11767  faeb  c4 26 c3          .byte $c4, $26, $c3, $c6, $d4, $d8
11768  faee  c6 d4 d8
11769  faf1  27 d9 c7          .byte $27, $d9, $c7, $28, $c2, $c8
11770  faf4  28 c2 c8
11771  faf7  d5 d6 29          .byte $d5, $d6, $29, $c9, $ca, $30
11772  fafa  c9 ca 30
11773  fafd  cd cb cf          .byte $cd, $cb, $cf, $ce, $db, $d0
11774  fb00  ce db d0
11775  fb03  cc dd 3e          .byte $cc, $dd, $3e, $5b, $ba, $3c
11776  fb06  5b ba 3c
11777  fb09  a9 c0 5d          .byte $a9, $c0, $5d, $93, $01, $3d
11778  fb0c  93 01 3d
11779  fb0f  de 3f 21          .byte $de, $3f, $21, $5f, $04, $22
11780  fb12  5f 04 22
11781  fb15  a0 02 d1          .byte $a0, $02, $d1, $83, $84, $38
11782  fb18  83 84 38
11783  fb1b  35 18 32          .byte $35, $18, $32, $34, $37, $31
11784  fb1e  34 37 31
11785  fb21  1b 2b 2d          .byte $1b, $2b, $2d, $0a, $8d, $36
11786  fb24  0a 8d 36
11787  fb27  39 33 08          .byte $39, $33, $08, $30, $2e, $91
11788  fb2a  30 2e 91
11789  fb2d  11 9d 1d          .byte $11, $9d, $1d, $ff, $ff
11790  fb30  ff ff
11791  fb32
11792  fb32
11793  fb32  ===> Tastaturtabelle ASCII mit CBM-taste <===
11794  fb32
11795  fb32  94 8d 9d   mode3  .byte $94, $8d, $9d, $8c, $89, $8a
11796  fb35  8c 89 8a
11797  fb38  8b 91 96          .byte $8b, $91, $96, $b3, $b0, $97
11798  fb3b  b3 b0 97
11799  fb3e  ad ae b1          .byte $ad, $ae, $b1, $01, $98, $b2
```

```
11800  fb41  01 98 b2
11801  fb44  ac 99 bc          .byte $ac, $99, $bc, $bb, $a3, $bd
11802  fb47  bb a3 bd
11803  fb4a  9a b7 a5          .byte $9a, $b7, $a5, $9b, $bf, $b4
11804  fb4d  9b bf b4
11805  fb50  b8 be 29          .byte $b8, $be, $29, $a2, $b5, $30
11806  fb53  a2 b5 30
11807  fb56  a7 a1 b9          .byte $a7, $a1, $b9, $aa, $a6, $af
11808  fb59  aa a6 af
11809  fb5c  b6 dc 3e          .byte $b6, $dc, $3e, $5b, $a4, $3c
11810  fb5f  5b a4 3c
11811  fb62  a8 df 5d          .byte $a8, $df, $5d, $93, $01, $3d
11812  fb65  93 01 3d
11813  fb68  de 3f 81          .byte $de, $3f, $81, $5f, $04, $95
11814  fb6b  5f 04 95
11815  fb6e  a0 02 ab          .byte $a0, $02, $ab, $03, $84, $38
11816  fb71  03 84 38
11817  fb74  35 18 32          .byte $35, $18, $32, $34, $37, $31
11818  fb77  34 37 31
11819  fb7a  1b 2b 2d          .byte $1b, $2b, $2d, $0a, $8d, $36
11820  fb7d  0a 8d 36
11821  fb80  39 33 08          .byte $39, $33, $08, $30, $2e, $91
11822  fb83  30 2e 91
11823  fb86  11 9d 1d          .byte $11, $9d, $1d, $ff, $ff
11824  fb89  ff ff
11825  fb8b
11826  fb8b
11827  fb8b  ===> Tastaturtabelle ASCII mit CTRL-taste <===
11828  fb8b
11829  fb8b  ff ff ff    mode4 .byte $ff, $ff, $ff, $ff, $ff, $ff
11830  fb8e  ff ff ff
11831  fb91  ff ff 1c          .byte $ff, $ff, $1c, $17, $01, $9f
11832  fb94  17 01 9f
11833  fb97  1a 13 05          .byte $1a, $13, $05, $ff, $9c, $12
11834  fb9a  ff 9c 12
11835  fb9d  04 1e 03          .byte $04, $1e, $03, $06, $14, $18
11836  fba0  06 14 18
11837  fba3  1f 19 07          .byte $1f, $19, $07, $9e, $02, $08
11838  fba6  9e 02 08
11839  fba9  15 16 12          .byte $15, $16, $12, $09, $0a, $92
11840  fbac  09 0a 92
11841  fbaf  0d 0b 0f          .byte $0d, $0b, $0f, $0e, $ff, $10
11842  fbb2  0e ff 10
11843  fbb5  0c ff ff          .byte $0c, $ff, $ff, $1b, $00, $ff
11844  fbb8  1b 00 ff
11845  fbbb  1c ff 1d          .byte $1c, $ff, $1d, $ff, $ff, $1f
11846  fbbe  ff ff 1f
11847  fbc1  1e ff 90          .byte $1e, $ff, $90, $06, $ff, $05
11848  fbc4  06 ff 05
11849  fbc7  ff ff 11          .byte $ff, $ff, $11, $ff, $84, $38
11850  fbca  ff 84 38
11851  fbcd  35 18 32          .byte $35, $18, $32, $34, $37, $31
11852  fbd0  34 37 31
11853  fbd3  1b 2b 2d          .byte $1b, $2b, $2d, $0a, $8d, $36
11854  fbd6  0a 8d 36
11855  fbd9  39 33 08          .byte $39, $33, $08, $30, $2e, $91
11856  fbdc  30 2e 91
11857  fbdf  11 9d 1d          .byte $11, $9d, $1d, $ff, $ff
```

```
11858  fbe2  ff ff
11859  fbe4
11860  fbe4
11861  fbe4  ===> Tastaturtabelle ASCII CAPS LOCK <===
11862  fbe4
11863  fbe4  14 0d 1d   mode5  .byte $14, $0d, $1d, $88, $85, $86
11864  fbe7  88 85 86
11865  fbea  87 11 33          .byte $87, $11, $33, $d7, $c1, $34
11866  fbed  d7 c1 34
11867  fbf0  da d3 c5          .byte $da, $d3, $c5, $01, $35, $d2
11868  fbf3  01 35 d2
11869  fbf6  c4 36 c3          .byte $c4, $36, $c3, $c6, $d4, $d8
11870  fbf9  c6 d4 d8
11871  fbfc  37 d9 c7          .byte $37, $d9, $c7, $38, $c2, $c8
11872  fbff  38 c2 c8
11873  fc02  d5 d6 39          .byte $d5, $d6, $39, $c9, $ca, $30
11874  fc05  c9 ca 30
11875  fc08  cd cb cf          .byte $cd, $cb, $cf, $ce, $2b, $d0
11876  fc0b  ce 2b d0
11877  fc0e  cc 2d 2e          .byte $cc, $2d, $2e, $3a, $40, $2c
11878  fc11  3a 40 2c
11879  fc14  5c 2a 3b          .byte $5c, $2a, $3b, $13, $01, $3d
11880  fc17  13 01 3d
11881  fc1a  5e 2f 31          .byte $5e, $2f, $31, $5f, $04, $32
11882  fc1d  5f 04 32
11883  fc20  20 02 51          .byte $20, $02, $51, $03, $84, $38
11884  fc23  03 84 38
11885  fc26  35 09 32          .byte $35, $09, $32, $34, $37, $31
11886  fc29  34 37 31
11887  fc2c  1b 2b 2d          .byte $1b, $2b, $2d, $0a, $0d, $36
11888  fc2f  0a 0d 36
11889  fc32  39 33 08          .byte $39, $33, $08, $30, $2e, $91
11890  fc35  30 2e 91
11891  fc38  11 9d 1d          .byte $11, $9d, $1d
11892  fc3b
11893  fc3b
11894  fc3b                    .end
11895  fc3b                    .lib c128.keyboard3
11896  fc3b
```

4.1.16 DIN-Tastatur-Abfrage

```
11898  fc3b
11899  fc3b
11900  fc3b                    *=$fc80
11901  fc80
11902  fc80
11903  fc80  ===> Flick-Routine DIN-Tastatur <===
11904  fc80
11905  fc80  8d c5 0a   patch  sta dinflg       Flag Akzente & ASCII/DIN-Modus
11906  fc83
11907  fc83   7.......  Umschaltung durch CAPS-LOCK-Taste:
11908  fc83   0xxxxxxx  ASCII / DIN-Tastatur
11909  fc83   1xxxxxxx  normal ASCII / CAPS-LOCK ASCII
11910  fc83             oder DIN / DIN ohne Akzente
11911  fc83
```

```
11912  fc83   .....210  Flag letzte gedrückte Taste:
11913  fc83   xxxxx000  kein Akzent
11914  fc83   xxxxx001  Accent aigu
11915  fc83   xxxxx010  Accent grave
11916  fc83   xxxxx100  Accent circonflexe
11917  fc83   xxxxx101  Reserve (0)
11918  fc83   xxxxx110  Reserve (0)
11919  fc83
11920  fc83   8d 18 d4          sta sidvol       Lautstärke = 0
11921  fc86
11922  fc86   60                rts
11923  fc87
11924  fc87
11925  fc87   ===> Tastatur-Abfrage (foreign key) <===
11926  fc87
11927  fc87   2c c5 0a   scnfky bit dinflg       Flag Akzente & ASCII/DIN-Modus
11928  fc8a
11929  fc8a    1xxxxxxx  normal ASCII / CAPS-LOCK ASCII
11930  fc8a              oder DIN / DIN ohne Akzente
11931  fc8a
11932  fc8a   30 37             bmi scnfk4       ASCII CAPS-LOCK oder DIN ohne
                                                 Akzente
11933  fc8c
11934  fc8c
11935  fc8c   ===> Test ASCII / DIN-Taste <===
11936  fc8c
11937  fc8c   a5 d3             lda shflag       Flag für Shift/CTRL/CBM-Taste
11938  fc8e   29 10             and #%00010000
11939  fc90
11940  fc90    xxx1xxxx  CAPS LOCK gedrückt
11941  fc90
11942  fc90   f0 0d             beq scnfk1       nicht gedrückt -> ASCII-Modus
11943  fc92
11944  fc92   ===> Test DIN-Tabelle eingeschaltet <===
11945  fc92
11946  fc92   ad 3f 03          lda nrmtab+1     addrh der akt. Tast.tab. normal
11947  fc95   c9 fd             cmp #>nrmdin     addrh der Tast.tab. DIN normal
11948  fc97   f0 2a             beq scnfk4       schon eingeschaltet
11949  fc99
11950  fc99   ===> Umschalten auf DIN-Tastatur <===
11951  fc99
11952  fc99   a9 34             lda #<keydin     Adreßtabelle der Tasturtabellen
                                                 (DIN-Tastatur)
11953  fc9b   a0 fe             ldy #>keydin     Adreßtabelle der Tasturtabellen
                                                 (DIN-Tastatur)
11954  fc9d   d0 0b             bne scnfk2       immer
11955  fc9f
11956  fc9f
11957  fc9f   ===> Test ASCII-Tab. eingeschaltet <===
11958  fc9f
11959  fc9f   ad 3f 03   scnfk1 lda nrmtab+1     addrh der akt. Tast.tab. normal
11960  fca2   c9 fa             cmp #>mode1      addrh der Tast.tab. ASCII normal
11961  fca4   f0 1d             beq scnfk4       akt. Tastatur abfragen, in Puffer
                                                 schreiben
11962  fca6
11963  fca6   ===> Umschalten auf DIN-Tastatur <===
11964  fca6
11965  fca6   a9 6f             lda #<keycod     Adreßtab. der Tast.-Decodier-Tab.
```

```
11966  fca8  a0 c0            ldy #>keycod     Adreßtab. der Tast.-Decodier-Tab.
11967  fcaa
11968  fcaa
11969  fcaa  ===> Adreßtab. akt. Tastatur einschalten <===
11970  fcaa
11971  fcaa  85 cc      scnfk2 sta keytab       addrl Zeiger in Tast.-Dekodier-Tab.
11972  fcac  84 cd             sty keytab+1     addrh Zeiger in Tast.-Dekodier-Tab.
11973  fcae
11974  fcae  a0 0b             ldy #11
11975  fcb0  b1 cc      scnfk3 lda (keytab),y   addrl Zeiger in Tast.-Dekodier-Tab.
11976  fcb2  99 3e 03          sta nrmtab,y     addrl akt. Tast.tab. normal
11977  fcb5  88                dey
11978  fcb6  10 f8             bpl scnfk3
11979  fcb8
11980  fcb8  ===> DIN-Flag rücksetzen <===
11981  fcb8
11982  fcb8  c8                iny              =0
11983  fcb9  8c c5 0a          sty dinflg       Flag Akzente & ASCII/DIN-Modus
11984  fcbc
11985  fcbc   wird in jedem Tastatur-Interrupt gesetzt
11986  fcbc
11987  fcbc  ===> akt. Zeichengenrator in VDC laden <===
11988  fcbc
11989  fcbc  08                php
11990  fcbd  78                sei
11991  fcbe  20 0c ce          jsr init80       Char.gen. in VDC laden
11992  fcc1  28                plp
11993  fcc2  60                rts
11994  fcc3
11995  fcc3
11996  fcc3  4c 5d c5   scnfk4 jmp scnkey       Tastatur Abfrage
11997  fcc6
11998  fcc6
11999  fcc6  ===> Test auf ASCII/DIN-Tastatur, ind. nach keyput <===
12000  fcc6
12001  fcc6  ae 3f 03   chkdin ldx nrmtab+1     addrh akt. Tast.tab. normal
12002  fcc9  e0 fd             cpx #>nrmdin     Tastaturtabelle DIN normal
12003  fccb  d0 55             bne jkychk       nicht DIN-Tastatur
12004  fccd
12005  fccd  ae c5 0a          ldx dinflg       Flag Akzente & ASCII/DIN-Modus
12006  fcd0  30 50             bmi jkychk       keine Bearbeitung DIN-Tastatur
12007  fcd2  f0 1d             beq accent       letztes Zeichen war keine Akzent
12008  fcd4
12009  fcd4
12010  fcd4  ===> letztes Zeichen Akzent, Test auf akt. Zeichen <===
12011  fcd4
12012  fcd4  bc 45 fe          ldy stgref,x     Zeiger auf letztes Zeichen
12013  fcd7  ca                dex              erniedrigen
12014  fcd8
12015  fcd8
12016  fcd8  ===> Test Zeichen unter Akzent zulässig <===
12017  fcd8
12018  fcd8  88         chrasm dey              Zeiger in Bereich zul. Zeichen
12019  fcd9  48                pha              akt. Zeichen
12020  fcda
12021  fcda  ===> Test noch in Bereich <===
12022  fcda
12023  fcda  98                tya              Zeiger in Bereich
```

```
12024  fcdb  dd 45 fe          cmp stgref,x     Obergrenze Bereich davor
12025  fcde  68                pla              akt Zeichen
12026  fcdf  90 08             bcc chras1       nicht gefunden
12027  fce1
12028  fce1  d9 4a fe          cmp acchtb-1,y   Zeichen unter Akzent zulässig ?
12029  fce4  d0 f2             bne chrasm       nein, weitersuchen
12030  fce6
12031  fce6  b9 65 fe          lda accdtb-1,y   Ersatzzeichen mit Akzent
12032  fce9
12033  fce9
12034  fce9  ===> Steuerzeichen nach Akzent unterdrücken <===
12035  fce9
12036  fce9  48          chras1 pha
12037  fcea
12038  fcea  29 7f             and #%01111111   Shift-Bit löschen
12039  fcec  c9 20             cmp #$20         kleinstes ASCII-Zeichen
12040  fcee  68                pla
12041  fcef  90 23             bcc accrts       Steuerzeichen
12042  fcf1
12043  fcf1
12044  fcf1  ===> Test akt. Zeichen auf gült. Akzent <===
12045  fcf1
12046  fcf1  a2 05       accent ldx #5
12047  fcf3  dd 3f fe    acc010 cmp acctbl-1,x  Tab. der Tastencodes gültiger
                                                Akzente
12048  fcf6  f0 03             beq acc020
12049  fcf8
12050  fcf8  ca                dex
12051  fcf9  d0 f8             bne acc010
12052  fcfb
12053  fcfb  8e c5 0a    acc020 stx dinflg       Nummer Akzent (0-5)
12054  fcfe  e0 00             cpx #0
12055  fd00  f0 20             beq jkychk       akt. Zeichen kein Akzent
12056  fd02
12057  fd02  a8                tay              Tastaturcode für Akzent
12058  fd03  24 f6             bit insflg       AUTO-INSERT-Flag
12059  fd05  30 0d             bmi accrts       auto-insert-Modus unterdrücken
12060  fd07
12061  fd07  ===> Test fester / blinkender Cursor <===
12062  fd07
12063  fd07  24 d7             bit mode         BS-Modus 40/80-Zeichen
12064  fd09  10 0a             bpl acc030       40-Zeichen BS
12065  fd0b
12066  fd0b
12067  fd0b  ===> 80-Zeichen BS <===
12068  fd0b
12069  fd0b  a2 0a             ldx #10
12070  fd0d  20 da cd          jsr vdcin        Daten vom VDC lesen, Register = xr
12071  fd10  29 40             and #%01000000
12072  fd12
12073  fd12   x1xxxxxx  Cursor blinkt
12074  fd12
12075  fd12  d0 06             bne acc040       Cursor blinkt
12076  fd14
12077  fd14  60          accrts rts
12078  fd15
12079  fd15
```

```
12080  fd15  ===> 40-Zeichen BS <===
12081  fd15
12082  fd15  ad 27 0a   acc030 lda blnsw        VIC Flag für Cursor-Blinken
12083  fd18  d0 fa             bne accrts       Cursor blinkt
12084  fd1a
12085  fd1a
12086  fd1a  ===> fester Cursor, Akzent nicht RVS <===
12087  fd1a
12088  fd1a  98         acc040 tya              Tastaturcode für Akzent
12089  fd1b  09 40             ora #%01000000
12090  fd1d  29 7f             and #%01111111
12091  fd1f  4c 2f cc          jmp displc       BS-Ausgabe ac mit akt.
                                                Farbe/Attribut
12092  fd22
12093  fd22
12094  fd22  ===> Abschluß Bearb. DIN-Tast. <===
12095  fd22
12096  fd22  a6 d3      jkychk ldx shflag       Flag für Shift/CTRL/CBM-Taste
12097  fd24  a4 d5             ldy lstx         letzte gedrückte Taste
12098  fd26  6c 3c 03          jmp (keychk)     Zeichen-Bearbeitung
12099  fd29
12100  fd29
12101  fd29  ===> Tastaturtabelle DIN normal <===
12102  fd29
12103  fd29  14 0d 1d   nrmdin .byte $14, $0d, $1d, $88, $85, $86
12104  fd2c  88 85 86
12105  fd2f  87 11 33           .byte $87, $11, $33, $57, $41, $34
12106  fd32  57 41 34
12107  fd35  59 53 45           .byte $59, $53, $45, $01, $35, $52
12108  fd38  01 35 52
12109  fd3b  44 36 43           .byte $44, $36, $43, $46, $54, $58
12110  fd3e  46 54 58
12111  fd41  37 5a 47           .byte $37, $5a, $47, $38, $42, $48
12112  fd44  38 42 48
12113  fd47  55 56 39           .byte $55, $56, $39, $49, $4a, $30
12114  fd4a  49 4a 30
12115  fd4d  4d 4b 4f           .byte $4d, $4b, $4f, $4e, $be, $50
12116  fd50  4e be 50
12117  fd53  4c af 2e           .byte $4c, $af, $2e, $bc, $bd, $2c
12118  fd56  bc bd 2c
12119  fd59  5b 2b bb           .byte $5b, $2b, $bb, $13, $01, $23
12120  fd5c  13 01 23
12121  fd5f  5d 2d 31           .byte $5d, $2d, $31, $3c, $04, $32
12122  fd62  3c 04 32
12123  fd65  20 02 51           .byte $20, $02, $51, $03, $84, $38
12124  fd68  03 84 38
12125  fd6b  35 09 32           .byte $35, $09, $32, $34, $37, $31
12126  fd6e  34 37 31
12127  fd71  1b 2b 2d           .byte $1b, $2b, $2d, $0a, $0d, $36
12128  fd74  0a 0d 36
12129  fd77  39 33 08           .byte $39, $33, $08, $30, $2e, $91
12130  fd7a  30 2e 91
12131  fd7d  11 9d 1d           .byte $11, $9d, $1d, $ff, $ff
12132  fd80  ff ff
12133  fd82
12134  fd82
```

```
12135  fd82  ===> Tastaturtabelle DIN mit Shift <===
12136  fd82
12137  fd82  94 8d 9d   shfdin .byte $94, $8d, $9d, $8c, $89, $8a
12138  fd85  8c 89 8a
12139  fd88  8b 91 40         .byte $8b, $91, $40, $d7, $c1, $24
12140  fd8b  d7 c1 24
12141  fd8e  d9 d3 c5         .byte $d9, $d3, $c5, $01, $25, $d2
12142  fd91  01 25 d2
12143  fd94  c4 26 c3         .byte $c4, $26, $c3, $c6, $d4, $d8
12144  fd97  c6 d4 d8
12145  fd9a  2f da c7         .byte $2f, $da, $c7, $28, $c2, $c8
12146  fd9d  28 c2 c8
12147  fda0  d5 d6 29         .byte $d5, $d6, $29, $c9, $ca, $3d
12148  fda3  c9 ca 3d
12149  fda6  cd cb cf         .byte $cd, $cb, $cf, $ce, $3f, $d0
12150  fda9  ce 3f d0
12151  fdac  cc c0 3a         .byte $cc, $c0, $3a, $dc, $dd, $3b
12152  fdaf  dc dd 3b
12153  fdb2  5e 2a db         .byte $5e, $2a, $db, $93, $01, $27
12154  fdb5  93 01 27
12155  fdb8  5c 5f 21         .byte $5c, $5f, $21, $3e, $04, $22
12156  fdbb  3e 04 22
12157  fdbe  a0 02 d1         .byte $a0, $02, $d1, $83, $84, $38
12158  fdc1  83 84 38
12159  fdc4  35 18 32         .byte $35, $18, $32, $34, $37, $31
12160  fdc7  34 37 31
12161  fdca  1b 2b 2d         .byte $1b, $2b, $2d, $0a, $8d, $36
12162  fdcd  0a 8d 36
12163  fdd0  39 33 08         .byte $39, $33, $08, $30, $2e, $91
12164  fdd3  30 2e 91
12165  fdd6  11 9d 1d         .byte $11, $9d, $1d, $ff, $ff
12166  fdd9  ff ff
12167  fddb
12168  fddb
12169  fddb  ===> Tastaturtabelle DIN mit CBM-taste <===
12170  fddb
12171  fddb  94 8d 9d   cbmdin .byte $94, $8d, $9d, $8c, $89, $8a
12172  fdde  8c 89 8a
12173  fde1  8b 91 96         .byte $8b, $91, $96, $a7, $a8, $97
12174  fde4  a7 a8 97
12175  fde7  a2 aa a3         .byte $a2, $aa, $a3, $01, $98, $a9
12176  fdea  01 98 a9
12177  fded  c4 99 c5         .byte $c4, $99, $c5, $d3, $ce, $a4
12178  fdf0  d3 ce a4
12179  fdf3  9a c2 df         .byte $9a, $c2, $df, $9b, $a1, $c9
12180  fdf6  9b a1 c9
12181  fdf9  d6 d7 d1         .byte $d6, $d7, $d1, $c3, $d5, $c1
12182  fdfc  c3 d5 c1
12183  fdff  cb da d8         .byte $cb, $da, $d8, $cd, $ab, $d9
12184  fe02  cd ab d9
12185  fe05  c8 bf ba         .byte $c8, $bf, $ba, $ca, $b0, $ac
12186  fe08  ca b0 ac
12187  fe0b  ad a6 db         .byte $ad, $a6, $db, $93, $01, $dd
12188  fe0e  93 01 dd
12189  fe11  de b9 81         .byte $de, $b9, $81, $b1, $04, $95
12190  fe14  b1 04 95
12191  fe17  a0 02 a5         .byte $a0, $02, $a5, $03, $84, $38
12192  fe1a  03 84 38
```

```
12193  fe1d  35 18 32          .byte $35, $18, $32, $34, $37, $31
12194  fe20  34 37 31
12195  fe23  1b 2b 2d          .byte $1b, $2b, $2d, $0a, $8d, $36
12196  fe26  0a 8d 36
12197  fe29  39 33 08          .byte $39, $33, $08, $30, $2e, $91
12198  fe2c  30 2e 91
12199  fe2f  11 9d 1d          .byte $11, $9d, $1d, $ff, $ff
12200  fe32  ff ff
12201  fe34
12202  fe34
12203  fe34  ===> Adreßtabelle der Tasturtabellen (DIN-Tastatur) <===
12204  fe34
12205  fe34  29 fd       keydin .word nrmdin     Tastaturtabelle DIN normal
12206  fe36  82 fd              .word shfdin     Tastaturtabelle DIN mit Shift
12207  fe38  db fd              .word cbmdin     Tastaturtabelle DIN mit CBM-taste
12208  fe3a  8b fb              .word mode4      Tastaturtabelle ASCII mit CTRL-taste
12209  fe3c  29 fd              .word nrmdin     Tastaturtabelle DIN normal
12210  fe3e  29 fd              .word nrmdin     Tastaturtabelle DIN normal
12211  fe40
12212  fe40
12213  fe40  ===> Tab. der Tastencodes gültiger Akzente <===
12214  fe40
12215  fe40  af c0 bf    acctbl .byte aigu, grave, crcflx, 0, 0 0 = Reserve für
                                                  weitere Accents
12216  fe43  00 00
12217  fe45
12218  fe45
12219  fe45  ===> Zeigertabelle auf für Akzente zuläss. Tastencodes <===
12220  fe45
12221  fe45  01 03 07    stgref .byte 1, 3, 7, 12, 12, 12
12222  fe48  0c 0c 0c
12223  fe4b
12224  fe4b
12225  fe4b  ===> Tab. der für Akz. zugel. Zeichen <===
12226  fe4b
12227  fe4b  45 c0       acchtb .byte 'e', grave für Accent aigu zugelassene Zeichen
12228  fe4d  41 45 55           .byte 'aeu', aigu      für Accent grave zugelassene
                                                  Zeichen
12229  fe50  af
12230  fe51  41 45 49           .byte 'aeiou'    für Accent circonflexe zugelassene
                                                  Zeichen
12231  fe54  4f 55
12232  fe56
12233  fe56  ===> Reserve für weitere zusammengesetzte Zeichen <===
12234  fe56
12235  fe56  ff ff ff           .byte $ff, $ff, $ff, $ff, $ff, $ff
12236  fe59  ff ff ff
12237  fe5c  ff ff ff           .byte $ff, $ff, $ff, $ff, $ff, $ff
12238  fe5f  ff ff ff
12239  fe62  ff ff ff           .byte $ff, $ff, $ff, $ff
12240  fe65  ff
12241  fe66
12242  fe66
12243  fe66  ===> Tab. Zeichen mit Akzent <===
12244  fe66
12245  fe66  ac          accdtb .byte $ac        Code 'e acc. aigu'
12246  fe67  bf          .byte  crcflx           grave + aigu -> circonflexe
12247  fe68
```

```
12248  fe68  b2          .byte  $b2              Code 'a acc. grave'
12249  fe69  ae          .byte  $ae              Code 'e acc. grave'
12250  fe6a  b3          .byte  $b3              Code 'u acc. grave'
12251  fe6b  bf          .byte  crcflx           aigu + grave -> circonflexe
12252  fe6c
12253  fe6c  b4          .byte  $b4              Code 'a acc. circonflexe'
12254  fe6d  b5          .byte  $b5              Code 'e acc. circonflexe'
12255  fe6e  b6          .byte  $b6              Code 'i acc. circonflexe'
12256  fe6f  b7          .byte  $b7              Code 'o acc. circonflexe'
12257  fe70  b8          .byte  $b8              Code 'u acc. circonflexe'
12258  fe71
12259  fe71
12260  fe71                     .end
12261  fe71                     .lib c128.kernal5
```

4.1.17 Kernal (Fortsetzung)

```
12262  fe71
12264  fe71
12265  fe71
12266  fe71                     *=$ff05
12267  ff05
12268  ff05
12269  ff05  ===> System NMI-Routine <===
12270  ff05
12271  ff05  78          nmi    sei
12272  ff06  48                 pha
12273  ff07
12274  ff07  8a                 txa
12275  ff08  48                 pha
12276  ff09
12277  ff09  98                 tya
12278  ff0a  48                 pha
12279  ff0b
12280  ff0b  ad 00 ff           lda mmucr          MMU 8722 Config. Reg. (Kopie)
12281  ff0e  48                 pha
12282  ff0f
12283  ff0f  a9 00              lda #%00000000
12284  ff11  8d 00 ff           sta mmucr          MMU 8722 Config. Reg. (Kopie)
12285  ff14
12286  ff14  6c 18 03           jmp (inmi)         Hardware NMI ($fa40)
12287  ff17
12288  ff17
12289  ff17  ===> System IRQ-Routine <===
12290  ff17
12291  ff17  48          irq    pha
12292  ff18
12293  ff18  8a                 txa
12294  ff19  48                 pha
12295  ff1a
12296  ff1a  98                 tya
12297  ff1b  48                 pha
12298  ff1c
12299  ff1c  ad 00 ff           lda mmucr          MMU 8722 Config. Reg. (Kopie)
12300  ff1f  48                 pha
12301  ff20
```

```
12302  ff20  a9 00           lda #%000000
12303  ff22  8d 00 ff        sta mmucr        MMU 8722 Config. Reg. (Kopie)
12304  ff25
12305  ff25  ba              tsx
12306  ff26  bd 05 01        lda stack+5,x    CPU-Status vom Stapel
12307  ff29  29 10           and #%00010000   BRK-Bit isolieren
12308  ff2b  f0 03           beq jmpirq       nicht gesetzt, IRQ
12309  ff2d
12310  ff2d  6c 16 03        jmp (ibrk)       BREAK Instruction ($b003)
12311  ff30
12312  ff30
12313  ff30  6c 14 03  jmpirq jmp (iirq)      Hardware-IRQ ($fa65)
12314  ff33
12315  ff33
12316  ff33  68        prend  pla
12317  ff34  8d 00 ff        sta mmucr        MMU 8722 Config. Reg. (Kopie)
12318  ff37
12319  ff37  68               pla
12320  ff38  a8               tay
12321  ff39
12322  ff39  68               pla
12323  ff3a  aa               tax
12324  ff3b
12325  ff3b  68               pla
12326  ff3c  40               rti
12327  ff3d
12328  ff3d
12329  ff3d  ===> System-RESET <===
12330  ff3d
12331  ff3d  a9 00     reset  lda #%00000000
12332  ff3f  8d 00 ff         sta mmucr        MMU 8722 Config. Reg. (Kopie)
12333  ff42
12334  ff42  4c 00 e0         jmp start        System-Start (RESET)
12335  ff45
12336  ff45
12337  ff45  ff ff            .byte $ff, $ff
12338  ff47
12339  ff47
12340  ff47  ===> C128-Kernal Sprungverteiler <===
12341  ff47
12342  ff47  4c fb e5  kspio  jmp spinot       auf I/O schnelle Floppy umschalten
                                               (Sprungv.)
12343  ff4a  4c 3d f2  kclsal jmp closal       alle Files von Gerät in ac schließen
                                               (Sprungv.)
12344  ff4d  4c 4b e2  kc64md jmp c64mod       auf C64-Modus umschalten (Sprungv.)
12345  ff50  4c a5 f7  kdmacl jmp dmacal       DMA-Anforderung ext. RAM zulassen
                                               (Sprungv.)
12346  ff53  4c 90 f8  kbotcl jmp bootcl       Boot-Programm von Diskette laden
                                               (Sprungv.)
12347  ff56  4c 67 f8  kphenx jmp phoenx       Aufruf Kaltstartroutinen
                                               Funktionskarte (Sprungv.)
12348  ff59  4c 9d f7  klkupl jmp lkupla       Geräte- & Sek.-Adr. über la (ac)
                                               suchen (Sprungv.)
12349  ff5c  4c 86 f7  lkups  jmp lkupsa       Geräte-Adr. über Sek.Adr. (yr)
                                               suchen (Sprungv.)
12350  ff5f  4c 2a c0  kswapr jmp jswap        Umschalten 40/80-Zeichen-BS
12351  ff62  4c 27 c0  kdlchr jmp jint80       Charg.gen. in VDC laden
12352  ff65  4c 21 c0  kpfkey jmp jkyset       Funktionstaste belegen
```

```
12353  ff68  4c 3f f7  kstbnk jmp setbnk    Bank für load/sav/verify setzen
                                            (Sprungv.)
12354  ff6b  4c ec f7  kgtcfg jmp getcfg    MMU-Wert zu Bank in xr nach ac
                                            (Sprungv.)
12355  ff6e  4c cd 02  kjsrfr jmp jsrfar    JSR, Bank xr
12356  ff71  4c e3 02  kjmpfr jmp jmpfar    JMP, Bank xr
12357  ff74  4c d0 f7  kifetv jmp indfet    LDA (FETVEC),y von Bank in xr
                                            (Sprungv.)
12358  ff77  4c da f7  kistav jmp indsta    STA (STAVEC),y in Bank in xr
                                            (Sprungv.)
12359  ff7a  4c e3 f7  kicmpv jmp indcmp    CMP (CMPVEC),y mit Bank in xr
                                            (Sprungv.)
12360  ff7d  4c 17 fa  kprimm jmp primm     folg. Text ausgeben und überspringen
                                            (Sprungv.)
12361  ff80
12362  ff80  00                 brk
12363  ff81
12364  ff81  4c 00 c0  kpcint jmp jpcint    Video-Chips/Editor init.
12365  ff84  4c 09 e1  kioini jmp ioinit    I/O-Bausteine initialisieren
                                            (Sprungv.)
12366  ff87  4c 93 e0  kramts jmp ramtas    RAM, Kass.-Puffer u. Vectoren
                                            einrichten (Sprungv.)
12367  ff8a  4c 56 e0  kresto jmp restor    Standard I/O-Vectoren einrichten
                                            (Sprungv.)
12368  ff8d  4c 5b e0  kvecto jmp vector    User I/O-Vectoren einrichten
                                            (Sprungv.)
12369  ff90  4c 5c f7  kstmsg jmp setmsg    Flag setzen Ausgabe/Unterdr.
                                            OPSYS-Meldung (Sprungv.)
12370  ff93  4c d2 e4  ksecnd jmp secnd     Ausgabe Sek.Adr. nach LISTEN auf
                                            IEC-Bus (Sprungv.)
12371  ff96  4c e0 e4  ktksa  jmp tksa      Ausgabe Sek.Adr. nach TALK auf
                                            IEC-Bus (Sprungv.)
12372  ff99  4c 63 f7  kmemtp jmp memtop    höchste Speichergrenze
                                            lesen/schreiben (Sprungv.)
12373  ff9c  4c 72 f7  kmembt jmp membot    unterste Speichergrenze
                                            lesen/schreiben (Sprungv.)
12374  ff9f  4c 12 c0  kscnky jmp jkey      Tastatur-Abfrage
12375  ffa2  4c 5f f7  ksetmo jmp settmo    Timeout-Flag IEC-Bus setzen
                                            (Sprungv.)
12376  ffa5  4c 3e e4  kacptr jmp acptr     Zeichen von IEC-Bus nach ac
                                            (Sprungv.)
12377  ffa8  4c 03 e5  kciout jmp ciout     Ausgabe ac auf IEC-Bus (Sprungv.)
12378  ffab  4c 15 e5  kuntlk jmp untlk     Ausgabe UNTALK auf IEC-Bus
                                            (Sprungv.)
12379  ffae  4c 26 e5  kunlsn jmp nunlsn    Ausgabe UNLISTEN auf IEC-Bus
                                            (Sprungv.)
12380  ffb1  4c 3e e3  klistn jmp listn     Ausgabe LISTEN auf IEC-Bus
                                            (Sprungv.)
12381  ffb4  4c 3b e3  ktalk  jmp talk      ,Ausgabe TALK auf IEC-Bus (Sprungv.)
12382  ffb7  4c 44 f7  kreads jmp readss    akt. Status (RS 232 bzw. System)
                                            nach ac (Sprungv.)
12383  ffba  4c 38 f7  kstlfs jmp setlfs    la, fa, sa aus ac, xr, yr setzen
                                            (Sprungv.)
12384  ffbd  4c 31 f7  kstnam jmp setnam    fnlen, fnadr/+1 aus ac, xr, yr
                                            setzen (Sprungv.)
12385  ffc0  6c 1a 03  kopen  jmp (iopen)   log. File öffnen ($efbd)
12386  ffc3  6c 1c 03  kclose jmp (iclose)  log. File schliessen ($f188)
12387  ffc6  6c 1e 03  kchkin jmp (ichkin)  Eingabe-Kanal öffnen ($f106)
```

```
12388  ffc9  6c 20 03   kckout jmp (ickout)     Ausgabe-Kanal öffnen ($f14c)
12389  ffcc  6c 22 03   kclrch jmp (iclrch)     Ein-/Ausg.-Kanal schließen ($f226)
12390  ffcf  6c 24 03   kbasin jmp (ibasin)     Zeichen akt. Kanal nach ac ($ef06)
12391  ffd2  6c 26 03   kbsout jmp (ibsout)     Ausgabe ac auf akt. Kanal ($ef79)
12392  ffd5  4c 65 f2   kloads jmp loadsp       load/verify (ac) ab Adresse in xr/yr
                                                (Sprungv.)
12393  ffd8  4c 3e f5   ksaves jmp savesp       save ab ac bis xr, yr (Sprungv.)
12394  ffdb  4c 65 f6   ksttim jmp settim       System-Uhr setzen (Sprungv.)
12395  ffde  4c 5e f6   krdtim jmp rdtim        System-Uhr nach ac, xr, yr
                                                (Sprungv.)
12396  ffe1  6c 28 03   kstop  jmp (istop)      Stopp-Taste lesen ($f66e)
12397  ffe4  6c 2a 03   kgetin jmp (igetin)     Zeichen akt. Kanal nach ac ($eeeb)
12398  ffe7  6c 2c 03   kclall jmp (iclall)     off. Files =0, Vorgabe-I/O ($f222)
12399  ffea  4c f8 f5   kclock jmp udtim        System-Uhr aktualisieren (Sprungv.)
12400  ffed  4c 0f c0   kscror jmp jscorg       BS-Organisation lesen
12401  fff0  4c 18 c0   kplot  jmp jplot        Cursor-Pos. lesen/schreiben
12402  fff3  4c 81 f7   kiobas jmp iobase       Basisadresse I/O-Bausteine nach
                                                xr/yr (Sprungv.)
12403  fff6
12404  fff6
12405  fff6  ff ff             .byte $ff, $ff
12406  fff8
12407  fff8
12408  fff8  ===> Adresse Initialisierung akt. System <===
12409  fff8
12410  fff8  24 e2       system .word mod128     Initialisierung C 128
12411  fffa
12412  fffa
12413  fffa  ===> System NMI-Vector <===
12414  fffa
12415  fffa  05 ff       sysnmi .word nmi        System NMI-Routine
12416  fffc
12417  fffc
12418  fffc  ===> System RESET-Vector <===
12419  fffc
12420  fffc  3d ff       sysres .word reset      System-RESET
12421  fffe
12422  fffe
12423  fffe  ===> System IRQ-Vector <===
12424  fffe
12425  fffe  17 ff       sysirq .word irq        System IRQ-Routine
12426  0000
12427  0000
12428  0000                     .end
12429  0000
12430  0000
12431  0000                     .end

Fehler in Pass 1 = 00000
Fehler in Pass 2 = 00000

Assemblierung Ende
```

4.2 Verweistabelle (Crossreference)

```
acc010 = $fcf3: 12047#, 12051
acc020 = $fcfb: 12048 , 12053#
acc030 = $fd15: 12064 , 12082#
acc040 = $fd1a: 12075 , 12088#
accdtb = $fe66: 12031 , 12245#
accent = $fcf1: 12007 , 12046#
acchtb = $fe4b: 12028 , 12227#
accrts = $fd14: 12041 , 12059 , 12077#, 12083
acctbl = $fe40: 12047 , 12215#
acp00a = $e44d:  6401#,  6402
acp01  = $e484:  6418 ,  6446#
acp03  = $e486:  6452#,  6461
acp03a = $e48d:  6456#,  6458
acp03b = $e498:  6466#,  6490
acp03c = $e49a:  6471#,  6473 ,  6476
acp03d = $e4a8:  6484#,  6486 ,  6489
acp04  = $e4b5:  6454 ,  6495#
acp04a = $e4bf:  6479 ,  6504#
acp04b = $e4cb:  6509 ,  6512#
acp0a1 = $e452:  6407#,  6435
acp0a2 = $e45c:  6413#,  6415 ,  6421
acp0a3 = $e47d:  6426 ,  6440#
acptr  = $e43e:  6387#,  9071 ,  9909 ,  9912 ,  9947 , 12376
aigu   =   $af:    23#, 12215 , 12228
aktcol = $035d:   266#
areg   =   $06:    34#, 11281$, 11312
autoff = $caea:  3874 ,  4102#
auton  = $caed:  3872 ,  4108#
ba     =   $c6:   118#, 10963$, 11114 , 11127 , 11476$
bakchr = $cc00:  3502 ,  3646 ,  3650 ,  3702 ,  4031 ,  4388 ,  4418#
bakot1 = $cc09:  4420 ,  4428#
bakout = $cc18:  4423 ,  4441#
bank   =   $02:    30#,  4610 ,  5318$,  5715$,  5720 ,  5800$,  5808 ,  5845$,  5847
                  5860 , 11308 , 11345$, 11528$
baudo  = $e850:  7586#,  9346 ,  9347
baudof = $0a16:   387#,  7315 ,  7318 ,  7623 ,  7627 ,  9372$,  9376$
baudop = $e864:  7600#,  9353 ,  9354
beep   = $c98e:  3294 ,  3792#
beeper =   $f9:   224#,  3792 ,  4210$
bellof = $cb3a:  3879 ,  4209#
bellon = $cb37:  3878 ,  4202#
birqvc = $4006:   490#, 11721
bit010 = $e698:  7033 ,  7036#
bit020 = $e69c:  7036 ,  7040#
bitabl = $035e:   267#,  3947$,  4272 ,  4291 ,  4296$,  4311
bitci  =   $a8:    82#,  7048$,  7067$,  7082 ,  7685$,  8235$,  8354$,  8375 ,
                 8613$,  8655 ,  8663$
bitcnt = $e68e:  7024#,  9326
bitnum = $0a15:   386#,  6963 ,  7138 ,  7684 ,  9327$
bitout = $cb90:  4274 ,  4301#
bitpos = $cb9f:  4271 ,  4289 ,  4310 ,  4325#
bits   = $ce6c:  3759 ,  4331 ,  5002#
```

```
bitsav = $cb8d:  4296#,  4312
blank  = $cc27:  3659 ,  3683 ,  4023 ,  4467#
blink  = $c6e7:  1954 ,  3131#
blkdat = $fa08: 11383 , 11632#
blknex = $f9b3: 11482 , 11563#
blkred = $f9d5: 11429 , 11587#
blnk2  = $e930:  7796#,  7798
blnkng = $0a39:   447#,  5560$,  8044$,  8799
blnon  = $0a26:   410#,  1641$,  3140 ,  3158 ,  3178 ,  3180$,  4166 ,  4189 ,
                 4195$,  4828 ,  4832$
blnsw  = $0a27:   415#,  1660$,  3134 ,  4803$,  4826$, 12082
bn10   = $ef15:  8951 ,  8967 ,  8982#
bn20   = $ef22:  8983 ,  8995#
bn30   = $ef5c:  8995 ,  9068#
bn40   = $ef63:  9069 ,  9076#,  9097
bn41   = $ef65:  9077#,  9084
bn42   = $ef66:  9078#,  9087
bn50   = $ef67:  8998 ,  9083#,  9098
bo10   = $ef84:  9107 ,  9115#
bo20   = $ef8a:  9115 ,  9125#
boot01 = $f86d: 11335#, 11349
boot02 = $f886: 11337 , 11348#
boot04 = $f8a1: 11373#, 11374 , 11376
boot05 = $f8a9: 11383#, 11387
boot06 = $f8eb: 11406 , 11421#, 11438
boot07 = $f8ee: 11419 , 11424#
boot08 = $f8fb: 11436#, 11442
boot09 = $f915: 11451#, 11456
boot10 = $f91f: 11461#, 11466
boot11 = $f92a: 11462 , 11468#
boot12 = $f938: 11478#, 11484
boot13 = $f945: 11479 , 11486#
boot14 = $f94b: 11490#, 11493
boot15 = $f97e: 11508 , 11524#
boot17 = $f9a7: 11546 , 11554#
boot19 = $f9be: 11566 , 11570#
bootcl = $f890: 11358#, 12346
botcls = $f98b: 11421 , 11486 , 11535#
break  = $f5b5: 10024 , 10381 , 10591 , 10631#
breake = $e715:  7195#,  7212
brkerr = $f323:  9943 , 10024#
brtbl1 = $f4c7: 10415#, 10416
brtbl2 = $f4ee: 10429 , 10447#
brtbl3 = $f4f1: 10439 , 10445 , 10448#
brtbl4 = $f4fd: 10449 , 10459#
brtbl5 = $f502: 10457 , 10463#
brtblk = $f4c5: 10304 , 10337 , 10409#, 10460
brtbt1 = $f4bc: 10400#, 10401
brtbyt = $f4ba: 10213 , 10237 , 10261 , 10265 , 10309 , 10330 , 10394#
brtcmd = $f50c: 10173 , 10477#
brtend = $f48c: 10164 , 10197 , 10220 , 10346#, 10362 , 10373 , 10386
brteoi = $f47f: 10288 , 10336#
bsi010 = $e7e4:  7426 ,  7440#
bsi232 = $e7ce:  7422#,  8957
bsit   = $eea8:  8841 ,  8844 ,  8852#
bsiv   = $ee9b:  8057 ,  8760 ,  8778 ,  8841#
bso100 = $e770:  7284 ,  7299#
bso120 = $e794:  7304 ,  7333#
```

```
bso232 = $e75f:  7289#,  9185
bsobad = $e75c:  7284#,  7292
bsour  =   $95:    65#,  6190$,  6290 ,  6312$,  6520$,  6536$,  6578$
bsour1 =   $a4:    78#,  6466$,  6496$,  6513 ,  8249 ,  8251$,  8267$,  8612$,  8682
                  8684$
btblck = $f46e: 10311 , 10321#
btbrke = $f4a6: 10302 , 10373#
btckst = $f41a: 10232 , 10245#
btfnam = $f3d2: 10181#, 10185
btfnd  = $f40a: 10216 , 10229#
btldab = $f440: 10269 , 10281#
btldad = $f421: 10240 , 10246 , 10256#
btlnls = $f477: 10314 , 10329#
btlode = $f498: 10251 , 10315 , 10359#
btnxbk = $f454: 10300#, 10324
btomer = $f4b2: 10305 , 10338 , 10386#
btopen = $f3ea: 10193 , 10204#
buf    = $0200:   233#,  7876
bufpt  =   $a6:    80#,  7931$,  7932 ,  9024$,  9055$,  9153$,  9303$
bumpnt = $cbfb:  4401 ,  4409#
burst  = $f3a1:  9885 , 10140#
burst1 = $f3c7: 10160 , 10172#
burst2 = $f3c9: 10173#, 10176
c3po   =   $94:    64#,  6181 ,  6187$,  6566 ,  6570$
c64beg = $e263:  5799 ,  5813#
c64md1 = $e255:  5799#,  5802
c64mod = $e24b:  5228 ,  5773#, 12344
capslk = $c644:  2878 ,  2968#
cas1   =   $c0:   113#,  8067$,  8904$,  8913
casout = $ef8c:  9128#,  9682
caston = $0a0b:   316#,  8037$,  8147
cbmdin = $fddb: 12171#, 12207
cbmkey = $e2c4:  5695 ,  5743 ,  5898#, 11437
charen =   $d9:   163#,  1642$,  1897
chips  = $d000:   498#, 11045 , 11046
chkdin = $fcc6:  3060 , 12001#
chras1 = $fce9: 12026 , 12036#
chrasm = $fcd8: 12018#, 12029
chrrom = $d000:   497#,  4911 ,  4912 ,  4954
ci2    = $e50c:  6567 ,  6574#
ci4    = $e511:  6571 ,  6578#
cia1   = $dc00:  1132#
cia2   = $dd00:  1326#
cint   = $c07b:  1512 ,  1613#
cint01 = $c0e8:  1688#,  1697
cint02 = $c0f8:  1702#,  1705
cint03 = $c108:  1713#,  1716
cint04 = $c113:  1721#,  1724
cint05 = $c124:  1708 ,  1732#
cint06 = $c141:  1744 ,  1747#
ciout  = $e503:  6566#,  9120 ,  9499 , 10577 , 10580 , 10588 , 12377
ck50   = $f17e:  9622 ,  9627#
ck60   = $f181:  9625 ,  9628#
ckdsrx = $e755:  7235 ,  7275#,  7364 ,  9393
cki010 = $e7aa:  7370#,  7375
cki020 = $e7b8:  7390#,  7395
cki080 = $e7bf:  7397#,  7414
cki100 = $e7c5:  7346 ,  7354 ,  7408#
```

```
cki110 = $e7cc:  7365 ,  7416#
cki232 = $e795:  7338#,  9525
ckirts = $f123:  9514 ,  9517 ,  9537#,  9560
ckiser = $f127:  9518 ,  9544#
ckit   = $c5bf:  2759 ,  2802#
ckit1  = $c5d7:  2809 ,  2830#
ckit2  = $c697:  2704 ,  2993 ,  3008 ,  3043#
ckitap = $f11d:  9521 ,  9530#
cko020 = $e73a:  7241#,  7246
cko030 = $e741:  7251#,  7255
cko040 = $e74e:  7264#,  7270
cko100 = $e75a:  7225 ,  7236 ,  7269 ,  7278#
cko232 = $e729:  7217#,  9595
ckorts = $f169:  9589 ,  9607#,  9631
ckoser = $f16d:  9590 ,  9615#
ckotap = $f163:  9593 ,  9600#
clall2 = $f236:  9799 ,  9803#
clkhi  = $e545:  6244 ,  6324 ,  6396 ,  6429 ,  6548 ,  6618 ,  6624#, 10208 , 10346
clklo  = $e54e:  5652 ,  6221 ,  6587 ,  6633#
cln232 = $f0b0:  9311 ,  9409#,  9667
closa1 = $f248:  9817 ,  9823#
closa2 = $f250:  9824 ,  9829#,  9839
closa3 = $f254:  9831#,  9835
closal = $f23d:  9814#, 11362 , 12343
closax = $f264:  9832 ,  9841#
clp00  = $c2ec:  2112 ,  2116 ,  2125#
clp1   = $c2ef:  2122 ,  2126#
clp2   = $c2a5:  2042 ,  2069#
clp21  = $c2b5:  2076 ,  2081#
clp22  = $c2b8:  2079 ,  2083#
clp7   = $c2f5:  2084 ,  2127 ,  2130#
clr80  = $c4c0:  2522 ,  2542#
clrbit = $cb85:  2330 ,  2348 ,  2514 ,  3251 ,  4081 ,  4289#
clreah = $0a3d:   459#,  2561$,  2583 ,  2584$,  2606 ,  2609$,  2640
clreal = $0a3c:   458#,  2557$,  2579 ,  2580$,  2645
clrln  = $c4a5:  1754 ,  2314 ,  2380 ,  2513#
clrprt = $c4aa:  2520#,  4008 ,  4039
clrpt1 = $c4b2:  2527#,  2535
clrpt3 = $c50b:  2575 ,  2592#
clrpt4 = $c536:  2616 ,  2619#
clrpt5 = $c53c:  2627#,  2641 ,  2646
clrpt6 = $c53e:  2587 ,  2618 ,  2632#
clrpt7 = $c543:  2635#,  2636
clrts  = $f16b:  9609#,  9646
cls10  = $c145:  1753#,  1758
clsei  = $f59e:  9720 , 10002 , 10604#, 10631
clsr   = $c142:  1735 ,  1741 ,  1752#,  3456
clsrts = $f5b3: 10605 , 10619#
cmpare = $02be:   239#, 11156
cmparo = $f81c: 11265#
cmpo   =   $b0:    90#,  7995$,  8115 ,  8119 ,  8125 ,  8197 ,  8214 ,  8220 ,  8225
                8288$,  8291$,  8343
cmpste = $eeb7:  8466 ,  8583 ,  8723 ,  8869#, 10582
cnc3x  = $c81d:  3411 ,  3417#
cntdn  =   $a5:    79#,  6300$,  6335$,  6390$,  6425 ,  6434$,  8708 ,  8714$,
                8787$, 10238$, 10295$, 10323$, 10331$, 10459$
coladj = $ce5c:  3372 ,  4982#
colchk = $c7da:  3347#,  3460
```

```
coldvc = $4000:   489#,  5343 ,  5344
color  =   $f1:   185#,  2531 ,  2611 ,  2919 ,  2921$,  3169 ,  3261 ,  3263$,
                 3364$,  3370 ,  3373$,  3525 ,  3527$,  3548 ,  3550$,  3599 ,
                 3601$,  3608 ,  3610$,  3617 ,  3619$,  3626 ,  3628$,  4467 ,
                 4477 ,  4787
colram = $d800:   500#,  1819 ,  5011
coltab = $ce4c:  3348 ,  4962#
colums =   $ee:   182#,  3929 ,  4541
config = $f7f0: 11082 , 11134 , 11143 , 11153 , 11161 , 11167#
contrl = $c7b9:  1594 ,  3314#
cr     =   $0d:    13#,  2013 ,  2072 ,  2083 ,  3211 ,  5087 ,  5095 ,  5108 ,  5114
                 5120 ,  5127 ,  5127 ,  5134 ,  9076 , 10876 , 10881 , 10887 , 10903
                10906 , 10909 , 10913 , 10916 , 11446 , 11472
crcflx =   $bf:    25#, 12215 , 12246 , 12251
crsr40 = $cd99:  4772 ,  4802#
crsra  = $cb14:  4124 ,  4140 ,  4159#,  4184
crsral = $caf2:  3890 ,  4116#
crsrb  = $cb33:  4168 ,  4195#
crsrbl = $cb21:  3877 ,  4173#
crsrc  = $cb36:  4117 ,  4130 ,  4197#
crsrd  = $cb1a:  4146 ,  4166#
crsrdn = $c85a:  3301 ,  3479#
crsrdo = $cd91:  4161 ,  4795#
crsrf1 = $cdb3:  4811 ,  4826#
crsrf2 = $cdc9:  4829 ,  4837#
crsrlf = $c875:  3430 ,  3502#,  3671
crsrnf = $cb0b:  3876 ,  4145#
crsrof = $cd9f:  2010 ,  4810#
crsrok = $c865:  3481 ,  3485#,  3504
crsron = $cd6f:  2005 ,  4771#
crsrrt = $c854:  3306 ,  3471#
crsrts = $c866:  3486#,  3493 ,  3503
crsrul = $cafe:  3892 ,  4129#
crsrup = $c867:  3427 ,  3491#
crsrw1 = $c85d:  3472 ,  3480#,  3495
crsrw2 = $c86d:  3495#,  3507
crsw   =   $d6:   149#,  1639$,  2016$,  2065 ,  2070$,  2118$,  3202$,  8987$
cs10   = $e9df:  7939 ,  7949 ,  7958#,  7969
cs25   = $e9e7:  7940 ,  7960 ,  7963#,  7970
cs30   = $e9cf:  7943#,  7973
cs40   = $e9d2:  7948#,  7950
csberr = $e42a:  6372#,  6443
cste1  = $e9c8:  7939#,  7988 ,  9259 , 10035
cste2  = $e9e9:  7969#,  8018 ,  9287 , 10652
ctlchr = $c7b6:  3235 ,  3309#
ctlexe = $c7f6:  3339 ,  3380#
ctlrts = $c7f5:  3315 ,  3375#
ctltbe = $c79a:  3293#,  3383 ,  3386
ctltbs = $c78c:  3275#,  3338
ctlvec = $0334:   258#,  1703$,  3309
ctserr = $e675:  6951 ,  6987#
cunlsn = $f5b0:  9504 , 10618#
curbnk = $0ac0:   462#,  5828$,  5840 ,  5864 ,  5878$, 11333$, 11335 , 11348$
curcnt = $0a28:   420#,  1659$,  3137$,  3152$,  4491$
curcol = $0a33:   440#,  4780$,  4816
curmod = $0a2b:   424#,  1680$,  4119 ,  4134 ,  4151 ,  4159$,  4178 ,  4796
cursor = $cd57:  1521 ,  2160 ,  4586 ,  4750#
d1cra  = $dc0e:  1249#,  5462$,  6743 ,  6746$,  6772 ,  6784$,  8032 ,  8148$
```

```
d1crb  = $dc0f:  1286#,  5473$,  5647$,  8034$,  8184$,  8636$
d1ddra = $dc02:  1155#,  5487$
d1ddrb = $dc03:  1162#,  5480$
d1icr  = $dc0d:  1220#,  5444$,  6243 ,  6267 ,  6392 ,  6452 ,  6736 ,  6762$,  6786
                 8030$,  8151 ,  8186 ,  8359$,  8370$,  8554$,  8556 ,  8633 , 10211
                10400 , 10415 , 11716
d1pra  = $dc00:  1136#,  2382$,  2697$,  2734$,  2820$,  3088$,  5449$, 10744$,
                10750$
d1sdr  = $dc0c:  1214#,  6201$,  6291$,  6495 , 10403 , 10418
d1t1h  = $dc05:  1170#,  6767$,  8145$
d1t1l  = $dc04:  1169#,  6770$,  8141$
d1t2h  = $dc07:  1175#,  5640$,  8144 ,  8170 ,  8174 ,  8181$,  8631$
d1t2l  = $dc06:  1174#,  5639$,  8136 ,  8173 ,  8180$,  8630$
d2cra  = $dd0e:  1432#,  5463$,  7313$,  7331$
d2crb  = $dd0f:  1469#,  5474$,  7640$,  7670$
d2ddra = $dd02:  1355#,  5511$
d2ddrb = $dd03:  1368#,  5481$,  9432$
d2icr  = $dd0d:  1403#,  5445$,  7006$,  7017$,  7093$,  7472$,  7511$,  7573$,
                 7643$,  9419$, 11689$, 11691
d2pra  = $dd00:  1330#,  1617 ,  1618$,  5499$,  6195 ,  6213 ,  6215$,  6261 ,  6262
                 6279$,  6305 ,  6306 ,  6330 ,  6333$,  6409 ,  6411$,  6413 ,  6414
                 6456 ,  6457 ,  6471 ,  6472 ,  6484 ,  6485 ,  6527 ,  6529$,  6553
                 6554 ,  6589 ,  6591$,  6624 ,  6626$,  6633 ,  6635$,  6642 ,  6644$
                 6651 ,  6653$,  6661 ,  6662 ,  7499 ,  7508$,  9443 ,  9444$, 10420
                10422$, 10468 , 10470$
d2prb  = $dd01:  1342#,  6945 ,  7230 ,  7251 ,  7257 ,  7262$,  7264 ,  7359 ,  7379
                 7384$,  7390 ,  7614 ,  9386 ,  9437$
d2t1h  = $dd05:  1384#,  7319$
d2t1l  = $dd04:  1383#,  7316$
d2t2h  = $dd07:  1389#,  7626 ,  7628$,  7647$,  7658$,  7682$
d2t2l  = $dd06:  1388#,  7621 ,  7624$,  7646$,  7655$,  7681$
d8502  =   $00:   516#,  5537$,  5794$
data   =   $c5:   117#,  8254$,  8295 ,  8320$,  8712$,  8728 ,  8736 ,  8737$
datahi = $e557:  6193 ,  6222 ,  6236 ,  6323 ,  6619 ,  6642#
datalo = $e560:  6317 ,  6428 ,  6506 ,  6546 ,  6651#
datax  =   $ef:   183#,  2091$,  2094$,  2096 ,  2130$,  2138 ,  2158 ,  3187$,  3209
                 3330$,  3842$
dctrl1 = $c7c9:  3318 ,  3329#
dctrl2 = $c7d0:  3321 ,  3324 ,  3337#
dctrl3 = $c7d2:  3338#,  3342
dctrl5 = $c7dc:  3348#,  3352
dctrl6 = $c7e5:  3349 ,  3359#
dctrl7 = $c7ec:  3360 ,  3370#
debpia = $e569:  6237 ,  6252 ,  6258 ,  6344 ,  6401 ,  6661#,  6663
decas1 = $f9fe: 11620#, 11624
decas2 = $fa05: 11621 , 11626#
decasc = $f9fb: 11573 , 11579 , 11618#
dejavu = $0a02:   300#,  5196 ,  5349$
delay  = $0a24:   407#,  1661$,  2992$,  3025 ,  3028$
delchr = $c93d:  3676 ,  3681 ,  3700#
delete = $c91b:  3304 ,  3671#
deloop = $c923:  3675#,  3705
delout = $c932:  3666 ,  3673 ,  3688#,  4017 ,  4053 ,  4066 ,  4082
dfltn  =   $99:    69#,  2075 ,  5564$,  7338$,  8935 ,  8966 ,  9537$,  9798 ,
                 9806$,  9823 ,  9827$
dflto  =   $9a:    70#,  2078 ,  5567$,  7217$,  9105 ,  9607$,  9790 ,  9803$,  9816
                 9820$
dinflg = $0ac5:   465#, 11905$, 11927 , 11983$, 12005 , 12053$
```

```
dirms1 = $fa1e: 11660#, 11675
dirms2 = $fa27: 11662 , 11665#
dirms3 = $fa3a: 11672 , 11677#
discli = $e59f:  6349 ,  6380 ,  6512 ,  6560 ,  6714#
dismo2 = $e59a:  6705#,  6706
dismo3 = $e59e:  6677 ,  6680 ,  6698 ,  6709#
disp80 = $cc4a:  4501 ,  4514#
displ  = $cc40:  3176 ,  4500#
displc = $cc2f:  2212 ,  4477#, 12091
displt = $cc32:  3649 ,  3703 ,  4483#
disply = $cc34:  1513 ,  4488#,  4836
dissei = $e573:  6192 ,  6220 ,  6235 ,  6387 ,  6545 ,  6586 ,  6671#
dlabye = $e535:  6382 ,  6540 ,  6608#
dlad00 = $e53b:  6614#,  6615
dladlh = $e538:  6511 ,  6609#
dline  = $ca52:  3875 ,  3975#
dmacal = $f7a5: 11082#, 12345
dmacmd = $df01:   502#, 11322$
dodma  = $03f0:   271#,  5395$, 11088
dodmao = $f85a:  5394 , 11321#
doit   = $c207:  1886 ,  1913 ,  1927#
dopfky = $c6ca:  3067 ,  3095#
doplo1 = $cc77:  4556 ,  4559#
doplo2 = $cc84:  4564 ,  4570#
doplo3 = $cc85:  4553 ,  4557 ,  4571#
doplo4 = $cc86:  4567 ,  4568 ,  4577#
doplo5 = $cc95:  4547 ,  4591#
doscrl = $c373:  2264 ,  2275#
dotab  = $c94f:  3295 ,  3710#
dotab1 = $c951:  3715#,  3720
dotab2 = $c95c:  3717 ,  3729#
dpsw   =   $9c:    72#,  7998$,  8203 ,  8347$,  8391$
dskbfr = $fa16: 11409 , 11410 , 11646#
dskint = $fa15: 11397 , 11398 , 11641#
dspnml = $cb48:  3885 ,  4225#
dspout = $cb4f:  4220 ,  4228#
dsprvs = $cb3f:  3889 ,  4217#
dsrerr = $e672:  6950 ,  6981#
dummy  =   $ff:   229#, 11242 , 11257$, 11272
eadp   = $eb88:  8275#,  8318
eah    =   $af:    89#,  7786 ,  7823 ,  7860$,  7894$,  8873 ,  9913$,  9931$,
                 9985$,  9986 , 10112$, 10134 , 10266$, 10276$, 10282 , 10451$,
                10452 , 10523$, 11478 , 11481$
eal    =   $ae:    88#,  7789 ,  7819 ,  7857$,  7888$,  8871 ,  9910$,  9928$,
                 9982$, 10108$, 10133 , 10262$, 10275$, 10281 , 10448$, 10522$,
                11111 , 11120,  11475
eally  = $f7c9:  9963 , 10437 , 11120#
ealsy  = $f7bf:  9976 , 10447 , 11111#
eeos   = $ca9f:  3871 ,  4038#
eloup2 = $cbd3:  4376#,  4389
eloup3 = $cbe5:  4383 ,  4388#
enabl  = $0a0f:   323#,  5636$,  7008 ,  7010$,  7095 ,  7096$,  7241 ,  7299 ,  7370
                 7408 ,  7451 ,  7457 ,  7475$,  7484 ,  7510 ,  7572 ,  7642 ,  7677
                 7678$,  9447$
enamo1 = $e5ba:  6715 ,  6718 ,  6729#
endbye = $cbea:  4380 ,  4386 ,  4391#
eom    =   $00:    12#, 11446 , 11472
erasf1 = $caa2:  4039#,  4046 ,  4050 ,  4051
```

```
erexit = $f6ad: 10860 , 10871#
err1   = $eff0:  9197 ,  9237#
err16  = $f320:  9991 , 10020#, 10115
err2   = $eff3:  9193 ,  9242#
err3   = $f149:  9511 ,  9577#,  9583
err4   = $f317:  9920 , 10008#, 10049 , 10056
err5   = $f140:  9548 ,  9565#,  9619 ,  9632
err6   = $f143:  9532 ,  9569#
err7   = $f146:  9573#,  9586 ,  9602
err8   = $f31a:  9879 , 10012#
err9   = $f31d: 10016#, 10030 , 10033
erro4  = $f55e: 10555#
erro8  = $f55b: 10551#, 10561
erro9  = $f558: 10547#, 10650
error1 = $f67c:  9237 , 10797#
error2 = $f67f:  9242 , 10803#
error3 = $f682:  9577 , 10809#
error4 = $f685:  9273 , 10008 , 10224 , 10555 , 10815#
error5 = $f688:  9487 ,  9565 , 10167 , 10821#
error6 = $f68b:  9569 , 10827#
error7 = $f68e:  9573 , 10833#
error8 = $f691: 10012 , 10551 , 10839#
error9 = $f694:  9250 , 10016 , 10547 , 10845#
errora = $f697: 10020 , 10389 , 10848#
esc    =   $1b:    14#,  3218 ,  3314 ,  3837
esc01  = $c9ca:  3838 ,  3848#
escape = $c9c1:  1522 ,  1596 ,  3837#
escchr = $c9be:  3223 ,  3832#
escrts = $c9dd:  3854 ,  3865#
escvct = $c9de:  3859 ,  3862 ,  3871#
escvec = $0338:   260#,  3832
etol   = $ca76:  3888 ,  4007#
etol1  = $ca79:  4008#,  4015
etout  = $ca88:  4017#,  4029
ets100 = $ca9a:  4026 ,  4031#
etsto1 = $ca8e:  4023#,  4032
etstol = $ca8b:  3887 ,  4022#
fa     =   $ba:   108#,  9211 ,  9469 ,  9653 ,  9708 ,  9776$,  9814$,  9829 ,  9859
                 9902 , 10150 , 10537 , 10569 , 10607 , 10955$, 10972 , 11359$, 11403
                11415
faf    = $e99a:  7901#,  7916 ,  9267 , 10046
faf20  = $e9a7:  7910#,  7921
faf30  = $e9bc:  7911 ,  7924#
faf40  = $e9bd:  7902 ,  7925#
fah    = $e8d0:  7704#,  7724 ,  7759 ,  7901 ,  9279 , 10054
fah40  = $e918:  7710 ,  7770#
fah45  = $e916:  7728 ,  7764#
fah46  = $e917:  7715 ,  7765#
fah50  = $e8ef:  7718 ,  7721 ,  7726#
fah55  = $e8fb:  7738#,  7743
fah56  = $e909:  7751#,  7756
fah57  = $e912:  7753 ,  7758#
fat    = $036c:   269#,  9212$,  9737 ,  9738$,  9775 ,  9834
fetch  = $02a2:   235#,  4685 ,  5387$,  5693 ,  5714 , 11136
fetcho = $f800:  5386 , 11238#
fetchx = $02aa:   236#,  4607$,  4682 ,  5683$, 11132$
findky = $cd20:  4617 ,  4624 ,  4666 ,  4700#
fistrt = $cbb5:  2024 ,  3975 ,  4352#,  4356
```

```
fkaddr = $e2bc:  5841 ,  5851 ,  5886#, 11340
fkbank = $e2c0:  5844 ,  5892#, 11344
flnmi  = $e8a9:  7537 ,  7567 ,  7654#
flsho1 = $cb2e:  4174 ,  4189#
flshof = $c8dc:  3450 ,  3626#
flshon = $c8d5:  3300 ,  3617#
fnadr  =   $bb:   109#, 10946$, 10947$, 11096
fnadrx = $0a35:   442#, 11093$, 11099
fnadry = $f7ae:  7841 ,  7913 ,  9319 ,  9498 , 10141 , 10181 , 10501 , 11093#
fnbank =   $c7:   119#, 10964$, 11095
fnd0   = $cbc0:  4353 ,  4361#
fndend = $cbc3:  2021 ,  3249 ,  3636 ,  3823 ,  4233 ,  4366#,  4368
fndeol = $cb52:  3882 ,  4233#
fndky1 = $cd23:  4702#,  4706
fndky2 = $cd2b:  4703 ,  4708#
fndsol = $cbb1:  3881 ,  4346#
fnlen  =   $b7:   105#,  7838 ,  7910 ,  9264 ,  9314 ,  9457 ,  9492 ,  9501 ,
                 9876 ,  9894$, 10043 , 10154$, 10205$, 10488 , 10497 , 10504 ,
                10560 , 10945$
framee = $e718:  7201#,  7211
frmerr = $e425:  6310 ,  6361#
fsblk  =   $be:   111#,  8059$,  8395 ,  8407 ,  8557 ,  8561$,  8569$,  8715 ,
                 8751$,  8781
fungt1 = $c6d1:  3101#,  3106
fungt3 = $c6da:  3102 ,  3108#
funtab = $c6dd:  3066 ,  3116#
funts1 = $c6af:  3066#,  3070
gdbln  = $0a29:   421#,  3163$,  3170 ,  4834
gdcol  = $0a2a:   422#,  3155 ,  3167$,  4835
get1ch = $cb58:  2090 ,  3647 ,  3701 ,  4239#,  4376
get232 = $eefd:  8956#,  9083
get80  = $cb65:  4242 ,  4255#
getbit = $cb74:  2234 ,  3480 ,  4014 ,  4028 ,  4045 ,  4072 ,  4266#,  4352 ,  4367
                 4385
getbt1 = $cb76:  2305 ,  2322 ,  2345 ,  2368 ,  3680 ,  3960 ,  4271#,  4308
getcfg = $f7ec: 11161#, 12354
getlfs = $f212:  9513 ,  9585 ,  9648 ,  9769#, 11060
getpnt = $cde6:  2450 ,  4259 ,  4523 ,  4875#
getrts = $ef04:  8942 ,  8960#
getsp  = $e574:  6676#,  8050
getusr = $cdf9:  2501 ,  4255 ,  4519 ,  4778 ,  4785 ,  4815 ,  4893#
gn10   = $eef9:  8936 ,  8950#
go64   = $e048:  5221 ,  5228#
graphm =   $d8:   155#,  1635$,  1836 ,  1848 ,  1851 ,  1875 ,  1942
grave  =   $c0:    24#, 12215 , 12227
helptb = $cef0:  5134#
holdof = $0a3a:   452#,  5562$,  6676 ,  6714 ,  8803
home   = $c150:  1752 ,  1763#,  3580
home1  = $c8bc:  3575 ,  3580#
homtst = $c8b3:  3303 ,  3573#
ibasin = $0324:   251#, 12390
ibrk   = $0316:   244#, 12310
ibsout = $0326:   252#, 12391
ichkin = $031e:   248#, 12387
ickout = $0320:   249#, 12388
iclall = $032c:   255#, 12398
iclose = $031c:   247#, 12386
iclrch = $0322:   250#, 12389
```

```
igetin = $032a:   254#, 12397
iirq   = $0314:   243#,  5276 ,  5280$,  8051 ,  8054 ,  8084 ,  8823$,  8826$,
                  8842$,  8845$, 12313
iline  = $ca3d:  3880 ,  3956#
iload  = $0330:   256#,  9849
imparm =   $ce:   123#, 11666$, 11669$, 11671
inbit  =   $a7:    81#,  7054 ,  7060$,  7072 ,  7108 ,  7166 ,  7619$,  8398$,  8430
                  8472 ,  8563$,  8756$,  8770$
incr   = $eec7:  8880 ,  8883#
incsal = $eec1:  8546 ,  8582 ,  8739 ,  8879#, 10593
indcmp = $f7e3: 11151#, 12359
indfet = $f7d0:  5849 ,  5862 , 11097 , 11132#, 12357
indsta = $f7da: 11115 , 11141#, 12358
indx   =   $ea:   178#,  2039 ,  2115 ,  4391$,  8990$
init80 = $ce0c:  1525 ,  4911#, 11991
initst = $0a04:   307#,  1707 ,  1727 ,  1728$,  5179$,  5600 ,  5606 ,  5607$, 11717
inmi   = $0318:   245#, 12286
ins10  = $c8ef:  3638 ,  3641#
ins30  = $c8f6:  3646#,  3654 ,  3657
ins50  = $c912:  3641 ,  3660#
insert = $c8e3:  2205 ,  3420 ,  3635#
insflg =   $f6:   209#,  2201 ,  4109$, 12058
insout = $c918:  3644 ,  3661 ,  3666#
insrt  =   $f5:   208#,  2162 ,  2193 ,  2196$,  2208$,  3266$,  3317 ,  3423 ,
                  3660$,  3665$
instl1 = $e0cf:  5357#,  5380
instl2 = $e0d7:  5366#,  5369
instl3 = $e0e2:  5374#,  5377
instl4 = $e0f0:  5386#,  5389
instl5 = $e0fb:  5394#,  5397
instll = $e0cd:  5181 ,  5356#
insttb = $e105:  5357 ,  5404#
int80a = $ce23:  4927#,  4934 ,  4950 ,  4955
int80b = $ce34:  4939#,  4941
iobase = $f781: 11045#, 12402
ioidly = $e1a8:  5613#,  5614 ,  5616
ioin01 = $e142:  5542#,  5546
ioin02 = $e147:  5548#,  5553
ioin03 = $e154:  5550 ,  5556#
ioin05 = $e18a:  5587 ,  5592#
ioin06 = $e194:  5593 ,  5600#
ioinit = $e109:  5184 ,  5440#, 11699 , 12365
ioisid = $e1ae:  5601 ,  5621#
ioivic = $e170:  5572#,  5575
iokeys = $e1b8:  5631#,  8819
iopen  = $031a:   246#, 12385
iosid1 = $e1b2:  5624#,  5626
irq    = $ff17:  8894 , 12291#, 12425
irqrt1 = $c229:  1936 ,  1940 ,  1944 ,  1947 ,  1950#
irqrt2 = $c233:  1951 ,  1956#
irqrts = $c214:  1838 ,  1936#
irqtmp = $0a09:   315#,  5561$,  8052$,  8055$,  8083 ,  8106$,  8820 ,  8825
isave  = $0332:   257#, 10532
isora1 = $e388:  6228#,  6229
isour  = $e38c:  6186 ,  6235#,  6576
isoura = $e37c:  6220#,  6522 ,  6537
isr01  = $e39a:  6238 ,  6243#
isr02  = $e3a4:  6252#,  6253
```

```
isr03  = $e3a9:  6258#,  6259
isr04  = $e3ae:  6247 ,  6261#,  6263 ,  6277
isr05  = $e3c3:  6269 ,  6276#
isr06  = $e3e2:  6282 ,  6285 ,  6299#
isr07  = $e3e6:  6305#,  6307 ,  6336
isr08  = $e412:  6294 ,  6338#
isr09  = $e416:  6344#,  6353
isr10  = $e420:  6345 ,  6352#
isrclk = $e3fd:  6318 ,  6324#
isrhi  = $e3fa:  6313 ,  6323#
istop  = $0328:   253#, 12396
jcurs  = $c01b:  1521#
jdsply = $c003:  1513#
jesc   = $c01e:  1522#
jint80 = $c027:  1525#,  5603 , 12351
jkey   = $c012:  1518#, 12374
jkychk = $fd22: 12003 , 12006 , 12055 , 12096#
jkyset = $c021:  1523#, 12352
jloop5 = $c009:  1515#,  8977 ,  8992
jlp2   = $c006:  1514#,  8945
jltlk  = $f207:  9645 ,  9757#
jmpfao = $f841: 11300#
jmpfar = $02e3:   241#, 11280 , 12356
jmpirq = $ff30: 12308 , 12313#
jpcint = $c000:  1512#,  5207 , 11700 , 12364
jpfao1 = $f843: 11301#, 11306
jplot  = $c018:  1520#, 12401
jprint = $c00c:  1516#,  9112
jrad2  = $eb47:  8210 ,  8229#,  8272
jrep   = $c015:  1519#
jrsirq = $c024:  1524#, 11711
jscorg = $c00f:  1517#, 12400
jsrfao = $f82b: 11280#
jsrfar = $02cd:   240#,  5877 , 11347 , 11530 , 12355
jswap  = $c02a:  1526#, 12350
jtg10  = $ef58:  9046 ,  9061#
jtg11  = $ef5b:  9052 ,  9063#
jtg35  = $ef3c:  9011 ,  9024#
jtg36  = $ef42:  9010 ,  9032#
jtg37  = $ef45:  9005 ,  9039#
jtget  = $ef48:  9004 ,  9009 ,  9045#,  9056
jtp10  = $efa7:  9141 ,  9158#
jtp20  = $e9be:  7930#,  9045 ,  9140
jwind  = $c02d:  1527#
jx115  = $f1a9:  9661 ,  9672#
jx117  = $f1c1:  9684 ,  9691#
jx120  = $f1cf:  9658 ,  9705#
jx130  = $f1e1:  9706 ,  9710 ,  9720#
jx150  = $f1e4:  9654 ,  9657 ,  9674 ,  9693 ,  9700 ,  9715 ,  9725#
jx170  = $f200:  9729 ,  9743#
jx340  = $f138:  9551 ,  9556#
jx350  = $f13b:  9554 ,  9557#
jx600  = $f209:  9758#,  9762
jx601  = $f211:  9759 ,  9764#
jx750  = $f22f:  9791 ,  9798#
jxrmv  = $f1e5:  9666 ,  9726#
kabot2 = $cc1d:  4430 ,  4445#
kacptr = $ffa5: 12376#
```

```
kbasin = $ffcf: 11602 , 12390#
kbotcl = $ff53: 12346#
kbsout = $ffd2:  7739 , 10174 , 10182 , 10502 , 10869 , 10934 , 11464 , 11549 , 11552
                11592 , 11674 , 12391#
kc64md = $ff4d: 12344#
kchkin = $ffc6: 11599 , 12387#
kciout = $ffa8: 12377#
kckout = $ffc9: 10159 , 11545 , 11588 , 12388#
kclall = $ffe7: 12398#
kclock = $ffea: 12399#
kclose = $ffc3:  9838 , 10351 , 11542 , 11557 , 12386#
kclrch = $ffcc:  1631 , 10187 , 10787 , 10852 , 11538 , 11554 , 11596 , 11607 ,
                12389#
kclsal = $ff4a: 12343#
kdlchr = $ff62: 12351#
kdmacl = $ff50: 12345#
keybnk =   $de:   168#,  3640 ,  3656 ,  3688 ,  4451$,  4612$,  4681
keychk = $033c:   262#, 12098
keycod = $c06f:  1603#,  1713 , 11965 , 11966
keyd   = $034a:   264#,  1983 ,  1984 ,  1985$,  3080$
keydin = $fe34: 11952 , 11953 , 12205#
keyerr = $cd1f:  4638 ,  4697#
keyidx =   $d2:   126#,  1970 ,  1974$,  3108$
keyins = $ccf6:  4631 ,  4643 ,  4656 ,  4665#
keylg2 = $c62f:  2902 ,  2943#
keylog = $c5e1:  1597 ,  2841#
keyput = $c6ad:  1598 ,  3065#
keyrts = $cd1e:  4677 ,  4696#
keyset = $cca2:  1523 ,  4604#
keytab =   $cc:   122#,  2712$,  2715$,  2763 ,  2975$,  2978$,  2984 , 11971$,
                11972$, 11975
keytmp =   $df:   169#,  2338 ,  2340$,  3637 ,  3653 ,  3691 ,  4064$,  4454$,
                4679$,  4688
keyvec = $033a:   261#,  2836
kgetin = $ffe4: 12397#
kgtcfg = $ff6b:  4611 , 11309 , 12354#
kicmpv = $ff7a: 12359#
kifetv = $ff74:  4929 , 12357#
kika26 = $0a0c:   317#,  8187$,  8259
kiobas = $fff3: 12402#
kioini = $ff84: 12365#
kistav = $ff77: 12358#
kjmpfr = $ff71: 12356#
kjsrfr = $ff6e: 12355#
klistn = $ffb1: 12380#
klkupl = $ff59: 12348#
kloads = $ffd5: 12392#
kmembt = $ff9c: 12373#
kmemtp = $ff99: 12372#
kopen  = $ffc0: 10156 , 11405 , 11418 , 12385#
kount  = $0a23:   406#,  1664$,  3031$,  3037$
kpcint = $ff81: 12364#
kpfkey = $ff65: 12352#
kphenx = $ff56: 12347#
kplot  = $fff0: 12401#
kprimm = $ff7d: 12360#
kramts = $ff87: 12366#
krdtim = $ffde: 12395#
```

```
kreads = $ffb7: 12382#
kresto = $ff8a: 12367#
ksaves = $ffd8: 12393#
kscnky = $ff9f: 12374#
kscror = $ffed: 12400#
ksecnd = $ff93: 12370#
ksetmo = $ffa2: 12375#
kspio  = $ff47: 12342#
kstbnk = $ff68: 12353#
kstlfs = $ffba: 12383#
kstmsg = $ff90: 12369#
kstnam = $ffbd: 12384#
kstop  = $ffe1:  8095 ,  9942 , 10301 , 10590 , 11695 , 12396#
ksttim = $ffdb: 12394#
kswapr = $ff5f: 12350#
ktalk  = $ffb4: 12381#
ktksa  = $ff96: 12371#
kunlsn = $ffae: 12379#
kuntlk = $ffab: 12378#
kvecto = $ff8d: 12368#
kyins1 = $cd05:  4676#,  4693
kylg21 = $c643:  2850 ,  2853 ,  2963#
kyndx  =   $d1:   125#,  1638$,  1967 ,  1973$,  2007 ,  3096$,  8941
kyset1 = $ccd5:  4642#,  4649
kyset2 = $cce3:  4632 ,  4652#
kyset3 = $cce8:  4655#,  4662
la     =   $b8:   106#,  9191 ,  9202 ,  9770$, 10158 , 10349 , 10954$, 11062
lat    = $0362:   268#,  9203$,  9734 ,  9735$,  9761 ,  9769 ,  9837
lcra   = $ff01:   843#
lcrb   = $ff02:   846#
lcrc   = $ff03:   849#
lcrd   = $ff04:   852#
ld112  = $f337: 10043#, 10068
ld150  = $f344: 10044 , 10054#
ld170  = $f34b: 10047 , 10058#
ld177  = $f35a: 10073#, 10086
ld178  = $f367: 10065 , 10085#
ld179  = $f36b: 10080 , 10091#
ld26   = $f28e:  9877 ,  9882#
ld27   = $f29b:  9886 ,  9893#
ld30   = $f2cc:  9923 ,  9933#
ld40   = $f2cf:  9938#,  9952 ,  9997
ld50   = $f2fb:  9956 ,  9976#
ld60   = $f2fe:  9965 ,  9971 ,  9982#
ld64   = $f30a:  9983 ,  9993#
ldad1  = $e987:  7850 ,  7882#,  7983 ,  8007
lddisk = $f27b:  9861 ,  9869#
lddone = $f39b:  9888 , 10003 , 10132#
ldrts  = $f3a0: 10036 , 10048 , 10055 , 10062 , 10135#, 10146
ldtape = $f326:  9863 , 10029#
ldtb1  = $c04c:  1565#,  1654 ,  1790 ,  2412 ,  2456
ldtb2  = $c033:  1535#,  1782 ,  2408 ,  2452
ldtbsa = $0a3b:   457#,  1655$,  1801 ,  2414
ldtnd  =   $98:    68#,  9195 ,  9201$,  9727$,  9728 ,  9733 ,  9757 ,  9784$,  9830
                11053
lfeed  = $c9b1:  3296 ,  3820#
lines  =   $ed:   181#,  3928
linrts = $ca51:  3966 ,  3970#
```

```
lintmp = $0a30:   437#,  2022$,  2036 ,  2111
linz1  = $0400:   273#,  1535 ,  1565 ,  5010
linz10 = $0568:   282#,  1544 ,  1574
linz11 = $0590:   283#,  1545 ,  1575
linz12 = $05b8:   284#,  1546 ,  1576
linz13 = $05e0:   285#,  1547 ,  1577
linz14 = $0608:   286#,  1548 ,  1578
linz15 = $0630:   287#,  1549 ,  1579
linz16 = $0658:   288#,  1550 ,  1580
linz17 = $0680:   289#,  1551 ,  1581
linz18 = $06a8:   290#,  1552 ,  1582
linz19 = $06d0:   291#,  1553 ,  1583
linz2  = $0428:   274#,  1536 ,  1566
linz20 = $06f8:   292#,  1554 ,  1584
linz21 = $0720:   293#,  1555 ,  1585
linz22 = $0748:   294#,  1556 ,  1586
linz23 = $0770:   295#,  1557 ,  1587
linz24 = $0798:   296#,  1558 ,  1588
linz25 = $07c0:   297#,  1559 ,  1589
linz3  = $0450:   275#,  1537 ,  1567
linz4  = $0478:   276#,  1538 ,  1568
linz5  = $04a0:   277#,  1539 ,  1569
linz6  = $04c8:   278#,  1540 ,  1570
linz7  = $04f0:   279#,  1541 ,  1571
linz8  = $0518:   280#,  1542 ,  1572
linz9  = $0540:   281#,  1543 ,  1573
list1  = $e343:  6179#,  6607
list2  = $e352:  6182 ,  6189#
list3  = $e370:  6209#,  6210
list4  = $e374:  6197 ,  6213#
listn  = $e33e:  6173#,  9470 ,  9616 , 10570 , 10608 , 12380
lkupla = $f79d: 11074#, 12348
lkupng = $f79b: 11055 , 11068#, 11077
lkupok = $f791: 11060#, 11076
lkups  = $ff5c: 12349#
lkups2 = $f789: 11054#, 11058
lkupsa = $f786: 11052#, 12349
load   = $f269:  9849#, 11523
loadsp = $f265:  9846#, 12392
locabs = $ce8e:  1693 ,  5038#
lock   = $c8a6:  3297 ,  3557#
locker = $fff5:   504#,  5677 ,  5678 ,  5744$
locks  =   $f7:   214#,  2846 ,  2906 ,  2959 ,  3558 ,  3565 ,  3566$
loczp  = $ce74:  1688 ,  5010#
loding = $f533:  9933 , 10125 , 10256 , 10512#
lodng1 = $f53b: 10514 , 10517#
lookup = $f202:  9192 ,  9510 ,  9582 ,  9749#, 11075
loop2  = $c30c:  2158#,  3203 ,  3205
loop3a = $c25e:  2006#,  2008
loop4  = $c258:  2000#,  2014
loop5  = $c29b:  1515 ,  2059#
lop2   = $c319:  2163 ,  2167#
lop5   = $c2bc:  2054 ,  2067 ,  2089#
lop52  = $c2d4:  2100 ,  2105#
lop53  = $c2d8:  2103 ,  2105 ,  2108#
lop54  = $c2ce:  2097 ,  2100#
low80  = $c88b:  3513 ,  3525#
lower  = $c880:  3299 ,  3512#
```

```
lp1    = $c247:  1984#,  1989
lp2    = $c234:  1514 ,  1967#,  2011
lp3    = $c244:  1968 ,  1983#
lp70   = $c292:  2037 ,  2042#
lp75   = $c294:  2040 ,  2047#
lp80   = $c296:  2030 ,  2033 ,  2052#
lstchr =   $f0:   184#,  2159$,  3217 ,  3573 ,  4713$
lstp   =   $e9:   177#,  1772$,  2035 ,  4579$,  8972$
lstshf = $0a25:   408#,  2852 ,  2909 ,  2936$,  3043$
lstx   =   $d5:   148#,  2986 ,  3045$, 12097
lsxp   =   $e8:   176#,  1765$,  2029 ,  2287 ,  2293$,  2333 ,  2336$,  3484$,
                 3969$,  4001$,  4076$,  4583$,  8975$
luking = $f50f:  9262 ,  9884 , 10038 , 10482#
lurts  = $f532: 10483 , 10489 , 10498 , 10507#
m51ajb = $0a12:   374#,  7654 ,  7657 ,  9356$,  9357$,  9362 ,  9366
m51cdr = $0a11:   355#,  6876 ,  6935 ,  7160 ,  7219 ,  7340 ,  9378
m51ctr = $0a10:   329#,  6896 ,  7026 ,  7075 ,  9320$,  9329
matrix = $c55d:  2680#
membot = $f772: 11030#, 12373
memsiz = $0a07:   314#, 11016 , 11017 , 11022$, 11023$
memstr = $0a05:   313#, 11032 , 11033 , 11038$, 11039$
memtop = $f763: 11014#, 12372
mmucr  = $ff00:   806#,  5169$,  5335 ,  5336 ,  5358$,  5733$,  5748$, 10114 ,
                11238 , 11239$, 11243$, 11252 , 11253$, 11258$, 11267 , 11268$,
                11273$, 11293$, 11310$, 11321 , 11323$, 11324$, 12280 , 12284$,
                12299 , 12303$, 12317$, 12332$
mmucrl = $d500:   805#,  5175$
mmumcr = $d505:   861#,  1669 ,  1673$,  1743 ,  5761 ,  5820$,  6748 ,  6750$,  6757
                 6759$
mmup0h = $d508:   936#
mmup0l = $d507:   935#
mmup1h = $d50a:   941#
mmup1l = $d509:   940#
mmurcr = $d506:   902#
mmutbl = $e04b:  5174 ,  5233#
mmuver = $d50b:   945#
mod128 = $e224:  5696 ,  5725#,  5735 ,  5736 , 12410
mode   =   $d7:   150#,  1636$,  1784 ,  1793 ,  1811 ,  2403 ,  2521 ,  2914 ,  3131
                 3359 ,  3512 ,  3534 ,  4116 ,  4129 ,  4145 ,  4173 ,  4241 ,  4500
                 4740 ,  4742$,  4750 ,  4771 ,  4810 , 12063
mode1  = $fa80:  1603 ,  1607 , 11727#, 11960
mode2  = $fad9:  1604 , 11761#
mode3  = $fb32:  1605 , 11795#
mode4  = $fb8b:  1606 , 11829#, 12208
mode5  = $fbe4:  1608 , 11863#
monbrk = $b003:   494#,  5293
moncal = $b000:   493#,  5215
moncmd = $b006:   495#,  5305
movc10 = $c34c:  2218 ,  2222 ,  2230#
movc20 = $c361:  2235 ,  2249#
movc30 = $c362:  2225 ,  2232 ,  2239 ,  2254#
movchr = $c33e:  2217#,  3643 ,  4234
movl10 = $c428:  2424#,  2432
movl80 = $c436:  2404 ,  2439#
movlin = $c40d:  2310 ,  2374 ,  2403#
movos1 = $e061:  5276#,  5285
movos2 = $e068:  5277 ,  5280#
movos3 = $e06f:  5281 ,  5284#
```

```
msg    = $f722:  7731 ,  7943 ,  7953 , 10486 , 10492 , 10643 , 10864 , 10931#, 10937
msgflg =   $9d:    73#,  7727 , 10482 , 10639 , 10856 , 10922 , 11002$
msgrts = $f72f: 10926 , 10939#
msgtbl = $f6b0: 10876#, 10931
mych   =   $bf:   112#,  8321$,  8372 , 11358$, 11389 , 11503
nbasin = $ef06:  5300 ,  8966#
nbsout = $ef79:  5301 ,  9103#
nc3    = $c326:  2188#,  3332 ,  3414
nchkin = $f106:  5297 ,  9510#
nckout = $f14c:  5298 ,  9582#
nclall = $f222:  5304 ,  9783#
nclose = $f188:  5296 ,  9643#
nclrch = $f226:  5299 ,  9789#
ndx    =   $d0:   124#,  1637$,  1988 ,  1991$,  2006 ,  3039 ,  3074 ,  3082$,  8940
                10788$
newch  = $ed5a:  8061 ,  8390 ,  8608#,  8705 ,  8773
ngetin = $eeeb:  5303 ,  8935#
nirq   = $fa65:  5292 ,  8854 , 11710#
nirq1  = $fa7d: 11712 , 11719 , 11722#
njt10  = $c767:  3238 ,  3242#
nload  = $f26c:  5306 ,  9854#
nmi    = $ff05:  5366 ,  5367$, 12271#, 12415
nmi232 = $e805:  7483#, 11704
nmirti = $e849:  7539 ,  7552 ,  7562 ,  7572#
nnmi   = $fa40:  5294 , 11687#
nnmi1  = $fa5f: 11692 , 11696 , 11704#
nnmi22 = $e82d:  7529 ,  7537#
nnmi25 = $e830:  7521 ,  7534 ,  7538#
nnmi30 = $e836:  7495 ,  7544#
nnmi40 = $e841:  7549 ,  7557#
nodev  = $e428:  6240 ,  6367#
nomod1 = $e235:  5743#,  5746
nopen  = $efbd:  5295 ,  9191#
notkat = $c646:  2896 ,  2945 ,  2955 ,  2960 ,  2973#
nowhop = $c379:  2270 ,  2282#
nrmdin = $fd29: 11947 , 12002 , 12103#, 12205 , 12209 , 12210
nrmtab = $033e:   263#,  1714$,  2711 ,  2714 ,  2974 ,  2977 , 11946 , 11959 ,
                11976$, 12001
nsave  = $f54e:  5307 , 10537#
nstop  = $f66e:  5302 , 10780#
nstopx = $f67b: 10782 , 10791#
ntcn1  = $c824:  3418 ,  3423#
nunlsn = $e526:  6599#,  9793 , 10599 , 10618 , 12379
nvs    = $c328:  2183 ,  2193#
nvs2   = $c33b:  2202 ,  2212#
nvsa   = $c32e:  2194 ,  2201#
nxln   = $c363:  2259#,  3256 ,  3479 ,  3824 ,  4405
nxln1  = $c377:  2261 ,  2281#
nxt3   = $c322:  2182#,  3244
nxt33  = $c320:  2180#,  3407
nxtbit =   $b5:   103#,  6860$,  6961$,  7504 ,  8363$,  8404 ,  8424 ,  8457
nxtchr = $cbed:  2125 ,  2231 ,  3471 ,  3648 ,  3700 ,  3704 ,  4397#
nxtx1  = $c811:  3404 ,  3407#
nxtx2  = $c814:  3401 ,  3410#
nxtx3  = $c818:  3413#,  3424
nxtx4  = $c83b:  3436 ,  3441#
nxtx5  = $c842:  3442 ,  3447#
nxtx6  = $c849:  3448 ,  3453#
```

```
nxtx7  = $c850:  3454 ,  3459#
ochar  =   $bd:   110#,  6846 ,  6847$,  6883 ,  6915 ,  6957$,  8373$,  8432 ,  8481
                 8527 ,  8544 ,  8587 ,  8623 ,  8687 ,  8689$,  8698$,  8720$,
                 8729$,  8734$,  8744$,  9960$,  9964 , 10434$, 10438
oenabl = $e67f:  6999#,  7103 ,  7322 ,  7403
op155  = $effe:  9248 ,  9253#
op160  = $f017:  9273#,  9282
op170  = $f01a:  9265 ,  9279#
op171  = $f02d:  9268 ,  9280 ,  9292#
op172  = $f03c:  9296 ,  9303#
op180  = $f03f:  9260 ,  9269 ,  9281 ,  9288 ,  9306#
op200  = $f023:  9255 ,  9287#
op35   = $f0f2:  9480 ,  9492#
op40   = $f0f8:  9498#,  9502
op45   = $f103:  9493 ,  9504#
openi  = $f0cb:  9226 ,  9454#,  9901 , 10566
opn020 = $f046:  9314#,  9324
opn025 = $f055:  9315 ,  9326#
opn026 = $f072:  9342 ,  9353#
opn027 = $f078:  9348 ,  9356#
opn028 = $f07e:  9334 ,  9362#
opn050 = $f0a3:  9384 ,  9391 ,  9398#
opn232 = $f040:  9218 ,  9311#
opncas = $eff6:  9219 ,  9247#
opnier = $f0d3:  9455 ,  9460#
opniok = $f0d5:  9458 ,  9466#
opnir2 = $f0ed:  9473 ,  9485#
opnrts = $efee:  9213 ,  9222 ,  9231#
out232 = $efb7:  9136 ,  9185#
outfn  = $f521: 10497#, 10644
outfn1 = $f527: 10501#, 10505
palcnt = $0a36:   443#,  5563$, 10720$, 10724$
palnts = $0a03:   302#,  5556$,  5592 ,  9341 , 10717
pat    = $0ac1:   463#,  5833$,  5865$, 11336
patch  = $fc80:  1634 , 11905#
pause  = $0a21:   396#,  1640$,  2856 ,  2857$,  2962$,  3053$,  3199
pchi   =   $03:    31#,  5875$, 11301 , 11341$, 11525$
pclo   =   $04:    32#,  5874$, 11342$, 11496$
pcntr  =   $a3:    77#,  6185$,  6188$,  6246 ,  8209 ,  8270 ,  8317$,  8333 ,
                  8609$,  8699$,  8701
pcra   = $d501:   842#
pcrb   = $d502:   845#
pcrc   = $d503:   848#
pcrd   = $d504:   851#
phoenx = $f867: 11331#, 12347
pi     =   $de:    17#,  2126
pitk   =   $ff:    18#,  2129
pky1   = $ceb2:  5074#
pky2   = $ceb9:  5081#
pky3   = $cebf:  5087#
pky4   = $cec9:  5095#
pky5   = $ced0:  5102#
pky6   = $ced6:  5108#
pky7   = $ceda:  5114#
pky8   = $cedf:  5120#
pkybuf = $1000:   484#,  1722$,  3095 ,  3105 ,  4630 ,  4671$,  4705
pkydef = $100a:   485#,  1971 ,  4647 ,  4648$,  4658 ,  4659$,  4689$
pkyend = $cef5:  5136#
```

```
pkytbl = $cea8:  1721 ,  5066#
pl128a = $e272:  5833#,  5835
pl128b = $e27a:  5839#,  5879
pl128c = $e289:  5847#,  5856
pl128d = $e2b6:  5852 ,  5868 ,  5878#
plot   = $cc6a:  1520 ,  4547#
pnt    =   $e0:   170#,  1689$,  1788$,  1802$,  1807 ,  1810 ,  2425$,  2529$,  2555
                2560 ,  3156 ,  4249 ,  4505$,  4720 ,  4724$,  4755 ,  4759 ,  4878
                4882
pntr   =   $ec:   180#,  1771$,  2052$,  2114 ,  3154 ,  3208 ,  3254$,  3689$,  3710
                3730$,  3737 ,  3820 ,  3827$,  3909 ,  4239 ,  4347$,  4374$,  4399
                4410$,  4418 ,  4441$,  4450 ,  4495 ,  4577$,  4595 ,  4756 ,  4777
                8971
pol128 = $e26b:  5768 ,  5827#
poll   = $e242:  5183 ,  5761#
prend  = $ff33:  8281 ,  8324 ,  8379 ,  8418 ,  8592 ,  8695 ,  8746 ,  8835 , 11705
                11722 , 12316#
primm  = $fa17: 11444 , 11470 , 11651#, 12360
print  = $c72d:  1516 ,  2000 ,  2081 ,  3187#
prtcr  = $c76f:  3212 ,  3215 ,  3249#
prtx1  = $c734:  3199#,  3200
prtx2  = $c756:  3219 ,  3228#
prtx3  = $c75c:  3229 ,  3234#
prty   =   $9b:    71#,  8278 ,  8311 ,  8312$,  8614$,  8692 ,  8693$,  8742 ,
ptr1   =   $9e:    74#,  5837$,  5848 ,  5861 ,  5872 ,  7296 ,  7775$,  7802 ,
                7832$,  7837 ,  7845$,  7908$,  7918$,  7920 ,  7996$,  8491 ,
                8494 ,  8503$,  8509 ,  8566 ,  9128$,  9158 ,  9176 ,  9874$,
                9922 , 10268 , 11468$, 11487 , 11498 , 11505$, 11514
ptr2   =   $9f:    75#,  5842$,  5873 ,  7827$,  7842 ,  7846$,  7905$,  7914 ,
                7919$,  7997$,  8508 ,  8520$,  8521$,  9882$,  9893 , 10184 ,
                10204 , 11365$, 11490$, 11507
puls6  = $eee0:  8901 ,  8913#
puls7  = $eee8:  8908 ,  8922#
puls7r = $eeea:  8914 ,  8924#
putbit = $cb81:  3963 ,  4279#
putbt1 = $cb83:  2307 ,  2371 ,  4284#
qtsw   =   $f4:   203#,  2019$,  2102 ,  2148 ,  2150$,  2165$,  3268$,  3323 ,  3410
qtswc  = $c2ff:  2108 ,  2145#,  3243
qtswl  = $c30b:  2146 ,  2153#
r8502  =   $01:   522#,  1624 ,  1629$,  1860 ,  1894 ,  1931$,  2686 ,  5524$,
                5781$,  7959 ,  7962 ,  8063 ,  8065$,  8638 ,  8640$,  8860 ,
                8862$,  8899 ,  8906 ,  8916 ,  8922$
rad2   = $ebcf:  8229 ,  8327#
rad2x  = $ebde:  8331 ,  8334 ,  8339#
rad2y  = $ebd7:  8328 ,  8333#
rad3   = $eb94:  8253 ,  8284#
rad4   = $ebb6:  8271 ,  8296 ,  8310#
rad5   = $eb58:  8240 ,  8243#
radbk  = $eb91:  8257 ,  8264 ,  8281#,  8315
radj   = $ec1f:  8206 ,  8390#
radk   = $ec12:  8364 ,  8372#
radkx  = $eb7b:  8261 ,  8266#
radl   = $eb56:  8222 ,  8242#,  8336
radq2  = $ec03:  8349 ,  8362#
radx2  = $eb52:  8216 ,  8239#
rambot = $1c00:   487#,  5339 ,  5340
ramtas = $e093:  5201 ,  5312#, 12366
ramtz0 = $e096:  5318#,  5320
```

```
rblk   = $e9f2:  7707 ,  7979#,  9051
rd10   = $ec49:  8409 ,  8413 ,  8418#,  8425 ,  8428 ,  8443 ,  8451 ,  8455
rd12   = $ec45:  8405 ,  8415#
rd15   = $ec2f:  8396 ,  8400#
rd160  = $ed0a:  8469 ,  8549#
rd161  = $ed0c:  8463 ,  8550#
rd167  = $ed1e:  8559 ,  8563#
rd175  = $ed2a:  8564 ,  8573#
rd180  = $ed4e:  8547 ,  8567 ,  8570 ,  8588 ,  8592#
rd20   = $ec4c:  8402 ,  8421#
rd200  = $ec68:  8422 ,  8442#
rd22   = $ec62:  8433 ,  8437#
rd300  = $ed51:  8448 ,  8574 ,  8597#,  8784 , 10576
rd40   = $ec79:  8434 ,  8437 ,  8453#
rd52   = $ecf0:  8523 ,  8533#
rd55   = $ecf4:  8492 ,  8536#
rd58   = $ecc8:  8474 ,  8508#
rd59   = $ecfb:  8488 ,  8505 ,  8534 ,  8540#
rd60   = $ec7f:  8421 ,  8457#
rd70   = $ec8d:  8458 ,  8466#
rd71   = $ec95:  8467 ,  8472#
rd80   = $ecab:  8477 ,  8482 ,  8487#
rd90   = $ed05:  8510 ,  8514 ,  8518 ,  8528 ,  8538 ,  8541 ,  8546#
rdbk   = $eb6b:  8201 ,  8233 ,  8236 ,  8256#,  8302 ,  8305 ,  8308
rdbk2  = $ec1c:  8352 ,  8379#
rdtim  = $f65e: 10761#, 12395
read   = $eaeb:  8170#,  8175 ,  8855
readss = $f744: 10972#, 12382
readst = $f755: 10974 , 10990#
recerr = $e712:  7129 ,  7189#
redit1 = $f9dc: 11591#, 11594
redit2 = $f9ef: 11602#, 11605
repeat = $c651:  1519 ,  2873 ,  2907 ,  2910 ,  2937 ,  2983#
reset  = $ff3d: 12331#, 12420
restor = $e056:  5206 ,  5265#, 11698 , 12367
rewrap = $ca32:  3923 ,  3944#
ribuf  =   $c8:   120#,  5327$,  5328$,  7149$,  7433
ridata =   $aa:    84#,  7061$,  7134 ,  7210 ,  7993$,  8401 ,  8416$,  8440$,
                8442$,  8446$,  8454$,  8550$
ridbe  = $0a18:   388#,  7123 ,  7131$,  7425 ,  9398
ridbs  = $0a19:   389#,  7128 ,  7424 ,  7434$,  9399$
rinone =   $a9:    83#,  7045 ,  7097$,  7113$,  8239$,  8242$,  8301 ,  8376 ,
                8615$,  8670 ,  8676$, 11452$
riprty =   $ab:    85#,  7055 ,  7056$,  7116$,  7167 ,  7852$,  8013$,  8450$,
                8576$,  8579 ,  8580$,  8586 ,  8774$,  8780$
rjdj   = $eb28:  8204 ,  8209#
robuf  =   $ca:   121#,  5331$,  5332$,  6972 ,  7297$
rodata =   $b6:   104#,  6836$,  6973$,  8377$,  8427 ,  8485$,  8487 ,  8531$,  8533
                8649$,  8664 ,  8788$
rodbe  = $0a1b:   391#,  6967 ,  7289 ,  7294$,  9401
rodbs  = $0a1a:   390#,  6966 ,  6975$,  7291 ,  9402$
rout1  = $eb9f:  8285 ,  8292#
rout2  = $eb9d:  8286 ,  8291#
rows   = $dc01:  1146#,  2384 ,  2700 ,  2751 ,  2752 , 10730 , 10731 , 10746 , 10747
rpt10  = $c661:  2987 ,  2999#
rpt20  = $c67e:  3002 ,  3011 ,  3014 ,  3017 ,  3025#
rpt40  = $c688:  3026 ,  3031#
rptflg = $0a22:   401#,  1667$,  3001
```

```
rsibuf = $0c00:   481#,  5326
rsobuf = $0d00:   482#,  5330
rsodne = $e67d:  6968 ,  6994#
rsp232 = $e7ec:  6174 ,  7449#,  8039
rspext = $e62b:  6884 ,  6894#,  6916 ,  6922
rspno  = $e636:  6877 ,  6909#
rspoff = $e7f2:  7457#,  7463
rspok  = $e803:  7452 ,  7477#
rsr018 = $e6b2:  7067#,  7161
rsr020 = $e6b4:  7050 ,  7072#
rsr030 = $e6df:  7049 ,  7123#
rsr031 = $e6f1:  7139#,  7144
rsr032 = $e6f9:  7140 ,  7149#
rsr050 = $e70d:  7168 ,  7179#
rsr060 = $e723:  7073 ,  7210#
rsrabl = $e6c2:  7088#,  7109 ,  7205
rsrcvr = $e69d:  7045#,  7649
rsrext = $e6b1:  7062#,  7083 ,  7162 ,  7172 ,  7179
rsrtrt = $e6d4:  7046 ,  7108#
rsstat = $0a14:   375#,  6988 ,  6989$,  7202 ,  7203$,  7276$,  7422 ,  7431$,
                 7441$,  9091 ,  9312$, 10978 , 10982$
rst005 = $e60c:  6838 ,  6845#
rst010 = $e61b:  6850 ,  6867#
rst030 = $e63a:  6879 ,  6915#
rst040 = $e640:  6878 ,  6922#
rst050 = $e644:  6832 ,  6928#
rst060 = $e657:  6941 ,  6956#
rstbgn = $e64a:  6831 ,  6935#,  7323
rstext = $e615:  6855#,  6901 ,  6904 ,  6930
rstir1 = $c1a3:  1830 ,  1838#
rstir2 = $c1ae:  1841 ,  1846#
rstir3 = $c1bd:  1852 ,  1860#
rstir4 = $c1da:  1876 ,  1884#
rstirq = $c194:  1524 ,  1826#
rstoa  = $efab:  9165#,  9186
rstor  = $efac:  9146 ,  9170#
rstor1 = $efb6:  9177 ,  9180#
rstrab = $e5ff:  6830#,  7538
rswext = $e62a:  6889#,  6910 ,  6917 ,  6923
runtb  = $cee7:  5127#
rvs    =   $f3:   198#,  2182 ,  3267$,  3592$
rvsoff = $c8bf:  3438 ,  3585#
rvson  = $c8c2:  3302 ,  3591#
sa     =   $b9:   107#,  9205 ,  9207$,  9253 ,  9294 ,  9454 ,  9475 ,  9530 ,
                 9550 ,  9600 ,  9621 ,  9672 ,  9691 ,  9712 ,  9773$,  9873 ,
                 9897$,  9904 , 10085 , 10376$, 10564$, 10572 , 10604 , 10610 ,
                10659 , 10671 , 10956$, 11063
sah    =   $ad:    87#,  8495 ,  8516 ,  8598$,  8727$,  8872 ,  8882$, 10284$, 10579
                11427$, 11483$
sal    =   $ac:    86#,  8498 ,  8512 ,  8601$,  8870 ,  8879$, 10283$, 11105 , 11126
                11426$
sally  = $f7cc:  8480 ,  8526 ,  8578 ,  8733 , 10587 , 11126#
salsy  = $f7bc:  8545 , 11105#, 11603
sat    = $0376:   270#,  9209$,  9740 ,  9741$,  9772 , 11057
sav80a = $0a31:   438#,  2439$,  2451 ,  2507 ,  2542$,  2619
sav80b = $0a32:   439#,  2440$,  2482 ,  2483$,  2503 ,  2543$,  2573
savesp = $f53e: 10522#, 12393
saving = $f5bc: 10567 , 10639#, 10655
```

```
savpos = $cc1e:  3635 ,  3672 ,  4007 ,  4022 ,  4038 ,  4058 ,  4071 ,  4450#
scatn  = $e4d7:  6527#,  6547 ,  6608 ,  9624
scbot  =   $e4:   172#,  1756 ,  2221 ,  2260 ,  2294 ,  2325 ,  2364 ,  3920$,  4049
                 4532 ,  4555
scd10  = $c388:  2295#,  2311
scd20  = $c3a0:  2299 ,  2314#
scd30  = $c386:  2288 ,  2291 ,  2294#
sclf   =   $e6:   174#,  1770 ,  2028 ,  2297 ,  2362 ,  2513 ,  3253 ,  3939$,  4013
                 4025 ,  4044 ,  4346 ,  4382 ,  4406 ,  4422 ,  4538 ,  4563 ,  4596
scn10  = $c57d:  2702 ,  2709#
scn20  = $c58e:  2726#,  2816 ,  2827
scn21  = $c597:  2730 ,  2740#
scn22  = $c59a:  2735 ,  2745#
scn24  = $c59d:  2751#,  2753
scn30  = $c5a5:  2758#,  2804
scn35  = $c5b7:  2768 ,  2779#
scnfk1 = $fc9f: 11942 , 11959#
scnfk2 = $fcaa: 11954 , 11971#
scnfk3 = $fcb0: 11975#, 11978
scnfk4 = $fcc3: 11932 , 11948 , 11961 , 11996#
scnfky = $fc87:  1518 ,  1953 , 11927#
scnkey = $c55d:  2686#, 11996
scnrts = $c6c4:  3003 ,  3020 ,  3029 ,  3032 ,  3041 ,  3048 ,  3076 ,  3087#
scold  = $e031:  5187 ,  5201#
scolor = $c17c:  1807#,  3165 ,  4493 ,  4776
scr10  = $c3dc:  2342 ,  2361#,  2375
scr60  = $c3f4:  2365 ,  2380#
scrdwn = $c37c:  2243 ,  2287#,  3956 ,  4080
scroll =   $f8:   219#,  2224 ,  2238 ,  2263 ,  2353 ,  3983 ,  3987$,  3992$,
                 4095$,  4307
scrorg = $cc5b:  1517 ,  4531#
scrset = $c15e:  1753 ,  1782#,  2295 ,  2361 ,  2520
scrst1 = $c166:  1785 ,  1788#
scrst2 = $c177:  1794 ,  1801#
scrst3 = $c17a:  1798 ,  1802#
scrst5 = $c18d:  1812 ,  1818#
scrst6 = $c191:  1816 ,  1820#
scrsw0 = $cae2:  3883 ,  4087#
scrsw1 = $cae5:  3884 ,  4094#
scrt   =   $e7:   175#,  2217 ,  2430 ,  2479 ,  2534 ,  2572 ,  2620 ,  3675 ,  3716
                 3729 ,  3921$,  4373 ,  4400 ,  4440 ,  4443 ,  4537 ,  4566 ,  8989
scru00 = $c3a8:  2321#,  2326
scru10 = $c3db:  2351 ,  2356#
scru15 = $c3b8:  2323 ,  2331#,  3989
scru20 = $c3c0:  2334 ,  2337#
scru30 = $c3c8:  2339 ,  2342#
scrup  = $c3a6:  2275 ,  2320#,  2354 ,  4062
sctop  =   $e5:   173#,  1763 ,  2268 ,  2320 ,  2328 ,  2337 ,  2344 ,  3491 ,
                 3938$,  3977 ,  3981$,  3994 ,  3998$,  4077 ,  4428 ,  4533 ,
                 4552 ,  4592
sddn   = $caca:  3894 ,  4071#
sddn2  = $cad5:  4073 ,  4077#
secnd  = $e4d2:  6520#,  9477 ,  9627 , 10573 , 10613 , 12370
secure = $e1f0:  5182 ,  5677#
sedeah =   $dd:   167#,  2419$,  4625$,  4642 ,  4652 ,  4654
sedeal =   $dc:   166#,  2409$,  2427 ,  4605$,  4622 ,  4627 ,  4665 ,  4669
sedsah =   $db:   165#,  2415$,  2460$,  2469 ,  2470$,  2492 ,  4618$,  4637 ,  4641
                 4655 ,  4783$,  4789 ,  4914$,  4951$,  4953
```

```
sedsal =   $da:  164#,  2410$,  2424 ,  2454$,  2465 ,  2466$,  2475 ,  2498 ,  3739
                3760$,  3768 ,  4301 ,  4325$,  4334 ,  4606$,  4628 ,  4670 ,
                4676$,  4913$,  4928 ,  4946 ,  4948$
serial = $0a1c:  392#,  5635$,  6274$,  6284 ,  6374 ,  6376$,  6499$,  6602 ,
                6604$,  9869 ,  9871$, 10189
setal1 = $ca36: 3947#,  3949
setbit = $cb93: 2315 ,  4284 ,  4307#
setbnk = $f73f: 10963#, 11394 , 11520 , 12353
setbot = $f77a: 5341 , 11030 , 11038#
setc41 = $c71f: 3159 ,  3175#
setc42 = $c72c: 3132 ,  3135 ,  3138 ,  3147 ,  3182#
sethtb = $ca16: 3873 ,  3908#
sethtt = $ca14: 3891 ,  3902#
setlfs = $f738: 10153 , 10954#, 11404 , 11416 , 12383
setmsg = $f75c: 11002#, 12369
setnam = $f731: 10945#, 11399 , 11411 , 11516 , 12384
settab = $c961:  3305 ,  3737#
settim = $f665: 10769#, 12394
settmo = $f75f: 11008#, 12375
settop = $f76b:  5337 , 11014 , 11022#
sfdx   =   $d4:   147#,  2694$,  2795$,  2835 ,  2953 ,  2983 ,  3044
sftcr  =   $8d:    15#,  3214 , 10916
sftspc =   $a0:    16#, 10881 , 10885 , 10906 , 10913
shfchr = $c802:  3231 ,  3394#
shfdin = $fd82: 12137#, 12206
shflag =   $d3:   128#,  2691$,  2726 ,  2779 ,  2787$,  2822$,  2824$,  2830$,
                2831$,  2864 ,  3056 , 11937 , 12096
shflg1 = $c5f8:  2842 ,  2864#
shflg2 = $c620:  2915 ,  2928#
shflg3 = $c628:  2859 ,  2923 ,  2935#
shfvec = $0336:   259#,  3394
shiftd = $c805:  1595 ,  3399#
sid    = $d400:   700#,  5624$
sidad1 = $d405:   738#,  3802$
sidct1 = $d404:   725#,  3810$,  3813$
sidenv = $d41c:   792#
sidf1l = $d400:   704#
sidf2h = $d401:   708#,  3807$
sidptx = $d419:   780#
sidpty = $d41a:   784#
sidrnd = $d41b:   788#
sidsr1 = $d406:   745#,  3804$
sidt1h = $d403:   718#
sidt1l = $d402:   714#
sidvol = $d418:   770#,  3798$, 11920$
simirq = $eec8:  8161 ,  8888#
skip1  =   $24:    20#,  3903 , 10127 , 10679
skip2  =   $2c:    21#,  1879 ,  2788 ,  3241 ,  3586 ,  3724 ,  3775 ,  4088 ,  4103
                4203 ,  4472 ,  4478 ,  6168 ,  6362 ,  6594 ,  6982 ,  7174 ,  7184
                7190 ,  7196 ,  8289 ,  9822 , 10798 , 10804 , 10810 , 10816 , 10822
               10828 , 10834 , 10840 , 10846 , 11106 , 11121 , 11488
slorts = $c40c:  2386 ,  2398#
slowb  = $e5bc:  6203 ,  6293 ,  6736#,  6738
slowmo = $c405:  2391#,  2393 ,  2396
smod   = $e041:  5211 ,  5220#
snsw1  =   $b4:    94#,  6830 ,  6849$,  6894$,  6903$,  6909$,  6928$,  6964$,
                7994$,  8232 ,  8256 ,  8314 ,  8330 ,  8348 ,  8360$,  8367$
spck2  = $c5bc:  2771 ,  2774 ,  2795#
```

```
speed  = $0a37:   444#,  5559$,  6679 ,  6687$,  6717 ,  6723 ,  6727$,  8806 ,  8812
                 8816$
spinot = $e5fb:  6793#, 12342
spinp  = $e5c3:  5649 ,  5651 ,  6743#,  6793 , 10209
split  = $0a34:   441#,  1683$,  1854
spmsg  = $f71e: 10517 , 10922#
spout  = $e5d6:  5650 ,  6199 ,  6289 ,  6757#,  6794
sprits = $0a38:   445#,  6690$,  6697 ,  6720 ,  8809
sreg   =   $05:    33#, 11287$
srer   = $eb4a:  8227 ,  8232#,  8279 ,  8299
sreset = $ca24:  1734 ,  1740 ,  3579 ,  3928#
stack  = $0100:   231#,  8496$,  8499$,  8513 ,  8517 , 11384$, 11390$, 11574$,
                11576$, 11580$, 11582$, 11591 , 11661$, 11664$, 11665 , 11668 , 12306
stah   =   $c2:   115#,  7780 ,  7813 ,  7866$,  7891$,  8597 , 10123$, 10530$,
                11366$, 11563 , 11570$
stal   =   $c1:   114#,  7783 ,  7809 ,  7863$,  7884$,  8600 , 10120$, 10527$,
                11369$, 11569$, 11578
start  = $e000:  5156#, 12334
stash  = $02af:   237#, 11146
stasho = $f80d: 11250#
stashx = $02b9:   238#, 11112$
status =   $90:    39#,  6508 ,  6539 ,  7980$,  9068 ,  9467$,  9472 ,  9479 ,  9547
                 9559 ,  9618 ,  9630 ,  9750$,  9857$,  9917 ,  9939 ,  9940$,  9949
                 9993 , 10060 , 10990 , 10995 , 10996$
stbot  = $ca1d:  3920#,  3930
stdeft = $c983:  1666 ,  1733 ,  3780#,  3896
stdft1 = $c987:  3783#,  3785
stdft2 = $c98d:  3787#,  3793
stgref = $fe45: 12012 , 12024 , 12221#
stkey  =   $91:    52#,  7751 , 10755$, 10780
stkptr =   $09:    37#, 11290$
stky   = $ee95:  8782 ,  8834#
stop3  = $ea9b:  8019 ,  8086 ,  8105#
stop4  = $eaa0:  8097 ,  8108#
stt1   = $eaa1:  8113#,  8276 ,  8323 ,  8346 ,  8394
stt2   = $eab6:  8126 ,  8129#
stt3   = $eabd:  8136#,  8138
stt4   = $eae9:  8153 ,  8155 ,  8158 ,  8164#
sttop  = $ca2e:  3915 ,  3938#
stu10  = $c156:  1770#,  3957 ,  4002
stupid = $0a0d:   318#,  8149$,  8263 ,  8268$
stupt  = $c15c:  1777#,  2089 ,  2230 ,  2282 ,  3497 ,  3694 ,  4011 ,  4042 ,  4361
                 4371 ,  4438 ,  4585
stvdc1 = $e1ee:  5658 ,  5671#
suup   = $cabc:  3893 ,  4058#
sv106  = $f5df: 10661 , 10664#
sv110  = $f5f6: 10673 , 10681#
sv30   = $f586: 10582#, 10594
sv31   = $f59b: 10583 , 10599#
svdisk = $f561: 10542 , 10560#
svrts  = $f5f7: 10640 , 10653 , 10666 , 10669 , 10682#
svtape = $f5c8: 10539 , 10649#
svxt   =   $92:    57#,  8246 ,  8247$,  8284 ,  8293$,  9643$,  9705
swap   = $cd2c:  3895 ,  4713#
swapm2 = $cd30:  4719#,  4727
swapm3 = $cd40:  4730#,  4738
swappr = $cd2e:  1526 ,  1732 ,  1736 ,  1746 ,  4718#
swarm  = $e034:  5197 ,  5206#
```

```
swpmap = $0a60:   461#,  4730 ,  4733$
swpout = $0a40:   460#,  1694$,  4719 ,  4722$
swrts  = $cd56:  4744#,  4751
syno   =   $96:    66#,  8307$,  8327 ,  8351 ,  8356$,  8362
sysirq = $fffe: 12425#
sysnmi = $fffa:  5374 ,  5375$, 12415#
sysres = $fffc:  5822 , 12420#
syssr1 = $e00c:  5174#,  5177
system = $fff8:  5703 ,  5704 ,  5737$,  5738$, 12410#
sysvec = $0a00:   299#,  5223 ,  5345$,  5346$, 11701
t2nmi  = $e878:  7533 ,  7551 ,  7614#
tabclr = $c980:  3774#,  3897
tabget = $c96c:  3719 ,  3738 ,  3756#
tabmap = $0354:   265#,  3740$,  3767 ,  3783$,  4731 ,  4735$
talk   = $e33b:  6167#,  9545 ,  9903 , 12381
tapadr = $e980:  7777 ,  7874#,  7882 ,  7930 ,  9247 ,  9678 , 10032 , 10649
tape   = $ea26:  8002 ,  8024#
tape1  =   $b2:    92#,  5323$,  5324$,  7713 ,  7738 ,  7796$,  7803$,  7810$,
                 7814$,  7820$,  7824$,  7843$,  7874 ,  7875 ,  7915 ,  9061 ,
                 9150$,  9159$,  9300$, 10074 , 10078 , 10092 , 10095 , 10099 , 10102
tapeh  = $e919:  7775#,  9291 ,  9699 , 10665 , 10678
tapirq = $eed0:  8899#, 11715
tblx   =   $eb:   179#,  1764$,  1777 ,  2032 ,  2047$,  2110 ,  2220 ,  2259 ,
                 2269$,  2281$,  2290 ,  2298 ,  2331$,  3492 ,  3496$,  3506$,
                 3652 ,  3678 ,  3692$,  3910 ,  3980 ,  3995$,  4010$,  4041$,
                 4048 ,  4078$,  4266 ,  4279 ,  4355$,  4360$,  4366$,  4370$,
                 4377 ,  4429 ,  4435$,  4453 ,  4582$,  4591 ,  8974
tbuffr = $0b00:   480#,  5322 , 11424 , 11425 , 11436 , 11451 , 11461 , 11492 ,
                11500$, 11504$, 11515 , 11524
tcolor =   $f2:   197#,  4247$,  4257$,  4483
temp   =   $b1:    91#,  8113$,  8121 ,  8122$,  8129$,  8132$,  8140 ,  8177$,  8190
                 8191$,  8194$,  8196$,  8200 ,  8215 ,  8221 ,  8226 ,  8245 ,
                 8339$,  8342
text   = $c1df:  1844 ,  1849 ,  1891#
text1  = $c201:  1918#,  1919
th20   = $e954:  7837#,  7847
th30   = $e967:  7839 ,  7850#
th40   = $e97f:  7778 ,  7869#
time   =   $a0:    76#,  7745 ,  7755 , 10687$, 10690$, 10693$, 10695 , 10697 , 10699
                10704$, 10705$, 10706$, 10762 , 10763 , 10764 , 10770$, 10771$,
                10772$
timer  = $0a1d:   393#, 10708 , 10711 , 10714$, 10715$, 10716$
timout = $0a0e:   319#, 11008$
tkatn  = $e4e9:  6545#,  9553
tkatn1 = $e4f5:  6553#,  6555 ,  6558
tksa   = $e4e0:  6536#,  9556 ,  9905 , 12371
tmp2   =   $c3:    116#,  5272$,  5273$,  5279 ,  5282$,  5679$,  5680$,  5682 ,
                 5705$,  5706$,  9846$,  9847$,  9927 ,  9930 , 10075$, 10079$,
                10107 , 10111 , 10119 , 10122 , 10273 , 10274
tnif   = $ee57:  8101 ,  8573 ,  8795#,  8834
tnif1  = $ee7f:  8804 ,  8807 ,  8818#
tniq   = $ee93:  8821 ,  8828#
tnof   = $eeb0:  8754 ,  8818 ,  8860#
tod10  = $dc08:  1179#
todhr  = $dc0b:  1201#
todmin = $dc0a:  1194#
todsec = $dc09:  1187#
toqm   = $c77d:  3261#,  3843 ,  3886
```

```
tp32   = $ea74:  8072#,  8076
tp35   = $ea76:  8073#,  8074
tp40   = $ea7d:  8083#,  8090
trd    = $e9fb:  7988#, 10126
tstmo1 = $e1ff:  5688#,  5699
tstmo2 = $e216:  5709#,  5718
tstop  = $ea8f:  7948 ,  8088 ,  8095#
twrt   = $ea18:  8012#, 10668
twrt2  = $ea1c:  7854 ,  8018#
twrt3  = $ea1f:  7989 ,  8019#
ud30   = $f602: 10688 , 10691 , 10694#
ud31   = $f619: 10701 , 10708#
ud32   = $f626: 10712 , 10715#
ud33   = $f629: 10709 , 10716#
ud60   = $f63d:  5185 ,  8089 , 10300 , 10718 , 10721 , 10730#, 10732 , 11694
ud70   = $f64d: 10746#, 10748
ud80   = $f65b: 10738 , 10755#
ud90   = $f65d: 10753 , 10756#
udst   = $f757:  6379 ,  6432 ,  8412 ,  8461 ,  8537 ,  8591 ,  9019 ,  9970 , 10341
                10360 , 10444 , 10995#
udtim  = $f5f8: 10687#, 10725 , 11714 , 12399
undrof = $c8ce:  3444 ,  3608#
undron = $c8c7:  3293 ,  3599#
unlock = $c8ac:  3298 ,  3564#
unlrts = $c8b0:  3559 ,  3566#
untlk  = $e515:  6586#,  9801 , 10001 , 12378
upp80  = $c89f:  3535 ,  3548#
upper  = $c892:  3433 ,  3534#
uprts  = $c89b:  3519 ,  3541#
user   =   $e2:   171#,  1808$,  1820$,  2428$,  2532$,  2595 ,  2599 ,  3166 ,  4246
                 4507$,  4896 ,  4900
vdcadr = $d600:   973#,  2635 ,  4847$,  4849 ,  4864$,  4866 ,  5582 ,  5664$
vdcchr = $2000:   971#,  4917 ,  4921
vdccol = $0800:   970#,  1651 ,  5039 ,  6092 ,  6093
vdcdat = $d601:   996#,  4852$,  4869 ,  5665$
vdcget = $cdd8:  4256 ,  4260 ,  4779 ,  4858#
vdcin  = $cdda:  2445 ,  2548 ,  2639 ,  2644 ,  4218 ,  4226 ,  4864#, 12070
vdcint = $e1dc:  5580 ,  5590 ,  5596 ,  5657#,  5666
vdcout = $cdcc:  2448 ,  2472 ,  2476 ,  2486 ,  2495 ,  2499 ,  2505 ,  2550 ,  2562
                 2566 ,  2600 ,  2604 ,  2633 ,  4228 ,  4762 ,  4766 ,  4797 ,  4821
                 4847#,  4884 ,  4888 ,  4902 ,  4906 ,  4918 ,  4922
vdcput = $cdca:  2569 ,  2613 ,  4522 ,  4526 ,  4790 ,  4817 ,  4842#,  4930 ,  4939
vdcscn =   $00:   969#,  5038 ,  6084 ,  6085 ,  6088 ,  6089
vdcsi  = $cddd:  4866#,  4867
vdcso  = $cdcf:  4849#,  4850
vdctbl = $e2f8:  5657 ,  5661 ,  6047#
vectab = $c065:  1594#,  1702
vector = $e05b:  5272#, 12368
vectss = $e073:  5265 ,  5266 ,  5292#
verck  =   $93:    59#,  7704 ,  7709$,  7981$,  8476 ,  8522 ,  8540 ,  9854$,  9955
                10428 , 10513
vic    = $d000:   575#,  5573$
viccr1 = $d011:   587#,  1840 ,  1868 ,  1903 ,  1933$,  1946 ,  5542 ,  5552 ,  8041
                 8048$,  8798 ,  8801$
viccr2 = $d016:   609#,  1873 ,  1910 ,  1934$
vicier = $d01a:   634#,  5632$,  8025$
vicifr = $d019:   625#,  1828 ,  1834$,  8028$
vicmcr = $d018:   619#,  1928$
```

```
vicmoe = $d015:   605#,  6689 ,  6693$,  6721$,  8810$
vicpen = $d02f:   681#,  2698$,  2719$,  2740$
vicrwr = $d012:   596#,  1855$,  1892$,  5549
vicspd = $d030:   686#,  1938 ,  5804$,  6682 ,  6695$,  6724$,  8813$
victbl = $e2c7:  5572 ,  5903#
vm1    = $0a2c:   433#,  1646$,  1900 ,  2928 ,  2930$,  3517 ,  3539 ,  3541$
vm2    = $0a2d:   434#,  1649$,  1865
vm3    = $0a2e:   435#,  1643$,  1797 ,  2459
vm4    = $0a2f:   436#,  1652$,  1815 ,  2494 ,  2608
vprty  = $ed34:  8578#,  8584
waitky = $c25b:  2005#,  2066
wblk   = $ea15:  8007#,  9145 ,  9683
wiggle = $f503: 10212 , 10236 , 10260 , 10264 , 10290 , 10321 , 10329 , 10336 ,
                10468#
window = $ca1b:  1527 ,  3915#
wrend  = $ee22:  8752 ,  8755#
write  = $ed69:  8623#,  8679
wrnc   = $ee1b:  8667 ,  8751#
wrt1   = $ed72:  8626 ,  8629#,  8767
wrt2   = $edcf:  8685 ,  8698#
wrt3   = $edcc:  8650 ,  8661 ,  8665 ,  8674 ,  8677 ,  8680 ,  8695#,  8703 ,  8721
                 8730 ,  8740
wrt4   = $ee12:  8702 ,  8742#
wrt6   = $edf3:  8709 ,  8723#
wrt61  = $edef:  8717 ,  8720#
wrt7   = $ee02:  8724 ,  8732#
wrtbk  = $ee18:  8746#,  8761 ,  8768 ,  8771 ,  8775
wrtl3  = $ed8b:  8648#,  8725
wrtn   = $ed90:  8655#,  8853
wrtn1  = $eda6:  8656 ,  8670#
wrtn2  = $edb3:  8671 ,  8679#
wrts   = $edd9:  8705#,  8789
wrtw   = $ed70:  8628#,  8673
wrtx   = $ed74:  8630#,  8660
wrtz   = $ee2e:  8766#,  8852
xmax   = $0a20:   394#,  1658$,  3075
xreg   =   $07:    35#, 11282$, 11313
xsav   =   $97:    67#,  8956$,  8958 ,  9002$,  9025 ,  9039
yreg   =   $08:    36#, 11283$, 11314
```

Anhänge

A Einführung in den Mikroprozessor 65xx bzw. 8502

A.1 Allgemeines

Im Rechner C-128 wird von Commodore der 8502 als Hauptprozessor verwendet. Er wird im C-128-Modus mit einer Taktfrequenz von wahlweise 1 MHz oder 2 MHz (doppelte Geschwindigkeit) und im C-64-Modus ausschließlich mit einer Taktfrequenz von 1 MHz betrieben. Im CP/M-Modus ist zusätzlich der eingebaute Prozessor Z80 in Betrieb. Die folgenden Ausführungen beziehen sich auf die 8502-CPU, die befehlskompatibel zur weitverbreiteten 6502-CPU ist. Das heißt, die beiden Prozessoren haben den gleichen Befehlsvorrat, Maschinencode, die gleiche Syntax und die gleichen Adressierungsarten. Die Unterschiede in der Belegung und Handhabung der Zero-Page Adressen "00" und "01" werden später erläutert.

A.2 Architektur und Organisation vom 65xx und 8502

Die CPU besteht im wesentlichen aus folgenden Funktionsblöcken:

Name	Bits	
Programmzähler (Program Counter)	16	(Low-Byte, High-Byte)
Akkumulator (Hauptregister)	8	
Index-Register X	8	
Index-Register Y	8	
Stack-Pointer - Zeiger in Stack	8	(Low-Byte)
Status-Register	8	
ALU = Arithmetic Logic Unit		
Interrupt-Logik (RES, IRQ, NMI)		
Timer		

A.2.1 PC - Programmzähler (Program Counter)

Der Programmzähler enthält die aktuelle Adresse, die der Prozessor gerade bearbeitet. Jedesmal, wenn die CPU einen Befehl aus dem Programmspeicher holt (ROM/RAM), wird der Wert des 16-Bit Programmzählers auf den Adress-Bus ausgegeben. Anschließend erhöht die CPU automatisch den PC

um einen Schritt (PC wird incrementiert) und kann dann die nächste Adresse bearbeiten.

A.2.2 AC - Akkumulator

Das wichtigste Register einer CPU ist der Akkumulator (8 Bit). In ihm werden alle logischen und arithmetischen Operationen durchgeführt. Außerdem enthält er vor diesen Operationen eines der beiden Operations-Bytes. Viele der später beschriebenen Prozessor-Befehle beziehen sich direkt auf den Akkumulator.

A.2.3 XR / YR - Index-Register X und Y

Die beiden in der CPU vorhandenen 8-Bit Index-Register X und Y können als schnell erreichbare Zwischenspeicher (ohne Zugriff auf externe RAMs) verwendet werden. Außerdem spielen sie bei den später besprochenen verschiedenen Adressierungsarten eine große Rolle.

A.2.4 SP - Stapel-Zeiger (Stack Pointer)

Wie die Zero-Page, ist auch der im RAM-Bereich von hexadezimal $0100 bis $01FF, der Stack, für den Prozessor von besonderer Bedeutung. Die CPU legt dort beispielsweise beim Sprung in eine Unterroutine die Rücksprungadresse bzw. bei einem Interrupt die aktuellen Inhalte der CPU-Register ab (Register retten). Der reservierte Speicherbereich umfaßt 256 Bytes. Der 8-Bit breite Stapelzeiger (Wert $00 bis $FF) zeigt auf die aktuelle Stapeladresse.

A.2.5 SR - Status-Register

Das Status-Register besteht aus acht Flip-Flops, wobei jedes Bit eine bestimmte Bedeutung hat. Die Status-Bits werden von der CPU in Abhängigkeit von den ausgeführten Operationen gesetzt bzw. zurückgesetzt. Einige Bits können auch mit Hilfe spezieller CPU-Befehle durch den Programmierer beeinflußt werden. Die einzelnen Bits werden auch als Flags bezeichnet. Die Carry-, Zero-, Overflow- und Negativ-Flags können als sogenannte Bedingungs-Bits im Status-Register, abhängig von ihrem Wert (0 oder 1), bei Verzweigungsbefehlen (Branch-Befehlen) den weiteren Ablauf beeinflussen.

Lage der einzelnen Flags im Status-Byte:

```
Bit7  Bit6  Bit5  Bit4  Bit3  Bit2  Bit1  Bit0
 N     V     E     B     D     I     Z     C
```

C = Carry-Flag - Status-Bit 0

Das Carry-Flag zeigt bei arithmetischen Operationen einen eventuell auftretenden Übertrag (neuntes Bit) an. Es kann jedoch auch vom Programmierer durch spezielle CPU-Befehle gesetzt oder gelöscht werden. Bei Verzweigungsbefehlen kann das Flag als Entscheidung verwendet werden.

Z = Zero-Flag (Null-Flag) - Status-Bit 1

Dieses Flag wird von der CPU automatisch bei Datentransfer- oder Berechnungsoperationen gesetzt, wenn das Ergebnis der Operation Null ist. Es kann auch bei Verzweigungsbefehlen als Entscheidung verwendet werden.

I = Interrupt-Disable-Flag - Status-Bit 2

Das Flag kann vom Programmierer durch CPU-Befehle gesetzt bzw. gelöscht werden. Man kann hiermit den IRQ-Eingang der CPU zeitweise unwirksam machen. Das Flag hat keinen Einfluß auf den NMI.

D = Decimal-Mode-Flag - Status-Bit 3

Wird das Flag vom Programmierer gesetzt, dann werden alle folgenden arithmetischen Operationen im BCD-Code (dezimal) durchgeführt, ansonsten erfolgen die Operationen binär.

B = Break-Command-Flag - Status-Bit 4

Dieses Flag wird von der CPU gesetzt, um anzuzeigen, daß eine Unterbrechung durch ein "Break" (Software-Interrupt) verursacht wurde.

E = Expansion-Bit - Status-Bit 5

Dieses Bit des Status-Registers ist unbenutzt.

V = Overflow-Flag - Status-Bit 6

Das Overflow-Flag wird bei binären arithmetischen Operationen beeinflußt. Es zeigt bei Verwendung des höchsten Bits (Bit 7) als Vorzeichenstelle an, ob der Rechenbereich bei der durchgeführten Operation überschritten wurde (z.B. Übertrag in die Vorzeichenstelle oder aus der Vorzeichenstelle nach Carry). Ist das Overflow-Flag nach einer Operation mit vorzeichenbe-

hafteten Zahlen gesetzt, ist das Ergebnis falsch und sollte nicht weiter verwendet werden. Hat Bit 7 keine Vorzeichenbedeutung, so ist der Wert des Overflow-Flags bedeutungslos.

N = Negativ-Flag - Status-Bit 7

Dieses Flag ermöglicht es dem Programmierer, mit einfachen Mitteln das Vorzeichen (Bit 7) des Ergebnisses zu prüfen, wenn vorzeichenbehaftete Zahlen vorliegen. Außerdem wird es gesetzt, wenn das Ergebnis der Operation im Akku größer als 127 ist. Das Negativ-Bit wird dem Bit 7 des Akku-Wertes gleichgesetzt.

Zusammenfassend kann man sagen, daß einige Flags nur durch das Benutzerprogramm, andere nur von der CPU und einige von beiden beeinflußt werden können. Jedes Flag hat dabei für den Programmierer zu einem bestimmten Zeitpunkt seine eigene Bedeutung. Durch die bedingten Sprungbefehle der CPU kann das Programm in Abhängigkeit der Wertigkeit der Status-Flags verschiedene Wege gehen.

A.2.6 ALU - Arithmetic Logic Unit

Alle logischen und arithmetischen Operationen finden in der ALU statt. Dazu gehört auch das Inkrementieren und Dekrementieren der internen Register. Die Ergebnisse der Operationen werden im Akku, in den Registern oder in den externen Speicherstellen abgelegt.

A.2.7 Interrupt-Logik

RES = Reset

Das Signal RESET ist eine grundlegende Steuerfunktion der CPU bzw. des gesamten Mikrocomputers. Es werden dabei alle wichtigen Register in einen Grundzustand versetzt, da nach dem Einschalten die Speicher und Peripherie-Register einen zufälligen Wert enthalten. Am wichtigsten ist der Grundzustand des Programmzählers ($FFFC), da der Prozessor von dieser Adresse die Startadresse für die Abarbeitung des Programmes holt (siehe Kapitel 2.9).

IRQ / NMI

Die CPU besitzt zwei Interrupt-Leitungen, IRQ und NMI. Diese Eingänge dienen dazu, von außen kommende Unterbrechungs-Anforderungen, z.B. Tastatureingaben, Meßwerte etc. einzulesen, ohne das vom Hauptprogramm

ständig abgefragt werden muß, ob Eingaben vorliegen. Über die Interrupt-Leitungen können die Eingabe-Bausteine eine Bedienung durch die CPU bei Bedarf anfordern. In einer speziellen Interrupt-Routine werden die aktuellen Register gerettet und auf den Stapel gelegt, die Anforderung bearbeitet und nach dem Zurückholen der Register das bisherige Programm fortgesetzt.

A.2.8 Timer

Der Timer sorgt in Verbindung mit dem extern angeschalteten, den Takt bestimmenden Quarz, sowohl prozessorintern für die benötigten Takte, als auch extern für die angeschlossenen Peripherie-Bausteine.

A.2.9 Speicheradressen mit besonderer Bedeutung

Der Prozessor kann insgesamt 64 KByte Speicher adressieren, der vom Hersteller des Computer-Systems beliebig für RAM, ROM oder Peripherie-Bausteine belegt werden kann.

Es gibt allerdings einige Speicherbereiche, die vom Prozessor für bestimmte Operationen unbedingt benötigt werden, und die deshalb auch entsprechend im Adressbereich belegt sein müssen.

Zero-Page

Der Adressbereich von hexadezimal 0000 bis 00FF bildet die sogenannte Zero-Page, eine "Speicherseite", deren oberes Adressbyte gleich null ist. In diesem Bereich können Befehle mit verkürzter Adressierung verwendet werden, die in der folgenden Befehlsübersicht besonders kenntlich gemacht wurden. Dieser Adressbereich muß unbedingt mit RAM belegt sein.

Stack (Prozessor-Stapelbereich)

Auch der Adressbereich von hexadizimal 0100 bis 01FF muß mit RAM belegt sein, da hier der Prozessor seinen Stack-Bereich verwaltet; einen Bereich, in dem z.B. die Rücksprungadressen bei Unterprogramm-Aufrufen abgelegt werden.

IRQ, NMI und RESET-Vektoren

Im oberen Adressbereich von hexadezimal FFFA bis FFFF liegen die Standardvektoren für NMI-Interrupt, Prozessor-Reset und den normalen Interrupt. Beim Auftreten z.B. eines Resets holt sich der Prozessor seine Start-

adresse aus den Adressen FFFC und FFFD und beginnt dann mit der Abarbeitung des Programm an der dort gefundenen Adresse.

Im einzelnen liegen hier folgende Vektoren:

Adresse	Vektor	
FFFA/FFFB	NMI	= Nicht maskierbarer Interrupt
FFFC/FFFD	Reset	= Prozessorstart
FFFE/FFFF	IRQ	= Normaler Interrupt

A.2.10 Besonderheiten des Prozessors 8502

Der im C-128 verwendete Mikroprozessor 8502 hat zusätzlich sieben Anschlüsse, die als Port mit sieben Bit dienen. Diese werden zur Steuerung verwendet. Die Festlegung, ob die einzelnen Port-Leitungen als Eingang oder Ausgang fungieren, wird durch das Datenrichtungs-Register, Adresse $0000, bestimmt. Die Daten für den Port selbst stehen im Daten-Register auf der Adresse $0001.

Datenrichtungs-Register Adresse $0000		Daten-Register Adresse $0001
Wert = 0 Lesen = Eingang Wert = 1 Schreiben = Ausgang		Bedeutung
Bit 0	Wert 1 = Ausgang	1 = BASIC-ROM, 0 = RAM
Bit 1	Wert 1 = Ausgang	1 = Kernal-ROM, 0 = RAM
Bit 2	Wert 1 = Ausgang	1 = I/O-Baust., 0 = Zeichengen.
Bit 3	Wert 1 = Ausgang	Kassette schreiben
Bit 4	Wert 0 = Eingang	Eingang Kassettenschalter
Bit 5	Wert 1 = Ausgang	Kassettenmotor Ein/Aus
Bit 6	Wert 0 = Eingang	Eingang für Taste DIN/Caps lock

A.3 Adressierungsarten

Eine besondere Stärke des Prozessors 8502 und 65xx ist die Vielzahl der verschiedenen Adressierungsarten. Die Art der Adressierung ist bei gleicher Befehlsart im Opcode enthalten.

A.3.1 Immediate - direkte Adressierung

Die Adressierungsart "Immediate" (= unmittelbare, direkte Adressierung) wird im Assemblerlisting durch ein #-Zeichen, gefolgt vom Operanden (Konstante oder Label), dargestellt. Nach der Befehlsausführung ist das gewählte Register, im Beispiel der Akku, mit dem Wert des Operanden geladen.

Bei der direkten Adressierung handelt es sich generell um Zwei-Byte-Befehle, da der Operand nicht größer als $FF, entsprechend 255, sein kann. Hier einige Beispiele:

Mnemonic	Opcode
LDA #240	$A9 $F0
LDA #$A1	$A9 $A1
LDA #'a'	$A9 $41

A.3.2 Zero-Page-Adressierung

Der Speicherbereich des Mikroprozessors 650x ist in Blöcke zu je 256 Byte, genannt Pages, eingeteilt. Dabei hat die niedrigste Page, ("Page 0", Adresse $0000 bis $00FF), im folgenden "Zero-Page" genannt, eine besondere Bedeutung.

Für die Behandlung der Zero-Page gibt es bei vielen Opcodes, z.B. LDA, besondere Befehle. Dadurch, daß im Befehl bereits die Information des High-Byte der Adresse steht (High-Byte = 00), wird für die Adressierung nur noch ein Byte (Low-Byte = 00-$FF) benötigt. Für die Darstellung des Befehls sind deshalb nur zwei Bytes notwendig (normalerweise drei), nämlich Befehl und Low-Byte der Adresse. Die Befehlsausführung wird dadurch schneller und das Programm nimmt weniger Speicherplatz in Anspruch.

Bei der Zero-Page-Adressierung wird beispielsweise bei dem Befehl LDA der Akku mit dem Inhalt der angegebenen Adresse geladen. Der Assembler erkennt am Operanden (nur ein Adress-Byte) selbstständig, daß es sich um eine Zero-Page-Adresse handelt.

A.3.3 Zero-Page indizierte Adressierung - Indexregister X

Für diese Adressierungsart gilt das unter "Zero-Page-Adressierung" ausgeführte. Es wird jedoch zu der im Operand angegebenen Adresse der Wert des Indexregisters X addiert. Im Beispiel des LDA wird also der Akku mit dem Inhalt der angegebenen, um das Indexregister X erhöhten Zero-Page-Adresse, geladen. Es ist dabei zu beachten, daß die Zero-Page durch den zu addierenden Wert des X-Registers überschritten werden kann.

A.3.4 Absolute Adressierung

Bei der absoluten Adressierung wird die Adresse, im Gegensatz zur Zero-Page-Adressierung, mit zwei Bytes dargestellt. Dadurch ist der gesamte Speicherbereich von $0000 bis $FFFF adressierbar.

Die absolut adressierten Befehle sind immer Drei-Byte-Befehle. Zu beachten ist die Reihenfolge der Adress-Bytes. Es wird immer zuerst das Low-Byte und dann das High-Byte verarbeitet.

A.3.5 Absolut indizierte Adressierung - Indexregister X und Y

Für diese Adressierungsart gilt das unter "Absolute Adressierung" ausgeführte. Es wird lediglich zu der im Operanden angegebenen Adresse der Wert des Indexregisters X bzw. Y addiert.

Im Beispiel "LDA operand,X" bzw. "LDA operand,Y" wird also der Akku mit dem Inhalt der angegebenen, um den Inhalt des Indexregisters erhöhten Adresse, geladen.

A.3.6 Indiziert-indirekte Adressierung - Indexregister X (Zero-Page)

Bei dieser Adressierungsart ist die Operandenadresse eine Speicherstelle in der Zero-Page, die eine als Operand zu verwendende Adresse enthält. Der Operand wird bei dieser Adressierungsart in der mnemonischen Schreibweise zur Unterscheidung in runde Klammern gesetzt.

In unserem Beispiel "LDA (operand,X)" wird der Wert des Indexregisters X zum Operand hinzugezählt. Die Summe ergibt einen Speicherplatz in der Zero-Page. Dieser Speicherplatz enthält das Low-Byte und der folgende Platz das High-Byte der Adresse. Der Akku wird mit dem Inhalt der Speicherstelle geladen, auf den die Adresse zeigt.

Ein Vorteil dieser indizierten Adressierung ist, daß eine 16-Bit-Adresse (Zwei-Byte-Adresse) durch nur zwei Befehls-Bytes bearbeitet werden kann, wobei ein Byte den Opcode, und das zweite Byte den indizierten Zeiger enthält. Zur Erklärung ein Beispiel aus der Praxis:

LDA (zero,X)

Dem Label "zero" wurde der Wert $12 zugeordnet, das Indexregister X enthält den Wert $04, die Speicherstellen $16 und $17 enthalten die Werte $01 und $25 und die Speicherstelle der Adresse $2501 enthält den Wert $56. Nach Ausführung des Befehls steht im Akku der Wert $56.

zero + X = $12 + $04 = $16

Die Speicherzellen $16 (Low-Byte) und $17 (High-Byte) enthalten die Adresse $2501. In der Speicherzelle $2501 steht $56. Also wird der Akku mit $56 geladen.

A.3.7 Indirekt-indizierte Adressierung - Indexregister Y (Zero-Page)

Eine weitere Art der Adressierung benutzt das Indexregister Y, wie im Beispiel "LDA (operand),Y".

Dem Label "page" wurde der Wert $38 zugewiesen, das Indexregister Y erhält den Wert $07, die Speicherstellen auf der Adresse $38 und $39 erhalten die Werte $00 und $75, also die Adresse $7500. Hierzu wird der Wert $07 des Y-Registers addiert. Die Speicherstelle der Adresse $7507 enthält den Wert $AF.

Nach Ausführung des Befehls steht im Akku der Wert $AF.

```
Adresse $7500 + $07 (Wert Y-Reg.) = $7507
Der Inhalt von $7507 = $AF
Der Akku wird mit $AF geladen.
```

A.3.8 Indirekt absolute Adressierung

Die bisher beschriebenen indirekten Adressierungen waren immer indiziert und hatten einen Bezug zur Zero-Page. Beim Mikroprocessor 650x gibt es jedoch eine Ausnahme: den indirekten Sprung. Beispiel:

JMP (lohi)

Bei der Abarbeitung dieses Befehls erfolgt der Sprung zu der Adresse, die in den Speicherstellen "lohi" (Adresse Low-Byte) und "lohi+1" (Adresse High-Byte) steht.

Ein indirekter Sprung über einen Zwei-Byte-Zeiger, der in zwei verschiedenen Pages liegt, kann nicht korrekt ausgeführt werden. An dieser Stelle hat sich ein Fehler in das Mikroprogramm des Prozessors selbst eingeschlichen. Aus Gründen der Kompatibilität zum 6510 des C-64 wurde dieser Fehler mit in den C-128 übernommen.

A.3.9 Relative Adressierung

Bei der relativen Adressierung stehen dem Benutzer acht bedingte Sprungbefehle, sogenannte Branch-Befehle, zur Verfügung. Mit dieser Adressierungsart ist es möglich, in Abhängigkeit von Flags relativ zur momentanen Adresse zu verzweigen. Da es sich nicht um absolute Sprünge handelt, sind Programme, in denen ausschließlich Branch-Befehle verwendet werden, voll verschiebbar, sofern keine Datenspeicher innerhalb des Programms liegen.

Die relative Adressierung ist nur bei den Branch-Befehlen möglich. Dafür ein Beispiel:

BEQ start Branch on Equal start

Ist das Ergebnis nach einem Vergleich gleich null, dann verzweige zu "start". In jedem anderen Fall wird das Programm fortgeführt.

Das zweite Byte eines Branch-Befehls ist eine mit Vorzeichen versehene Zahl, die zur Ermittlung der Verzweigungsadresse als positiver oder negativer Offset zum PC (Programmzähler) addiert wird. Die Adresse für die Verzweigung kann nur 1 bis 127 ($00 bis $7f) Bytes vorwärts bzw. 1 bis 128 Bytes ($80 bis $ff) zurück von der Adresse nach dem Branch-Befehl liegen.

A.3.10 Implizite Adressierung - Akkumulator Adressierung

Die impliziten Befehle sind Ein-Byte-Befehle, die keinen Operanden benötigen, weil alle Informationen bereits im Befehl enthalten (impliziert) sind. Dazu folgendes Beispiel:

TAX Transfer Akku zum X-Register

Eine Zwitterposition nehmen die Befehle ASL, LSR, ROL, ROR ein, weil sowohl der Akku (impliziert), als auch Speicherplätze als Operand verwendet werden können. Soll der Akku verwendet werden (Ein-Byte Befehl), so wird hinter den Befehl, durch ein Space getrennt, ein "A" gesetzt, z.B. "ASL A".

B Prozessor-Befehle

Im folgenden finden Sie eine Übersicht aller Prozessor-Befehle. Vorab einige Erklärungen der verwendeten Abkürzungen:

Abkürzung	Bedeutung
lo	Low-Byte - niederwertiges Byte einer Adresse
hi	High-Byte - höherwertiges Byte einer Adresse
lohi	Zwei-Byte-Adresse, 1. Byte = Low-Byte, 2. Byte = High-Byte
aa	Zero-Page-Adresse (1 Byte-Adresse)
nn	hexadezimaler Wert (1 Byte)
A	Akkumulator
X	X-Register
Y	Y-Register
SP	Stapel-Zeiger
M	Memory = RAM-Adresse (1 Byte oder 2 Byte)
#	Kennzeichnung der direkten Adressierung (Immediate)
$	Kennzeichnung einer hexadezimalen Zahl
(*1)	wird bei diesem Befehl die Pagegrenze überschritten, so wird zu der angegebenen Anzahl von Taktzyklen ein Zyklus hinzuaddiert
(*2)	Wie bei (*1), zusätzlich muß noch ein Taktzyklus addiert werden, wenn der bedingte Sprung durchgeführt wird; die Bedingung also zutrifft.

ADC - Add Memory to Accumulator with Carry

Addiere Direktwert oder Inhalt einer Adresse und Carry-Bit zum Akku Ergebnis steht im Akku und Carry

`A + M + C = A, C` beeinflußte Flags: `NV----ZC`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
ADC #nn	$69	2	2	unmittelbar
ADC aa	$65	2	3	Zero-Page
ADC aa,X	$75	2	4	Zero-Page,X
ADC lohi	$6D	3	4	absolut
ADC lohi,X	$7D	3	4 (*1)	absolut,X
ADC lohi,Y	$79	3	4 (*1)	absolut,Y
ADC (aa,X)	$61	2	6	(indirekt,X)
ADC (aa),Y	$71	2	5 (*1)	(indirekt),Y

AND - And Memory with Accumulator

Logisches UND von Direktwert oder Inhalt einer Adresse mit Akku, Ergebnis steht im Akku

`A UND M = A` beeinflußte Flags: `N-----Z-`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
AND #nn	$29	2	2	unmittelbar
AND aa	$25	2	3	Zero-Page
AND aa,X	$35	2	4	Zero-Page,X
AND lohi	$2D	3	4	absolut
AND lohi,X	$3D	3	4 (*1)	absolut,X
AND lohi,Y	$39	3	4 (*1)	absolut,Y
AND (aa,X)	$21	2	6	(indirekt,X)
AND (aa),Y	$31	2	5	(indirekt),Y

ASL - Shift Left One Bit (Memory or Accumulator)

Inhalt von Akku oder Inhalt einer Adresse um ein Bit nach links schieben, Bit 7 nach Carry, Null nach Bit 0

```
C <- 76543210 <- 0                    beeinflußte Flags: N-----ZC
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
ASL A	$0A	1	2	Akkumulator
ASL aa	$06	2	5	Zero-Page
ASL aa,X	$16	2	6	Zero-Page,X
ASL lohi	$0E	3	6	absolut
ASL lohi,X	$1E	3	7	absolut,X

BCC - Branch on Carry Clear

Verzweigung, wenn Carry-Flag nicht gesetzt

```
Verzweigung, wenn C = 0               beeinflußte Flags: --------
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BCC nn	$90	2	2 (*2)	Relativ

BCS - Branch on Carry Set

Verzweigung, wenn Carry-Flag gesetzt

```
Verzweigung, wenn C = 1               beeinflußte Flags: --------
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BCS nn	$B0	2	2 (*2)	Relativ

BEQ - Branch on Equal

Verzweigung, wenn Zero-Flag gesetzt (Ergebnis gleich 0)

Verzweigung, wenn Z = 1 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BEQ nn	$F0	2	2 (*2)	Relativ

BIT - Test Bits in Memory with Accumulator

Bit 6 und 7 werden ins Status-Register übertragen. Falls das Ergebnis von A UND M Null ist, dann Z = 1; ansonsten Z = 0

A UND M, M7 -> N, M6 -> V beeinflußte Flags: NV----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BIT aa	$24	2	3	Zero-Page
BIT lohi	$2C	3	4	absolut

BMI - Branch on Minus

Verzweigung, wenn Negativ-Flag gesetzt

Verzweigung, wenn N = 1 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BMI nn	$30	2	2 (*2)	Relativ

BNE - Branch on Not Equal

Verzweigung, wenn Zero-Flag nicht gesetzt

Verzweigung, wenn Z = 0 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BNE nn	$D0	2	2 (*2)	Relativ

BPL - Branch on Plus

Verzweigung, wenn Negativ-Flag nicht gesetzt

Verzweigung, wenn N = 0 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BPL nn	$10	2	2 (*2)	Relativ

BRK - Force Break

Programmunterbrechung (Software-Interrupt), I = 1

PC+2,Flags -> Stapel, SP=SP-3 beeinflußte Flags: ---B-I--

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BRK	$00	1	7	implizit

BVC - Branch on Overflow Clear

Verzweigung, wenn Overflow-Flag nicht gesetzt

Verzweigung, wenn V = 0 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BVC nn	$50	2	2 (*2)	Relativ

BVS - Branch on Overflow Set

Verzweigung, wenn Overflow-Flag gesetzt

Verzweigung, wenn V = 1 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
BVS nn	$70	2	2 (*2)	Relativ

CLC - Clear Carry Flag

Carry-Flag löschen

C = 0 beeinflußte Flags: -------C

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CLC	$18	1	2	implizit

CLD - Clear Decimal Mode

Dezimal-Modus abschalten und Binär-Modus einschalten

D = 0 beeinflußte Flags: ----D---

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CLD	$D8	1	2	implizit

CLI - Clear Interrupt Disable Bit

Interrupt-Flag löschen

I = 0 beeinflußte Flags: -----I--

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CLI	$58	1	2	implizit

CLV - Clear Overflow Flag

Overflow-Flag löschen

V = 0 beeinflußte Flags: -V------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CLV	$B8	1	2	implizit

CMP - Compare Memory and Accumulator

Akkumulator wird mit Direktwert oder Inhalt einer Adresse verglichen, Ergebnis durch Flags auswertbar

A - M beeinflußte Flags: N-----ZC

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CMP #nn	$C9	2	2	unmittelbar
CMP aa	$C5	2	3	Zero-Page
CMP aa,X	$D5	2	4	Zero-Page,X
CMP lohi	$CD	3	4	absolut
CMP lohi,X	$DD	3	4 (*1)	absolut,X
CMP lohi,Y	$D9	3	4 (*1)	absolut,Y
CMP (aa,X)	$C1	2	6	(indirekt,X)
CMP (aa),Y	$D1	2	5 (*1)	(indirekt),Y

CPX - Compare Memory and Index X

Indexregister X wird mit Direktwert oder Inhalt einer Adresse verglichen, Ergebnis durch Flags auswertbar

X - M beeinflußte Flags: N-----ZC

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CPX #nn	$E0	2	2	unmittelbar
CPX aa	$E4	2	3	Zero-Page
CPX lohi	$EC	3	4	absolut

CPY - Compare Memory and Index Y

Indexregister Y wird mit Direktwert oder Inhalt einer Adresse verglichen, Ergebnis durch Flags auswertbar

Y - M beeinflußte Flags: N-----ZC

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
CPY #nn	$C0	2	2	unmittelbar
CPY aa	$C4	2	3	Zero-Page
CPY lohi	$CC	3	4	absolut

DEC - Decrement Memory by One

Der Inhalt der Adresse wird um 1 vermindert

M = M - 1 beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
DEC aa	$C6	2	5	Zero-Page
DEC aa,X	$D6	2	6	Zero-Page,X
DEC lohi	$CE	3	6	absolut
DEC lohi,X	$DE	3	7	absolut,X

DEX - Decrement Index X by One

Indexregister X wird um 1 vermindert

X = X - 1 beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
DEX	$CA	1	2	implizit

DEY - Decrement Index Y by One

Indexregister Y wird um 1 vermindert

`Y = Y - 1` beeinflußte Flags: `N-----Z-`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
DEY	$88	1	2	implizit

EOR - Exclusive-OR Memory with Accumulator

Exklusiv-ODER Verknüpfung von Wert oder Inhalt einer Adresse mit Akku, Ergebnis steht im Akku

`A Ex-ODER M = A` beeinflußte Flags: `N-----Z-`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
EOR #nn	$49	2	2	unmittelbar
EOR aa	$45	2	3	Zero-Page
EOR aa,X	$55	2	4	Zero-Page,X
EOR lohi	$4D	3	4	absolut
EOR lohi,X	$5D	3	4 (*1)	absolut,X
EOR lohi,Y	$59	3	4 (*1)	absolut,Y
EOR (aa,X)	$41	2	6	(indirekt,X)
EOR (aa),Y	$51	2	5 (*1)	(indirekt),Y

INC - Increment Memory by One

Inhalt einer Adresse wird um 1 erhöht

`M = M + 1` beeinflußte Flags: `N-----Z-`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
INC aa	$E6	2	5	Zero-Page
INC aa,X	$F6	2	6	Zero-Page,X
INC lohi	$EE	3	6	absolut
INC lohi,X	$FE	3	7	absolut,X

INX - Increment Index X by One

Indexregister X wird um 1 erhöht

`X = X + 1` beeinflußte Flags: `N-----Z-`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
INX	$E8	1	2	implizit

INY - Increment Index Y by One

Indexregister Y wird um 1 erhöht

`Y = Y + 1` beeinflußte Flags: `N-----Z-`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
INY	$C8	1	2	implizit

JMP - Jump to New Location

Sprung zu einer neuen Adresse

`PC = lohi` beeinflußte Flags: `--------`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
JMP lohi	$4C	3	3	absolut
JMP (lohi)	$6C	3	5	indirekt

JSR - Jump to New Location Saving Return Adress

Sprung zu einer neuen Adresse; Rücksprungadresse wird auf den Stapel gelegt

PC+2 lohi -> Stapel, SP=SP-2 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
JSR lohi	$20	3	6	absolut

LDA - Load Accumulator with Memory

Lade Akkumulator mit Direktwert oder Inhalt einer Adresse

M -> A beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
LDA #nn	$A9	2	2	unmittelbar
LDA aa	$A5	2	3	Zero-Page
LDA aa,X	$B5	2	4	Zero-Page,X
LDA lohi	$AD	3	4	absolut
LDA lohi,X	$BD	3	4 (*1)	absolut,X
LDA lohi,Y	$B9	3	4 (*1)	absolut,Y
LDA (aa,X)	$A1	2	6	(indirekt,X)
LDA (aa),Y	$B1	2	5 (*1)	(indirekt),Y

LDX - Load Index X with Memory

Lade Indexregister X mit Direktwert oder Inhalt einer Adresse

M -> X beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
LDX #nn	$A2	2	2	unmittelbar
LDX aa	$A6	2	3	Zero-Page
LDX aa,Y	$B6	2	4	Zero-Page,Y
LDX lohi	$AE	3	4	absolut
LDX lohi,Y	$BE	3	4 (*1)	absolut,Y

LDY - Load Index Y with Memory

Lade Indexregister Y mit Direktwert oder Inhalt einer Adresse

M -> Y beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
LDY #nn	$A0	2	2	unmittelbar
LDY aa	$A4	2	3	Zero-Page
LDY aa,X	$B4	2	4	Zero-Page,X
LDY lohi	$AC	3	4	absolut
LDY lohi,X	$BC	3	4 (*1)	absolut,X

LSR - Shift Right One Bit (Memory or Accumulator)

Inhalt einer Adresse oder Akku nach rechts schieben Wert 0 nach Bit 7, Bit 0 nach Carry

0 -> 76543210 -> C beeinflußte Flags: N-----ZC

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
LSR A	$4A	1	2	Akkumulator
LSR aa	$46	2	5	Zero-Page
LSR aa,X	$56	2	6	Zero-Page,X
LSR lohi	$4E	3	6	absolut
LSR lohi,X	$5E	3	7	absolut,X

NOP - No Operation

Keine Prozessoraktivität für 2 Taktzyklen

beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
NOP	$EA	1	2	implizit

ORA - OR Memory with Accumulator

Logisches ODER von Direktwert oder Inhalt einer Adresse mit Akku

A ODER M -> A beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
ORA #nn	$09	2	2	unmittelbar
ORA aa	$05	2	3	Zero-Page
ORA aa,X	$15	2	4	Zero-Page,X
ORA lohi	$0D	3	4	absolut
ORA lohi,X	$1D	3	4 (*1)	absolut,X
ORA lohi,Y	$19	3	4 (*1)	absolut,Y
ORA (aa,X)	$01	2	6	(indirekt,X)
ORA (aa),Y	$11	2	5	(indirekt),Y

PHA - Push Accumulator on Stack

Inhalt des Akku auf dem Stapel ablegen

A -> Stapel,SP=SP-1 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
PHA	$48	1	3	implizit

PHP - Push Processor Status on Stack

Inhalt des Status-Registers auf dem Stapel ablegen

P -> Stapel, SP=SP-1 beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
PHP	$08	1	3	implizit

PLA - Pull Accumulator from Stack

Byte vom Stapel in den Akku bringen

Stapel -> A, SP=SP+1 beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
PLA	$68	1	4	implizit

PLP - Pull Processor Status from Stack

Byte vom Stapel in das Status-Register bringen

Stapel -> SR, SP=SP+1 beeinflußte Flags: NVEBDIZC

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
PLP	$28	1	4	implizit

ROL - Rotate Left One Bit (Memory or Accumulator)

Linksrotation um 1 Bit (Inhalt einer Adresse oder Akkumulator)

<- C <- 76543210 <- C <- beeinflußte Flags: N-----ZC

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
ROL A	$2A	1	2	Akkumulator
ROL aa	$26	2	5	Zero-Page
ROL aa,X	$36	2	6	Zero-Page,X
ROL lohi	$2E	3	6	absolut
ROL lohi,X	$3E	3	7	absolut,X

ROR - Rotate Right One Bit (Memory or Accu)

Rechtsrotation um 1 Bit (Inhalt einer Adresse oder Akkumulator)

```
-> C -> 76543210 -> C ->          beeinflußte Flags: N-----ZC
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
ROR A	$6A	1	2	Akkumulator
ROR aa	$66	2	5	Zero-Page
ROR aa,X	$76	2	6	Zero-Page,X
ROR lohi	$6E	3	6	absolut
ROR lohi,X	$7E	3	7	absolut,X

RTI - Return from Interrupt

Return vom Interrupt

```
Stapel -> PC,SR  ,SP=SP+3         beeinflußte Flags: NVEBDIZC
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
RTI	$40	1	6	implizit

RTS - Return from Subroutine

Return vom Unterprogramm

```
PC vom Stapel zurück              beeinflußte Flags: --------
PC = PC + 1, SP=SP+2
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
RTS	$60	1	6	implizit

SBC - Subtract Memory from Accu with Borrow

Subtrahiert Direktwert oder Inhalt einer Adresse vom Akku mit Übertrag

`A - M - C -> A` beeinflußte Flags: `NV----ZC`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
SBC #nn	$E9	2	2	unmittelbar
SBC aa	$E5	2	3	Zero-Page
SBC aa,X	$F5	2	4	Zero-Page,X
SBC lohi	$ED	3	4	absolut
SBC lohi,X	$FD	3	4 (*1)	absolut,X
SBC lohi,Y	$F9	3	4 (*1)	absolut,Y
SBC (aa,X)	$E1	2	6	(indirekt,X)
SBC (aa),Y	$F1	2	5 (*1)	(indirekt),Y

SEC - Set Carry Flag

Setzt Carry-Flag

`C = 1` beeinflußte Flags: `-------C`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
SEC	$38	1	2	implizit

SED - Set Decimal Modus

Setzt Dezimal-Flag (Dezimal-Modus Ein)

`D = 1` beeinflußte Flags: `----D---`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
SED	$F8	1	2	implizit

SEI - Set Interrupt Disable Status

Setzt Interrupt-Flag (Interrupt-Maskierung setzen)

`I = 1` beeinflußte Flags: `-----I--`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
SEI	$78	1	2	implizit

STA - Store Accumulator in Memory

Akkumulatorinhalt im Speicher ablegen

`A -> M` beeinflußte Flags: `--------`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
STA aa	$85	2	3	Zero-Page
STA aa,X	$95	2	4	Zero-Page,X
STA lohi	$8D	3	4	absolut
STA lohi,X	$9D	3	5	absolut,X
STA lohi,Y	$99	3	5	absolut,Y
STA (aa,X)	$81	2	6	(indirekt,X)
STA (aa),Y	$91	2	6	(indirekt),Y

STX - Store Index X in Memory

Inhalt von Indexregister X im Speicher ablegen

`X -> M` beeinflußte Flags: `--------`

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
STX aa	$86	2	3	Zero-Page
STX aa,Y	$96	2	4	Zero-Page,Y
STX lohi	$8E	3	4	absolut

STY - Store Index Y in Memory

Inhalt von Indexregister Y im Speicher ablegen

```
Y -> M                                  beeinflußte Flags: --------
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
STY aa	$84	2	3	Zero-Page
STY aa,X	$94	2	4	Zero-Page,X
STY lohi	$8C	3	4	absolut

TAX - Transfer Accumulator to Index X

Kopiere Akkumulatorinhalt nach Indexregister X

```
A -> X                                  beeinflußte Flags: N-----Z-
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
TAX	$AA	1	2	implizit

TAY - Transfer Accumulator to Index Y

Kopiere Akkumulatorinhalt nach Indexregister Y

```
A -> Y                                  beeinflußte Flags: N-----Z-
```

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
TAY	$A8	1	2	implizit

TSX - Transfer Stack Pointer to Index X

Kopiere Stapelzeiger nach Indexregister X

SP -> X beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
TSX	$BA	1	2	implizit

TXA - Transfer Index X to Accumulator

Kopiere Inhalt von Indexregister X in Akkumulator

X -> A beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
TXA	$8A	1	2	implizit

TXS - Transfer Index X to Stack Pointer

Kopiere Inhalt von Indexregister X nach Stapelzeiger

X -> SP beeinflußte Flags: --------

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
TXS	$9A	1	2	implizit

TYA - Transfer Index Y to Accumulator

Kopiere Inhalt von Indexregister Y in Akkumulator

Y -> A beeinflußte Flags: N-----Z-

Mnemonic	Opcode	Byteanzahl	Taktzyklen	Adressierung
TYA	$98	1	2	implizit

Zusätzliche Prozessor-Befehle

Die CPU 8502 des C-128 ist eine Weiterentwicklung des Mikroprozessors 6502. Dieser Prozessor war von Anfang an in der Lage, mehr als die im Datenblatt genannten Befehle auszuführen. Diese Befehle sollten eigentlich in den neueren Versionen des Prozessors eliminiert werden, Tests haben aber ergeben, daß sie unverändert vorhanden und damit auch noch auf dem 8502 ausführbar sind.

Sie finden im folgenden eine Aufstellung der zusätzlichen Befehle mit ihren empirisch ermittelten Funktionen. Es sei hier allerdings noch einmal darauf hingewiesen, daß die Funktion dieser Befehle vom Hersteller - also MOS bzw. Commodore - nicht garantiert wird. Daher müssen diese Befehle nicht unbedingt auf jedem 8502 laufen. Im einfachsten Fall probieren Sie es einfach bei Ihrem Rechner einmal aus.

Diese Befehle sind natürlich hervorragend geeignet, eigene Maschinenprogramme vor dem Interpretieren zu schützen, wenn Sie anstelle gleichwertiger oder ähnlicher Normalbefehle verwendet werden. Herkömmliche Dis-Assembler und Monitore - so auch der Monitor im C-128 - sind nicht in der Lage, diese Befehle zu interpretieren und somit erscheint das Programm als mehr oder weniger chaotische Folge von "???"-Befehlen. Ein Alptraum für jeden, der dieses Programm verstehen will ...

Folgende Bezeichnungen haben sich in letzter Zeit eingebürgert (siehe auch 64er-Sonderheft 8/1985):

A11 - And Register with $11

Das X- bzw. Y-Register wird mit der Speicherzelle $11 UND-verknüpft und das Ergebnis durch X bzw. Y indiziert abgespeichert.

Mnemonic	Opcode	Byteanzahl
A11 lohi,X	$9C	3
A11 lohi,Y	$9E	3

AAX - And Accu with X and Store Accu

Die Wirkung ist klar.

Mnemonic	Opcode	Byteanzahl
AAX #nn	$8B	2
AAX aa	$87	2
AAX aa,Y	$97	2
AAX lohi	$8F	3
AAX (aa,X)	$83	2

ASR - And with Accu and Shift Right

Entspricht der Befehlsfolge AND/LSR

Mnemonic	Opcode	Byteanzahl
ASR #nn	$6B	2

ARR - And with Accu and Rotate Right

Entspricht der Befehlsfolge AND/ROR

Mnemonic	Opcode	Byteanzahl
ASR #nn	$4B	2

AXS - And Accu with X and Subtract from Data

Der Wert wird vom Ergebnis der UND-Vernüpfung zwischen den Registern A und X subtrahiert und in das X-Register geschoben.

Mnemonic	Opcode	Byteanzahl
ASR #nn	$CB	2

DCP - Decrement and Compare with A

Entspricht der Folge DEC/CMP

Mnemonic	Opcode	Byteanzahl
DCP aa	$C7	2
DCP aa,X	$D7	2
DCP lohi	$CF	3
DCP lohi,X	$DF	3
DCP lohi,Y	$DB	3
DCP (aa,X)	$C3	2
DCP (aa),Y	$D3	2

DOP - Double NOP

Die folgenden Codes wirken wie der Befehl NOP, sind aber zwei Byte lang. Das zweite Byte wird übersprungen.

$04, $14, $34, $44, $54, $64, $74, $D4, $F4, $80, $89, $93

ISC - Increment and Subtract with Carry

Entspricht der Folge INC/SBC

Mnemonic	Opcode	Byteanzahl
ISC aa	$E7	2
ISC aa,X	$F7	2
ISC lohi	$EF	3
ISC lohi,X	$FF	3
ISC lohi,Y	$FB	3
ISC (aa,X)	$E3	2
ISC (aa),Y	$F3	2

KIL - Killercodes

Die folgenden Codes führen zu einem Totalabsturz des Prozessors, aus dem nur noch durch ein RESET zu entkommen ist:

$02, $12, $22, $32, $42, $52, $62, $72, $92, $B2, $D2, $F2

ARR - And with Accu and Rotate Right

Entspricht der Befehlsfolge LDA/AND/TAX/TSX

Mnemonic	Opcode	Byteanzahl
LAR lohi,Y	$BB	3

LAX - Load A and X

Entspricht der Folge LDA/TAX

Mnemonic	Opcode	Byteanzahl
LAX aa	$A7	2
LAX aa,X	$B7	2
LAX lohi	$AF	3
LAX lohi,Y	$BB	3
LAX (aa,X)	$A3	2
LAX (aa),Y	$B3	2

NOP - No Operation

Die folgenden Codes haben die gleiche Wirkung wie das normale NOP:

$1A, $3A, $5A, $7A, $DA, $FA

RLA - Rotate Left, And with A, Store A

Entspricht der Folge ROL/AND/STA

Mnemonic	Opcode	Byteanzahl
RLA aa	$27	2
RLA aa,X	$37	2
RLA lohi	$2F	3
RLA lohi,X	$3F	3
RLA lohi,Y	$3B	3
RLA (aa,X)	$23	2
RLA (aa),Y	$33	2

RRA - Rotate Right and Add with Carry

Entspricht der Folge ROR/ADC

Mnemonic	Opcode	Byteanzahl
RRA aa	$67	2
RRA aa,X	$77	2
RRA lohi	$6F	3
RRA lohi,X	$7F	3
RRA lohi,Y	$7B	3
RRA (aa,X)	$63	2
RRA (aa),Y	$73	2

SLO - Shift Left and Or with A

Entspricht der Folge ASL/ORA

Mnemonic	Opcode	Byteanzahl
SLO aa	$07	2
SLO aa,X	$17	2
SLO lohi	$0F	3
SLO lohi,X	$1F	3
SLO lohi,Y	$1B	3
SLO (aa,X)	$13	2
SLO (aa),Y	$03	2

SRE - Shift Right and Eor with A

Entspricht der Folge LSR/EOR

Mnemonic	Opcode	Byteanzahl
SRE aa	$47	2
SRE aa,X	$57	2
SRE lohi	$4F	3
SRE lohi,X	$5F	3
SRE lohi,Y	$5B	3
SRE (aa,X)	$43	2
SRE (aa),Y	$53	2

TOP - Triple NOP

Die folgenden Codes haben die gleiche Wirkung wie NOP. Die zwei zusätzlichen Bytes werden überlesen.

$0C, $1C, $3C, $5C, $7C, $DC, $FC

C Umrechnungstabelle

Dezimal	Binär	Oktal	Hexadezimal
0	00000000	000	00
1	00000001	001	01
2	00000010	002	02
3	00000011	003	03
4	00000100	004	04
5	00000101	005	05
6	00000110	006	06
7	00000111	007	07
8	00001000	010	08
9	00001001	011	09
10	00001010	012	0A
11	00001011	013	0B
12	00001100	014	0C
13	00001101	015	0D
14	00001110	016	0E
15	00001111	017	0F
16	00010000	020	10
17	00010001	021	11
18	00010010	022	12
19	00010011	023	13
20	00010100	024	14
21	00010101	025	15
22	00010110	026	16
23	00010111	027	17
24	00011000	030	18
25	00011001	031	19
26	00011010	032	1A
27	00011011	033	1B
28	00011100	034	1C
29	00011101	035	1D
30	00011110	036	1E
31	00011111	037	1F
32	00100000	040	20
33	00100001	041	21
34	00100010	042	22
35	00100011	043	23
36	00100100	044	24
37	00100101	045	25
38	00100110	046	26
39	00100111	047	27
40	00101000	050	28
41	00101001	051	29
42	00101010	052	2A
43	00101011	053	2B
44	00101100	054	2C
45	00101101	055	2D
46	00101110	056	2E
47	00101111	057	2F
48	00110000	060	30
49	00110001	061	31

Dezimal	Binär	Oktal	Hexadezimal
50	00110010	062	32
51	00110011	063	33
52	00110100	064	34
53	00110101	065	35
54	00110110	066	36
55	00110111	067	37
56	00111000	070	38
57	00111001	071	39
58	00111010	072	3A
59	00111011	073	3B
60	00111100	074	3C
61	00111101	075	3D
62	00111110	076	3E
63	00111111	077	3F
64	01000000	100	40
65	01000001	101	41
66	01000010	102	42
67	01000011	103	43
68	01000100	104	44
69	01000101	105	45
70	01000110	106	46
71	01000111	107	47
72	01001000	110	48
73	01001001	111	49
74	01001010	112	4A
75	01001011	113	4B
76	01001100	114	4C
77	01001101	115	4D
78	01001110	116	4E
79	01001111	117	4F
80	01010000	120	50
81	01010001	121	51
82	01010010	122	52
83	01010011	123	53
84	01010100	124	54
85	01010101	125	55
86	01010110	126	56
87	01010111	127	57
88	01011000	130	58
89	01011001	131	59
90	01011010	132	5A
91	01011011	133	5B
92	01011100	134	5C
93	01011101	135	5D
94	01011110	136	5E
95	01011111	137	5F
96	01100000	140	60
97	01100001	141	61
98	01100010	142	62
99	01100011	143	63
100	01100100	144	64
101	01100101	145	65
102	01100110	146	66
103	01100111	147	67
104	01101000	150	68
105	01101001	151	69

Dezimal	Binär	Oktal	Hexadezimal
106	01101010	152	6A
107	01101011	153	6B
108	01101100	154	6C
109	01101101	155	6D
110	01101110	156	6E
111	01101111	157	6F
112	01110000	160	70
113	01110001	161	71
114	01110010	162	72
115	01110011	163	73
116	01110100	164	74
117	01110101	165	75
118	01110110	166	76
119	01110111	167	77
120	01111000	170	78
121	01111001	171	79
122	01111010	172	7A
123	01111011	173	7B
124	01111100	174	7C
125	01111101	175	7D
126	01111110	176	7E
127	01111111	177	7F
128	10000000	200	80
129	10000001	201	81
130	10000010	202	82
131	10000011	203	83
132	10000100	204	84
133	10000101	205	85
134	10000110	206	86
135	10000111	207	87
136	10001000	210	88
137	10001001	211	89
138	10001010	212	8A
139	10001011	213	8B
140	10001100	214	8C
141	10001101	215	8D
142	10001110	216	8E
143	10001111	217	8F
144	10010000	220	90
145	10010001	221	91
146	10010010	222	92
147	10010011	223	93
148	10010100	224	94
149	10010101	225	95
150	10010110	226	96
151	10010111	227	97
152	10011000	230	98
153	10011001	231	99
154	10011010	232	9A
155	10011011	233	9B
156	10011100	234	9C
157	10011101	235	9D
158	10011110	236	9E
159	10011111	237	9F
160	10100000	240	A0
161	10100001	241	A1

Dezimal	Binär	Oktal	Hexadezimal
162	10100010	242	A2
163	10100011	243	A3
164	10100100	244	A4
165	10100101	245	A5
166	10100110	246	A6
167	10100111	247	A7
168	10101000	250	A8
169	10101001	251	A9
170	10101010	252	AA
171	10101011	253	AB
172	10101100	254	AC
173	10101101	255	AD
174	10101110	256	AE
175	10101111	257	AF
176	10110000	260	B0
177	10110001	261	B1
178	10110010	262	B2
179	10110011	263	B3
180	10110100	264	B4
181	10110101	265	B5
182	10110110	266	B6
183	10110111	267	B7
184	10111000	270	B8
185	10111001	271	B9
186	10111010	272	BA
187	10111011	273	BB
188	10111100	274	BC
189	10111101	275	BD
190	10111110	276	BE
191	10111111	277	BF
192	11000000	300	C0
193	11000001	301	C1
194	11000010	302	C2
195	11000011	303	C3
196	11000100	304	C4
197	11000101	305	C5
198	11000110	306	C6
199	11000111	307	C7
200	11001000	310	C8
201	11001001	311	C9
202	11001010	312	CA
203	11001011	313	CB
204	11001100	314	CC
205	11001101	315	CD
206	11001110	316	CE
207	11001111	317	CF
208	11010000	320	D0
209	11010001	321	D1
210	11010010	322	D2
211	11010011	323	D3
212	11010100	324	D4
213	11010101	325	D5
214	11010110	326	D6
215	11010111	327	D7
216	11011000	330	D8
217	11011001	331	D9

Dezimal	Binär	Oktal	Hexadezimal
218	11011010	332	DA
219	11011011	333	DB
220	11011100	334	DC
221	11011101	335	DD
222	11011110	336	DE
223	11011111	337	DF
224	11100000	340	E0
225	11100001	341	E1
226	11100010	342	E2
227	11100011	343	E3
228	11100100	344	E4
229	11100101	345	E5
230	11100110	346	E6
231	11100111	347	E7
232	11101000	350	E8
233	11101001	351	E9
234	11101010	352	EA
235	11101011	353	EB
236	11101100	354	EC
237	11101101	355	ED
238	11101110	356	EE
239	11101111	357	EF
240	11110000	360	F0
241	11110001	361	F1
242	11110010	362	F2
243	11110011	363	F3
244	11110100	364	F4
245	11110101	365	F5
246	11110110	366	F6
247	11110111	367	F7
248	11111000	370	F8
249	11111001	371	F9
250	11111010	372	FA
251	11111011	373	FB
252	11111100	374	FC
253	11111101	375	FD
254	11111110	376	FE
255	11111111	377	FF

Stichwortverzeichnis

.byte, 103
.word, 103

1-MHz-Betrieb, 12
2-MHz-Betrieb, 12, 30
40-Zeichen-Modus, 11, 56
6502, 9, 405
6510, 10
65xx-Prozessor, 9
80-Zeichen-Bildschir,m 33
80-Zeichen-Modus, 56
80-Zeichen-Schirm, 64
8502, 9, 405
8502-CPU, 10

A11, 435
AAX, 435
AD-Wandler, 25
ADC, 416
Adresse, hexadezimale 101
  setzen, 82, 83
  symbolische, 101
Adressierung,direkte 410
  verkürzte, 409
Adressierungsart, 410, 412
Adressierunsart, 100
Akkumulator, 406
Alarmzeit, 41, 42
ALU, 408
AND, 416
ARR, 436, 437
ASCII, 51
ASCII-Code, 101
ASCII-Zeichen, 104
ASCII-Zeichensatz, 92
ASL, 417
ASR, 435
Assembler, 48, 99
Assembler-Direktive, 103
Assemblerlisting, 54, 99
Assemblersprache, 99
Attack, 21, 22
Attributbyte, 55
Attributspeicher, 59
Auflösung, 12, 34, 60
Ausgabe-Kanal öffnen 76
AXS, 436

Bandpaß-Filter, 24
Bank, beliebige, 31
  setzen, 69
Bank 0, 29
BASIC, 12
BASIC-Interpreter, 47
BASIC-Variablen-Zeiger 89
Basis-Adresse, 79
Baudrate, 43
BCC, 417
BCD, 41, 51
BCD-Code, 407
BCS, 417
Bedingungs-Bit, 406
Befehle, zusätzliche, 434
BEQ, 418
Berechnungsoperation, 407
Betriebssystem, 47, 99
Bildschirm, 75
  initialisieren, 69
  umschalten, 68
Bildschirm-Editor, 47, 53
Bildschirmfenster, 53, 58, 65
  definieren, 65
Bildschirmformat, 9
Bildschirmpunkt, 12
Bildschirmspeicher, 57
Bildschirmsteuerung, 11
Bildschirmzeichen, 33
bindec, 51
BIT, 418
Bitmap-Grafik, 12, 34
Blockcursor, 60
BMI, 418
BNE, 419
Boot-Programm, 68
BPL, 419
Branch-Befehl, 406, 413
Break, 407
Break-Command-Flag, 407
BRK, 48, 419
BVC, 419
BVS, 420
Byte lesen, 90
  schreiben, 91
  vergleichen, 91

C-128-Modus, 405
C-64-Modus, 11, 12, 405
  umschalten, 67
Carry-Flag, 407
Centronics-Schnittstelle 92
CIA, 38, 43
CIA-Behandlung, 92
CLC, 420
CLD, 420
CLI, 420
CLV, 421
CMP, 421
CMPARE, 32
cmpbyt, 91
Common Area, 30
Compiler, 99
Complex Interface Adapter 43, 46
Computerprogramm,
  mnemonisches, 99
  symbolisches, 99
CP/M, 9, 68
CPX, 421
CPY, 422
CRA, 39
CRB, 40
Cross-Assembler, 100
Crossreferenzliste, 101, 105
Cursorform, 60, 61
Cursormodus, 59
Cursorposition, 58, 59
  lesen, 78
  schreiben, 78
Cursortaste, 47

Datei sichern, 77
Dateinummer,logische, 68
Dateitabelle, 68
Daten-Register, 410
Datenrichtungs-Register, 410
Datenrichtungsregister, 38
Datentransfer, 88
Datentransferoperation, 407
Datenverkehr, 65
DCP, 436
DDRA, 38
DDRB, 38
DEC, 422
Decay, 21, 22
Decimal-Mode-Flag, 407
Deklaration, 102
DEX, 422
DEY, 423
DIN-Zeichensatz, 92
DMA-Anforderung, 67
DOP, 437
Dreieckschwingung, 21
Drucker, 65

Echtzeituhr, 41
  programmierbare, 41
Editor, 65, 99
  initialisieren, 55, 71
Editor-Funktion, 61
Editor-Routine, 29
Ein-/Ausgabe-Baustein, 38
Ein-/Ausgabe-Kanal
  schließen, 76
Eingabe-Kanal öffnen, 75
Eingabekanal, aktiver, 76
EOR, 423
ESC-Funktion, 61
ESCAPE-Sequenz, 61
Expansion-Bit, 407

Farbe, 34
Farbinformation, 11, 55
Farbwechsel, 12
Fehlermeldung, 72
FETCH, 31
File öffnen, 75
schließen, 67, 75
Filetabelle löschen, 78
Filter programmierbares, 15, 23
Filter-Frequenz, 23
Filterart, 24
Filterkontrollregist,er 24
Floppy, 65
schnelle, 67
Frequenz, 17
Fullscreen-Editor, 53
Funktionsblock, 405
Funktionskarte, 68
Funktionstaste, 57, 63
belegen, 69

Generator 2, 23
Gerät, beliebiges, 75
Geräteadresse, 68
Gerätenummer, 68
getbyt, 90
getind, 88
getvdb, 86
getvdc, 84
Grafik, 12, 34
Grafikausgabe, 9
Grafikbereich, 12
Grafikzeichen, 56
Großbuchstabe, 56
Grundeinstellung, 55
Grundton, 20

HELP-Taste, 57
Hex-Adresse, 101
Hexadezimal-Wert, 51
Hilfsbaustein, 11
Hochpaß-Filter, 25
Hüllkurve, 15
Hüllkurvenzyklus, 21

I/O-Baustein, ausblenden, 82
einblenden, 81
I/O-Bausteine initialisieren, 71
I/O-Bereich, 81
I/O-Vektor einrichten, 71
ICR, 42, 43
IEC-Bus, 72, 75
IEC-Timeout, 73
Immediate-Befehl, 104
INC, 423
Index-Speicherstelle lesen, 90
Indexfeld, 57
Interrupt, 39, 42, 65, 406, 409
Interrupt-Disable-Flag, 407
Interrupt-Kontrollregister, 41
Interrupt-Leitung, 408
Interrupt-Routine, 409
INX, 424
INY, 424
IRQ, 65, 408
IRQ-Eingang, 407
ISC, 437

jcurs, 59
jdsply, 55, 56
jesc, 61
jint80, 64
jkey, 58
jkyset, 63
jloop5, 57
jlp2, 57
JMP, 424
JMPFAR, 33
jpcint, 55
jplot, 58
jprint, 57, 61
jrep, 58
jrsirq, 64
jscorg, 58
JSR, 425
JSRFAR, 32
jswap, 65
JUMP, 53
jwind, 65

kacptr, 73
Kaltstartroutine, 68
Kammerton, 17
Kassette, 75
Kassettenpuffer einrichten, 71
Kassettenrekorder, 38, 65
kbasin, 76
kbotcl, 68
kc64md, 67
kchkin, 75
kciout, 73
kckout, 76
kclall, 78
kclock, 78
kclose, 75
kclrch, 76
kclsal, 67
kdlchr, 69
kdmacl, 67
Kernal, 47, 65, 99
Kernal-Routine, 29, 47
KEY-Befehl, 57
Key-Definition, 57
kgetin, 78
kicmpv, 70
kifetv, 70
KIL, 437
kiobas, 79
kioini, 71
kistav, 70
kjmpfr, 69
kjsrfr, 69
Klangänderung, 20
Klangerzeugung, 15
elektronische, 22
Klangquelle, externe, 24
Klein-/Großschreibung, 56
klistn, 74
klkupl, 68
klkups, 68
kloads, 76
kmembt, 73
kmemtp, 72
Kommentar-Feld, 104
Kommunikation, 11
Kompatibilität, 11
Kontrollregister, 39, 40
kopen, 75
kpcint, 71
kpfkey, 69
kphenx, 68
kplot, 78
kprimm, 70
kptcfg, 69
kramts, 71
krdtim, 77
kreads, 74
kresto, 71
ksaves, 77
kscnky, 73
kscror, 78
ksecnd, 72
ksetmo, 73
kspio, 67
kstbnk, 69
kstlfs, 75
kstmsg, 71
kstnam, 75
kstop, 78
ksttim, 77
kswapr, 68
ktalk, 74
ktksa, 72
kunlsn, 74
kuntlk, 74
kvecto, 71

Label, 101
linksbündiges, 102
zuweisen, 102
Label-Definition, 105
Labelname, 99, 105

Lautstärke, 22, 24
Lautstärke-Register, 26
LAX, 438
LDA, 425
LDX, 425
LDY, 426
Line-Editor, 53
LISTEN, 72
  ausgegeben, 74
LSR, 426

MAKHEX, 51
Maschinen-Code, 102
Maschinencode, 99, 100
Maschineninstruktion, 99
Maschinenprogramm, 48
  schützen, 434
Mikroprozessor, 9
MMU, 26
MMU-Register, 29
MMU-Wert, 69
Mnemonic, 100, 103
Monitor, 9, 47, 48, 99
Monitorprogramm, 48, 99
movabs, 86
Musik-Synthesizer, 22

Negativ-Bit, 408
Negativ-Flag, 408
NMI, 65, 408
NMI-Interrupt, 409
NOP, 426, 438
NTSC, 17
Null-Flag, 407

Oberton, 20
Object-Code, 100
Opcode, 101
Operand, 101, 103
Operation, arithmetische, 406, 406, 407
  logische, 406
Operations-Code, 101
ORA, 427
Overflow-Flag, 407

Paddle, 25
Page, 411
PAL, 17
Parallel-Schnittstelle, 43
Peripherie-Baustein, 11, 47
PHA, 427
PHP, 427
PLA, 428
PLAY-Funktion, 47
PLP, 428
Port, 9
Port-Leitung, 410
Portleitung, 39
PRA, 38
PRB, 38
Programm laden 76
  vergleichen 76
Programm-Adresse, neue, 102
Programmspeicher, 405
Programmzähler, 405
Prozessor-Reset, 409
Prozessorgeschwindigkeit, 12
Prozessorregister, 48
Pulsbreite, 20
Punktreihe, 60
putbyt, 91
PUTHEX, 51
PUTHXS, 51
putind, 87
putvdb, 85
putvdc, 83
PUTWRD, 51

Quarz, 409
Quell-Code, 100, 102

RAM einrichten 71
RAM-Bank, 88
RAM-Konfigurations-Register, 88
RAM-Routine,indirekte 30
Random-Register, 25
Raster-Interrupt, 64
Rauschen, 22
Rauschgenerator, 22
Rechteckschwingung, 22
Rechteckspannung, 20
Register,internes 408
  retten, 406
Release, 21, 22
REPEAT-Test, 58
Reset, 408, 409
Reset-Taster, 65
RESTORE-Taste, 42
Ringmodulator, 21
RLA, 438
ROL, 428
ROR, 429
Routine, 80
RRA, 439
RS232-Schnittstelle, 38, 42, 65, 75
rstsys, 82
RTI, 429
RTS, 429
Rücksprungadresse, 406

Sägezahnschwingung, 21
SBC, 430
Schnittstelle, serielle, 43
Schwingungsform, 15
SDR, 43
SEC, 430
SED, 430
SEI, 431
Sekundäradresse, 68, 68
  ausgeben, 72
setadr, 82
setit, 83
setsys, 81
Shift-RUN/STOP-Taste, 57
SID, 15, 20
SLO, 439
Sofware-Interrupt, 407
Source-Code, 100
Spalte, 58
Spaltenoffset, 59
Speicher-Aufteilung, 28
Speicher-Verwaltungs-Baustein, 26
Speicheradresse, höchste, 72
  unterste, 73
Speicherbank, 30
  aktuelle, 81
  beliebige, 69
Speicherbereich
  reservierter, 406
Speicherplatz, 9
Speicherseite vertauschen 89
Speicherverteilung, 69, 88
  neue, 89
Sprite, 12
Sprite-Kollision, 47
Sprung, absoluter, 413
  indirekter, 413
Sprungbefehl, bedingter, 413
Sprungleiste, 53, 65
SRE, 440
STA, 431
Stack, 406
Stack-Bereich, 409
Stapeladresse,aktuelle 406
Stapelbereich, 409
STASH, 31
Status,aktueller 74
Status-Bit, 406
Status-Register, 406
Steckerbelegung, 92
Steuertaste, 47
Steuerzeichen, 58
Stop-Taste lesen, 78
Stummschalten, 25
STX, 431
STY, 432
Sustain, 21, 22
swap, 89
SYNC, 21
Systemabsturz, 30

Systemparameter, 65
Systemroutine, 65
Systemuhr, 77
    aktualisieren, 78
    setzen, 77

TA, 39
Takt, 409
Taktrate, 43
TALK, 72
    ausgegeben, 74
Tastatur, 38
Tastatur-Abfrage, 58, 73
Tastatur-Puffer, 57
Tastaturmatrix, 12
Tastaturtabelle, 58
Tastendruck, 57
Tastennummmer, 58
TAX, 432
TAY, 432, 434
TB, 39
tcolor, 57
Testbit, 21
Text ausgeben, 70
    überspringen, 70
Tiefpaß-Filter, 24
Timeout-Flag, setzen, 73
Timer, 39, 409
Timer-Register, 39
TOD, 41
Tongenerator, 17
TOP, 440
TSX, 433
TXA, 433
TXS, 433

Uhrzeit, 77
    auslesen, 41
    einschreiben, 41
Unlisten ausgegeben, 74
unpack, 52
UNTALK, 74
Unterbrechung, 407
Unterprogramm, 69
Unterstrich-Cursor, 60
User-Port, 38

VDC, 33, 56
VDC 8563, 35
VDC-RAM, 34, 82, 83, 83, 84, 85, 86, 86, 87, 88
VDC-Steuerung, 80
Vektor einrichten, 71
Vektorentabelle, 71
Verweistabelle, 101, 105
Verzweigungsbefehl, 407
Verzweiungsbefehl, 406
VIC, 11, 12
Video-Chip initialisieren, 55, 71
Vorzeichen prüfen, 408
Vorzeichenstelle, 407

Wah-Wah-Effekt, 24
Wiederholung, 58

Z80, 9, 405
Zahl, vorzeichenbehaftete, 407
Zahlensystem,beliebiges 53
    dezimales, 100
    duales, 100
    hexadezimales, 100
    oktales, 100
Zehnertastatur, 12
Zeichen
    inverses, 56
    lesen, 78, 84, 86, 88
    schreiben, 83, 85, 86, 87
    übertragen, 92
Zeichengenerator, 64
Zeichenposition, aktuelle, 59
Zeichensatz, 33, 56
    aktueller, 64
Zeichensatz-ROM, 61
Zeile, 58
Zeilenbreite, 58
Zeilennummer, 101
Zeitmessung, 39
Zero-Flag, 407
Zero-Page, 31, 409, 411
Zwei-Byte-Zeiger, 413
Zwischenspeicher, 406

Bücher zum Commodore 128 PC



H. Ponnath

Grafik-Programmierung C128

1. Quartal 1986, ca. 250 Seiten
inkl. Beispieldiskette

Ein mächtiges Werkzeug hat der Anwender von Computergrafik mit dem Basic 7.0 des Commodore 128 PC in den Händen! Was man damit alles anfangen kann, soll Ihnen dieses Buch zeigen: hochauflösende Grafik, Multicolorbilder, Sprites und Shapes werden anhand von vielen Beispielprogrammen besprochen. Die Videochips und ihre Möglichkeiten sind ebenso Thema wie einige nützliche Assemblerroutinen, die Speicherorganisation, der 80-Zeichen-Bildschirm und vieles andere mehr. Außerdem enthält das Buch eine Diskette mit allen Programmen.

Best.-Nr. MT 90202
ISBN 3-89090-202-2
DM 52,-/sFr. 47,80/öS 405,60



P. Rosenbeck

Das Commodore 128-Handbuch

Juli 1985, 383 Seiten

Dieses Buch sagt Ihnen alles, was Sie über Ihren C128 wissen müssen: die Hardware, die drei Betriebssystem-Modi und was die CP/M-Fähigkeit für Ihren Computer bedeutet. Aber Sie werden irgendwann Lust verspüren, tiefer in Ihren C128 einzusteigen. Auch dafür ist gesorgt: an einen Assemblerkurs, der Ihnen zugleich die Funktionsweise des eingebauten Monitors nahebringt, schließen sich Kapitel an, die mit Ihnen auf Entdeckungsreise ins Innere der Maschine gehen. Daß die Reise spannend wird, dafür sorgen die Beispiele, aus denen Sie viel über die Interna des Systems lernen können – bis hin zur Grafik-Programmierung.

Best.-Nr. MT 809
ISBN 3-89090-195-6
DM 52,-/sFr. 47,80/öS 405,60



J. Hückstädt

BASIC 7.0 auf dem Commodore 128

Juli 1985, 239 Seiten

Das neue BASIC 7.0 des C128 eröffnet mit seinen ca. 150 Befehlen ganz neue Dimensionen der BASIC-Programmierung. Es ermöglicht dem Anfänger den einfachen und effektiven Zugriff auf die erstaunlichen Grafik- und Tonmöglichkeiten des C128; der Fortgeschrittene findet die nötigen Informationen für (auch systemnahe) Profi-Programmierung mit strukturierten Sprachmitteln.
An praxisnahen Beispielen (wie z.B. der Dateiverwaltung) zeigt der Autor auf, wie man die für den 128er typischen Merkmale und Eigenschaften (Sprites, Shapes, hochauflösende Grafik, Musikprogrammierung und Geräusche) optimal nutzt!.

Best.-Nr. MT 808
ISBN 3-89090-170-0
DM 52,-/sFr. 47,80/öS405,60

Markt & Technik-Fachbücher erhalten Sie bei Ihrem Buchhändler

Markt&Technik
BUCHVERLAG

Hans-Pinsel-Str. 2, 8013 Haar bei München