Korrekturen und Ergänzungen zum deutschen C128-Handbuch

Marco Baye

17. Dezember 2014

Zusammenfassung

Dies ist eine Auflistung aller Auslassungen und sachlichen Fehler, die mir im deutschen C128-Handbuch aufgefallen sind. Sie bezieht sich auf die Ringbuchversion, Art.-Nr. 580128, Copyright 1985 (noch ohne den C128D auf dem Cover).

Version 1 dieses Texts erschien in Ausgabe 1997-04 der Zeitschrift *Computer-Flohmarkt*.

Teilweise wurde Version 1 auch bei der Erstellung des digitalen Handbuchs für iDOC beachtet.

Version 2 dieses Texts erschien am 11. Dezember 2014 im C128-Bereich von www.forum64.de als Textdatei.

Version 3 dieses Texts erschien am 15. Dezember 2014 als PDF-Datei ebenda. Es wurden noch die im Thread und im C64-Wiki aufgeführten Fehler hinzugefügt.

Version 4 dieses Texts erschien am 17. Dezember 2014, ebenfalls als PDF-Datei. Es wurde die Anmerkung zu `BUMP` (Seite 4-26 im Handbuch, hier 4.9) korrigiert.

Inhaltsverzeichnis

Kapitel 1

Allgemeine Anmerkungen

Dieser Abschnitt enthält Erläuterungen, die keiner bestimmten Seite zugeordnet werden können. Alle weiteren Abschnitte haben eine Seitenreferenz als Überschrift.

1.1 Die Farbpaletten

Die beiden Videochips des 128ers, *VIC* und *VDC*, verwenden unterschiedliche Farbpaletten. Die Zuordnung der BASIC-Farbnummern erreicht zwar eine große Übereinstimmung, aber bei zwei Codes lässt sich der Unterschied nicht ignorieren:

BASIC7-Code	VIC-Farbe	VDC-Farbe
9	orange	dunkel-lila
12	dunkelgrau	dunkel-türkis

Je nach RGBI-Monitor oder -Kabel kann es auch passieren, dass auf dem VDC-Bild die Farbe Braun eher als dunkles Gelb dargestellt wird — darauf hat der Rechner aber keinen Einfluss[1].

1.2 CONTROL-4 ist nicht grün

Wie man die Farbe Cyan/Türkis als „grün“ bezeichnen kann, verstehe ich beim besten Willen nicht; vermutlich handelt es sich um einen Übersetzungsfehler. Dieser Fehler zieht sich durch das gesamte Handbuch (z.B. auf den Seiten 4-35, 4-135, 4-138, 5-75, 5-76, 5-79).

[1] Aus diesem Grund könnte man auch argumentieren, dass der VDC selbst *gar keine Palette hat* — er kennt eigentlich nur Farbnummern.

1.3 Relative Koordinaten

Pixelkoordinaten müssen in BASIC7 nicht unbedingt absolut angegeben werden, es geht auch relativ, und zwar wahlweise kartesisch oder polar. Beispiel:

```
DRAW TO 100, 100
```

zieht eine Linie bis an die ganz normal angegebene Endposition.
Man kann aber auch Vorzeichen angeben:

```
DRAW TO +10, -20
```

ist eine relative Angabe, der neue Zielpunkt befindet sich zehn Pixel weiter rechts und zwanzig Pixel weiter oben. Man kann auch einen Wert absolut und den anderen relativ angeben.

Für Polarnotation setzt man ein Semikolon statt eines Kommas:

```
DRAW TO 20; 45
```

zieht eine Linie zu einem Zielpunkt in einer Entfernung von zwanzig Pixeln unter einem Winkel von 45 Grad. Der Winkel wird im Uhrzeigersinn gemessen, Null steht für „oben“.

Diese alternativen Formate sind nicht nur bei allen Grafikbefehlen möglich, sondern z.B. auch bei `MOVSPR`.

1.4 Das Color-RAM

Beim C128 hat das Color-RAM einen zweiten Layer bekommen, vermutlich damit der Benutzer zwischen Text- und MultiColor-Grafikbildschirm hin- und herschalten kann, ohne die jeweilige Farbinformation zu verlieren. Das Handbuch verschweigt diesen zweiten Layer des Color-RAMs aber leider komplett: Nur an einer einzigen Stelle wird der „Farbspeicher“ überhaupt erwähnt (Seite B-14), aber die dort gegebene Information ist falsch. Da die Auswahl des gewünschten Color-RAM-Layers über den Prozessorport getroffen wird, habe ich die nötigen Informationen in der Anmerkung zu Seite H-7 (H.1) untergebracht, denn nur dort wird der C128-Prozessorport erwähnt.

1.5 Bug in der `SCRATCH`-Rückmeldung

Bei den Commodore-Diskettenlaufwerken ist nicht unbedingt Verlass auf die Angaben in der `FILES SCRATCHED`-Meldung: Die Anzahl der gelöschten Dateien wird nur mit zwei Stellen angezeigt, so dass nach dem Löschen von mehr als 99 Dateien in einem einzigen Durchgang ein falscher Wert ausgegeben wird. Diese Information könnte man auf den Seiten 4-112, 6-13 und/oder 8-6 gebrauchen.

Kapitel 2

Allgemeine Eigenschaften von BASIC

2.1 Seite 2-4

Bei den reservierten Wörtern fehlen `GO`, `OFF` und `QUIT`, außerdem fehlt bei `SPRCOLOR` ein „r“.

2.2 Seite 2-10

Es ist nur sehr schlecht zu erkennen: Ganz unten auf der Seite steht

```
"U ">"U"
```

Man beachte das Leerzeichen im ersten String, denn ohne dieses Leerzeichen ergibt diese Vergleichsoperation nicht -1, sondern 0.

Kapitel 3

Eingeben und Verwalten von BASIC-Programmen

3.1 Seite 3-3

Die Commodore-Taste hat angeblich zwei Funktionen; dann werden aber drei aufgezählt. Die dritte ist missverständlich formuliert, da die Eingabe von Grafikzeichen nicht an den DIN-Modus gebunden ist.

Kapitel 4

C128-Modus

4.1 Seite 4-3

Um softwaremäßig zwischen DIN- und ASCII-Modus umzuschalten, muss vor den angegebenen POKEs auch noch ein

```
POKE 0, PEEK(0) OR 64
```

erfolgen.

4.2 Seite 4-5

Bei den **ESC**-Sequenzen fehlt der Buchstabe **O**. Die gleiche Funktion wie die Sequenz **ESC O** hat übrigens die Sequenz **ESC ESC**.

4.3 Seite 4-6

Eine nahezu vollständige Tabelle von `CHR$`-Codes findet man in Anhang A, aber hier hätte ruhig erwähnt werden können, dass man mit **CONTROL** und den Buchstabentasten **A** bis **Z** die Codes 1 bis 26 erzeugen kann.

4.4 Seite 4-7

Beim zweiten `CHR$`-Beispiel fehlt das Dollarzeichen. Weiter unten wird beschrieben wie man die DIN-Unterstützung abschaltet. Die Aussage, dass man das nur durch einen Reset rückgängig machen könne, ist natürlich falsch. Bei der Erklärung der Accent-Taste wird behauptet, man könne per Leertaste einen Accent ohne Zeichen darunter erzeugen — das funktioniert aber nicht.

4.5 Seite 4-13 (APPEND)

Neben *SEQ*- und *USR*-Dateien kann APPEND auch auf *PRG*-Dateien angewendet werden.

Alle gängigen Commodore-Diskettenlaufwerke haben den Fehler, dass bei Verwendung von APPEND die im Verzeichnis angezeigte Blockanzahl der Datei immer mindestens um Eins erhöht wird, selbst wenn die neuen Daten noch in den aktuell letzten Block passen. Der Fehler steckt nicht in der BASIC-Anweisung, sondern im Laufwerks-DOS, tritt also auch bei Verwendung von BASIC2 auf.

4.6 Seite 4-13/14 (ASC())

Hier fehlt ASC(). Diese Funktion gibt es zwar bereits im BASIC2, aber im BASIC7 wurde sie verändert: Wendet man sie auf einen Leerstring an, liefert sie nun keinen Fehler mehr wie im 64er-Modus, sondern Null.

4.7 Seite 4-17 (BANK)

Hier fehlt die Information, welche Bedeutung die sechzehn verschiedenen Werte haben:

<table>
<tr><th rowspan="2">BANK</th><th rowspan="2">Wert in $ff00</th><th colspan="4">Speicher im Adressbereich</th></tr>
<tr><th>$0000–$3fff</th><th>$4000–$7fff</th><th>$8000–$bfff</th><th>$c000–$ffff</th></tr>
<tr><td>0</td><td>$3f</td><td colspan="4">RAM 0</td></tr>
<tr><td>1</td><td>$7f</td><td colspan="4">RAM 1</td></tr>
<tr><td>2</td><td>$bf</td><td colspan="4">RAM 2</td></tr>
<tr><td>3</td><td>$ff</td><td colspan="4">RAM 3</td></tr>
<tr><td>4</td><td>$16</td><td colspan="2">RAM 0</td><td rowspan="4">U36</td><td rowspan="8">IO</td><td rowspan="4">U36</td></tr>
<tr><td>5</td><td>$56</td><td colspan="2">RAM 1</td></tr>
<tr><td>6</td><td>$96</td><td colspan="2">RAM 2</td></tr>
<tr><td>7</td><td>$d6</td><td colspan="2">RAM 3</td></tr>
<tr><td>8</td><td>$2a</td><td colspan="2">RAM 0</td><td rowspan="4">Ext</td><td rowspan="4">Ext</td></tr>
<tr><td>9</td><td>$6a</td><td colspan="2">RAM 1</td></tr>
<tr><td>10</td><td>$aa</td><td colspan="2">RAM 2</td></tr>
<tr><td>11</td><td>$ea</td><td colspan="2">RAM 3</td></tr>
<tr><td>12</td><td>$06</td><td colspan="2" rowspan="4">RAM 0</td><td>U36</td><td rowspan="4">Ed</td><td rowspan="4">Kernal</td></tr>
<tr><td>13</td><td>$0a</td><td>Ext</td></tr>
<tr><td>14</td><td>$01</td><td colspan="2" rowspan="2">BASIC7</td><td>CS</td></tr>
<tr><td>15</td><td>$00</td><td>IO</td></tr>
</table>

Mit „U36“ ist hier der freie ROM-Sockel im 128er gemeint, während „Ext“ ein ROM am Expansionsport bezeichnet. „BASIC7“, „Ed“ (Editor), „CS“

(Charset) und „Kernal“ beziehen sich auf die vorhandenen System-ROMs. Dabei ist zu beachten, dass die RAM-Bänke 2 und 3 nicht existieren und stattdessen die RAM-Bänke 0 und 1 benutzt werden, d.h. die BASIC-Anweisung `BANK 7` hat faktisch den gleichen Effekt wie `BANK 5`. Der Defaultwert nach dem Einschalten ist übrigens `BANK 15`.

4.8 Seite 4-24 (`BSAVE`)

Die Erklärungen für „Dateiname“ und „Bank“ beziehen sich fälschlicherweise aufs Laden anstatt aufs Speichern.

4.9 Seite 4-26 (`BUMP()`)

Die Funktion `BUMP()` gibt keine Spritenummer (1–8) zurück, sondern den Inhalt des entsprechenden VIC-Kollisionsregisters (0–255). Darin steht jedes gesetzte Bit für ein kollidiertes Sprite. Das aufgeführte Beispiel mit `ON...GOTO` kann so also nicht funktionieren und ist auch nicht sinnvoll machbar, denn schließlich kann jederzeit mehr als nur ein Sprite kollidiert sein.[1]

4.10 Seite 4-27 (`CATALOG`)

CATALOG ist ein *Extended Token* und benötigt daher ein Byte mehr als DIRECTORY. Das muss zwar nicht unbedingt im Handbuch stehen, dürfte aber einen Programmierer interessieren.

4.11 Seite 4-28 (`CHAR`)

Hier fehlen einige wichtige Informationen; und unter „Bemerkungen“ sind beide Absätze über den Mehrfarbenmodus falsch. Aber der Reihe nach:

4.11.1 Allgemein

Der Parameter **Invers** darf offiziell nur die Werte 0 oder 1 annehmen. Tatsächlich kann man die Werte 0 bis 255 benutzen, es wird aber immer nur das unterste Bit beachtet. Das heißt, gerade Zahlen werden wie Null, ungerade Zahlen werden wie Eins behandelt.

Ist der Text/Grafik-Splitscreen aktiv, wirkt `CHAR` immer auf die Grafik.

[1] Ältere Versionen dieser Fehlerliste enthielten die Information, `v = BUMP(2)` funktioniere wie im Handbuch beschrieben — das war leider falsch.

4.11.2 Textmodus

Im Textmodus darf nur Farbquelle 0 oder 1 benutzt werden, der Wert hat aber gar keinen Effekt — es wird immer die aktuelle Textfarbe benutzt. Da hier Steuerzeichen beachtet werden, kann man damit natürlich auch mitten im String die Zeichenfarbe wechseln.

Invers sorgt lediglich dafür, dass vor dem Textstring ein `CHR$(18)` und danach ein `CHR$(146)` ausgegeben werden. Aktiviert man vorher, z.B. per

```
PRINT CHR$(34);
```

den Quote-Modus, werden Steuerzeichen nicht ausgeführt, sondern angezeigt.

4.11.3 Grafikmodus

Im Grafikmodus werden Steuerzeichen nicht ausgeführt, sondern als Zeichen ausgegeben.

BASIC7 hat kein offizielles Interface, um im Grafikmodus auf den Kleinbuchstaben-Zeichensatz umzuschalten, dies kann jedoch per POKE erreicht werden:

```
POKE 4588, 216
```

... schaltet auf Kleinschrift/Großschrift (ab Adresse `$d800` in Bank 14),

```
POKE 4588, 208
```

... schaltet zurück auf Großschrift/Grafikzeichen (ab `$d000` in Bank 14). Mit entsprechend niedrigen Werten könnte man auch auf eigene RAM-Charsets schalten; bei eingeschalteter Grafik hat man aber keine große Auswahl, was den freien Speicher angeht. Am ehesten bieten sich an:

`POKE 4588, 20:REM` Charset von `$1400` bis `$1800`
`POKE 4588, 24:REM` Charset von `$1800` bis `$1c00`

Wenn die CHAR-Anweisung die Zeichensatzdaten liest, greift sie immer nur auf die ersten 128 Zeichenmuster zu (= vier Pages = 1 KiB). Verlangt man über den **Invers**-Parameter eine Invertierung, so wird diese tatsächlich „zu Fuß“ durchgeführt; die Anweisung verlässt sich also nicht darauf, dass hinter den 128 Zeichenmustern noch invertierte Muster kommen (was beim ROM-Charset der Fall ist, bei einem eigenen Zeichensatz aber nicht unbedingt sein muss).

4.11.4 HiRes-Grafik

Im HiRes-Modus bestimmt die Farbquelle, welcher Teil der Farbkacheln verändert wird:
0 setzt die Hintergrundfarbe, die Vordergrundfarben bleiben erhalten.
1 setzt die Vordergrundfarbe, die Hintergrundfarben bleiben erhalten.

Die Bitmuster der Zeichen werden durch die Farbquelle nicht beeinflusst, man kann sie nur mit **Invers** invertieren.

4.11.5 MultiColor-Grafik

Der ROM-Zeichensatz eignet sich eigentlich nicht zur Darstellung auf der MultiColor-Grafik, da diese eine halbierte X-Auflösung hat. Mit einem eigenen, speziell angepassten Charset und den oben angegebenen POKEs kann es natürlich dennoch sinnvoll benutzt werden, aber selbst dann bleibt eine Unschönheit: Man kann keine „echten“ MultiColor-Charsets nehmen, denn die Char-Anweisung beachtet von allen Bitpaaren der Zeichensatzmuster immer nur das höherwertige Bit. Das Byte `%00011011` in einem Zeichensatz würde also so behandelt:

`%00......`	linkes Bit ist gelöscht, d.h. Pixel wird gelöscht
`%..01....`	linkes Bit ist gelöscht, d.h. Pixel wird gelöscht
`%....10..`	linkes Bit ist gesetzt, d.h. Pixel wird gesetzt
`%......11`	linkes Bit ist gesetzt, d.h. Pixel wird gesetzt

Auf dem MultiColor-Grafikbildschirm resultiert dieses Bitmuster also in zwei gelöschten und zwei gesetzten MC-Pixeln, aber nie in vier verschiedenfarbigen Pixeln. Welche Farbquellen zum Löschen und zum Setzen benutzt werden, hängt von **Farbquelle** und **Invers** ab, wobei die Farbquelle 0 signifikant anders behandelt wird als die Farbquellen 1 bis 3:

Farbquellen 1, 2 und 3: Die zu setzenden Pixel werden mit der gewählten Farbquelle dargestellt. In den Farbkacheln wird auch nur die gewählte Farbquelle verändert (auf ihren aktuellen Wert gesetzt). Die zu löschenden Pixel bekommen Farbquelle 0. **Invers** invertiert einfach nur die Bitmuster der Textzeichen, so dass man statt „Schrift in Farbquelle 1/2/3 auf Farbquelle 0“ auch „Schrift in Farbquelle 0 auf Farbquelle 1/2/3“ ausgeben kann.

Farbquelle 0: Die Farbkacheln werden *überhaupt nicht verändert*, und der **Invers**-Parameter bestimmt nun die Hintergrundfarbe:
Wenn **Invers** = 0, so wird in „Farbquelle 1 auf Farbquelle 2“ dargestellt.
Wenn **Invers** = 1, so wird in „Farbquelle 1 auf Farbquelle 3“ dargestellt.

Die theoretisch möglichen Farbquell-Kombinationen „2 auf 1“, „3 auf 1“, „2 auf 3“ und „3 auf 2“ können mit `CHAR` also nicht erzeugt werden.

4.12 Seite 4-35 (COLOR)

Die COLOR-Anweisung akzeptiert nicht nur zwei Argumente, sondern auch noch ein optionales drittes. Die dritte Zahl gibt die Helligkeit an — dies ist ein Überbleibsel vom BASIC3.5 des C16/+4, das beim 128er keinerlei Effekt hat.

4.13 Seite 4-36 (CONCAT)

Im Beispiel steht „D$1“, korrekt wäre „D1$“.

4.14 Seite 4-40 (DEC)

Bei „Format“ steht „v$ =“, korrekt wäre „v =“.

4.15 Seite 4-45 (DO...LOOP)

Der Softwarestack des BASIC7 ist groß genug, um DO-LOOP-Schleifen in bis zu 102 Ebenen schachteln zu können.

4.16 Seite 4-48 (DOPEN)

Die Fehlermeldung FILE ALREADY OPEN gibt es nicht, es ist FILE OPEN gemeint.

4.17 Seite 4-50 (DRAW)

Zeile 110 muss heißen

```
110 DRAW TO 25, 30
```

Ein Komma hinter dem (ggfs. nicht vorhandenen) Farbcode ist nur nötig, wenn Startkoordinaten folgen.

4.18 Seite 4-61 (FETCH)

Das angegebene Beispiel entspricht nicht dem angegebenen Format. Außerdem beschreibt der letzte Absatz der Bemerkungen nicht FETCH, sondern STASH.

4.19 Seite 4-66 (GRAPHIC)

Der SPRDEF-Befehl (Seite 4-120 im Handbuch, hier 4.33) führt intern

```
GRAPHIC 1,1
```

aus. Benutzt man in einem Programm sowohl Grafik als auch benutzerdefinierte Funktionen (vgl. DEF FN, siehe Seite 5-13 im Handbuch, hier 5.1), so sollten die Funktionen erst nach der Reservierung des Grafikspeichers definiert werden — andernfalls können die Funktionen nicht korrekt aufgerufen werden, da beim Verschieben des Programms im Speicher die Funktionsreferenzen nicht mitverschoben werden.

4.20 Seite 4-68 (GSHAPE)

Im ersten Absatz wird auf MOVSPR verwiesen, damit hat diese Anweisung aber rein gar nichts zu tun.

Modus 4 entspricht einer logischen *Exklusiv-Oder*-Operation.

4.21 Seite 4-77 (KEY)

Will man die Funktionstasten aus einem Programm heraus abfragen, sind die vorgegebenen Belegungen eher hinderlich als brauchbar. Man könnte die Tasten zwar explizit mit den vom C64 bekannten CHR$-Codes belegen, aber dann ärgert sich der Nutzer nach dem Verlassen des Programms über die nicht mehr wie gewohnt funktionierenden F-Tasten. Besser ist:

```
POKE 828, 183
```

Dies deaktiviert die Belegungen komplett; die F-Tasten haben dann die CHR$-Werte wie am C64. **SHIFT-STOP** und **HELP** haben dann die Codes 131 und 132. Rückgängig macht man diese Aktion durch:

```
POKE 828, 173
```

Die Tasten funktionieren danach wieder normal.

4.22 Seite 4-78/79 (MID$())

Hier fehlt MID$(). Diese Funktion gibt es zwar bereits in BASIC2, aber in BASIC7 kann MID$() auch als Anweisung benutzt werden. Beispiel:

```
MID$(A$, 4, 7) = B$
```

Diese Anweisung überschreibt `A$` ab dem vierten Zeichen mit den ersten sieben Zeichen von `B$`. Ist `B$` kürzer als die (optionale!) Längenangabe, wird die kleinere der beiden Längen genommen.

Der String kann nur überschrieben, aber nicht verlängert werden — ein Versuch ergibt einen `?ILLEGAL QUANTITY ERROR`.

Der String wird tatsächlich an Ort und Stelle im Speicher geändert, es wird kein neuer Eintrag auf dem Stringheap erzeugt.

4.23 Seite 4-80/81 (`OFF`)

Hier fehlt `OFF`. Die Eingabe dieses Schlüsselworts erzeugt allerdings nur einen `?UNIMPLEMENTED COMMAND ERROR`.

4.24 Seite 4-84 (`PLAY`)

Bevor die Länge der Noten gesetzt wird, sind Viertelnoten die Voreinstellung.
„`.N`“ wird erklärt mit: „Die folgende Note wird als punktierte Note (die Hälfte ihres Wertes) gespielt.“
Korrekt wäre: „Die folgende Note wird als punktierte Note (um die Hälfte ihres Wertes verlängert) gespielt.“

4.25 Seite 4-93 (`PUDEF`)

In Zeile 10 fehlt ein Leerzeichen; statt

```
PUDEF ".,"
```

ist

```
PUDEF " .,"
```

gemeint.

4.26 Seite 4-93/94 (`QUIT`)

Hier fehlt `QUIT`. Die Eingabe dieses Schlüsselworts erzeugt allerdings nur einen `?UNIMPLEMENTED COMMAND ERROR`.

4.27 Seite 4-94 (`RCLR()`)

`RCLR` erlaubt das Lesen der Farbe von sieben verschiedenen „Farbquellen“, und `RDOT` auf der Folgeseite erlaubt das Auslesen der Farbquelle unter dem

Pixelcursor. Leider ist es dennoch nicht möglich, die tatsächliche Farbe eines Pixels zu ermitteln, denn die „Farbquellen“ können ja in jeder Farbkachel unterschiedlich sein — und `RCLR` gibt immer nur den zuletzt eingestellten Wert für jede der Farbquellen zurück.

4.28 Seite 4-95 (`RDOT()`)

Ein Druckfehler: `n` darf natürlich nicht 3 sein.

4.29 Seite 4-96 (`RECORD`)

Es steht zwar auf dieser Seite, aber meines Erachtens nicht deutlich genug: *Sowohl bei der Datensatznummer als auch bei der Bytenummer beginnt die Zählung bei EINS*! Ich bitte dies bei Berechnungen zu beachten.

Verwendet man dennoch den Wert Null, so wird dieser vom Diskettenlaufwerk stillschweigend wie der Wert Eins behandelt.

4.30 Seite 4-102 (`RGR()`)

Die Funktion

```
v = RGR(n)
```

kann Werte bis neun liefern. Wenn sich der Cursor auf dem 80-Zeichen-Schirm befindet, liefert die Funktion den Modus des 40-Zeichen-Schirms plus fünf. Nach

```
GRAPHIC 3:GRAPHIC 5
```

liefert `RGR` also acht.

4.31 Seite 4-110 (`SCALE`)

`SCALE` kann virtuelle Auflösungen bis 32767 verarbeiten. Die angegebenen maximal 1023 sind ein Überbleibsel vom BASIC3.5 des C16/+4.

4.32 Seite 4-114 (`SLEEP`)

`SLEEP` kann man auch mit dem Argument Null benutzen. Ein

```
SLEEP 0
```

kehrt aber nicht sofort zurück, sondern wartet auf den nächsten Interrupt. Da dieser im 128er-Modus ein Rasterinterrupt des VIC-IIe-Videochips ist, kann man damit ein Programm mit dem Bildaufbau synchronisieren.

4.33 Seite 4-120 (SPRDEF)

SPRDEF kennt einen Kopierbefehl namens „C“, der nicht im Handbuch dokumentiert ist. Damit kann ein anderes Spritemuster in das aktuelle kopiert werden.

Was auch noch unbedingt erwähnt werden sollte: SPRDEF löscht den Grafikbildschirm — was verwunderlich ist, denn man könnte dieses Programm ebenso gut für den Textbildschirm schreiben.

Wurde zuvor noch keine Grafik benutzt, so werden 9 KiB BASIC-Speicher für die Grafik reserviert (vgl. GRAPHIC, Seite 4-66 im Handbuch, hier 4.19).

4.34 Seite 4-126 (STASH)

Wie schon bei FETCH auf Seite 4-61 stimmt auch hier das Beispiel nicht mit dem Format überein.

4.35 Seite 4-127 (SWAP)

Bei SWAP gilt das Gleiche, außerdem beschreibt auch hier der letzte Absatz der Bemerkungen nicht SWAP, sondern STASH.

4.36 Seite 4-127/128 (SYS)

Hier fehlt SYS. Diese Anweisung gibt es zwar bereits im BASIC2, aber im BASIC7 wurde SYS so erweitert, dass man gleich die Prozessorregister setzen kann. Beispiel:

```
SYS 4864, 1,, 3, 4
```

Diese Anweisung setzt den Akku auf 1, lässt das X-Register unverändert, setzt das Y-Register auf 3 und die Prozessorstatusflags auf 4 (insofern das möglich ist), bevor die Routine an Adresse 4864 aufgerufen wird.

4.37 Seite 4-128 (TEMPO)

Der erlaubte Bereich für „n“ beträgt nicht 0–255, sondern 1–255.
Die angegebene Umrechnungsformel

$$Dauer = 19.22/n$$

gilt nur für NTSC-Systeme. Bei PAL-Systemen wäre

$$Dauer = 23.06/n$$

korrekt.
Voreingestellt ist nicht der Wert 8, sondern der Wert 16.

4.38 Seite 4-139

Das Feld „Beachte“ ist unnötig, da bei einem `SYNTAX ERROR` eh automatisch die Grafik deaktiviert wird. Zeile 50 des Beispielprogramms (`LOCATE`) ist überflüssig, da in Zeile 60 Koordinaten angegeben werden. Der Satz „Zunächst müssen Sie den grafischen Cursor [...] setzen [...]“ ist aus genau diesem Grund daher fehl am Platz.

4.39 Seite 4-143

Hier wird wieder `SCALE` mit den falschen Maximalwerten genannt. Korrekt ist 32767.

4.40 Seite 4-159

Hier werden alle drei Fehler der `TEMPO`-Beschreibung wiederholt:

- der Wert Null ist gar nicht erlaubt
- voreingestellt ist 16 statt 8
- der Umrechnungsfaktor beträgt bei PAL 23.06 statt 19.22

4.41 Seite 4-160

Mit „R“ wird nicht der Ton gehalten, sondern nach dem aktuell gespielten Ton eine Pause entsprechender Länge eingelegt.

In der Tabelle wird als Voreinstellung für „Stimme“ der Wert 0 angegeben, korrekt wäre 1.

Bei „Filter“ ist die Bedeutung invertiert, korrekt wäre: „0 = aus, 1 = an“.

4.42 Seite 4-165

Es wird nicht erläutert, durch welche Filterkombination das Kerbfilter realisiert wird — ich nehme an, dies geschieht durch eine Verbindung von Tiefpass und Hochpass?

Kapitel 5

C64-Modus

5.1 Seite 5-13 (`DEF FN`)

Laut Handbuch kann eine ganze „Argumentliste“ übergeben werden; tatsächlich ist aber immer nur ein einziger Parameter erlaubt.

Wird nach einer Funktionsdefinition Grafikspeicher reserviert oder freigegeben (vgl. `GRAPHIC`, Seite 4-66 im Handbuch, hier 4.19), so kann die definierte Funktion nicht mehr korrekt aufgerufen werden. Will man beides nutzen, sollte man erst den Grafikspeicher reservieren und dann die Funktionen definieren (und den Grafikspeicher nie mehr freigeben).

5.2 Seite 5-15 (`DIM`)

`DIM` kann auch zum Anlegen nicht-dimensionierter Variablen benutzt werden.

5.3 Seite 5-23 (`GOSUB...RETURN`)

„Unterprogramme können in bis zu 23 Ebenen geschachtelt werden“. Im 128er-Modus sind es sogar 66.

5.4 Seite 5-68 (`TI`)

`TI` erlaubt keine direkten Zuweisungen, kann aber durch eine Änderung von `TI$` (Seite 5-69) verändert werden. Es handelt sich nur um zwei verschiedene Anzeigeformen von ein- und derselben Uhr.

5.5 Seite 5-70 (`USR()`)

Die Adresse des `USR`-Vektors ist falsch angegeben: Beim C64 ist es 785/786 und im 128er-Modus 4633/4634.

5.6 Seite 5-81

„Leider können Sie den Ball noch nicht sehen, er hat nämlich dieselbe Farbe wie der Hintergrund.“ — das stimmt nur für C64-Kernalversion 2; die meisten 64er (und der C64-Modus des 128ers) haben aber Kernalversion 3.

5.7 Seite 5-88

In Zeile 110 des Beispielprogramms ist der letzte `DATA`-Wert falsch, statt `224` muss da `244` stehen.

5.8 Seite 5-89

Dies ist zwar das 64er-Kapitel, aber da im 128er-Sprite-Kapitel die Spritezeiger totgeschwiegen wurden, kommt der Hinweis eben hier: Die Standardwerte für die Spritezeiger in den Speicherstellen 2040 bis 2047 sind im 128er-Modus die Zahlen 56 bis 63, so dass die Sprites im Speicher von `$0e00`–`$0ffe` liegen. Im Grafikmodus werden die Spritezeiger in den Speicherstellen 8184 bis 8191 erwartet, die vom System erzeugten Standardwerte sind natürlich ebenfalls 56–63.

Kapitel 6

Floppy-Disk-Betrieb mit BASIC

6.1 Seite 6-3

In Zeile 10 muss die letzte Variable nicht `FS`, sondern `FT` heißen.

6.2 Seite 6-7

In der Liste der Dateitypen fehlt `DEL` für gelöschte Dateien. Diese Einträge können zwar nur mit einem Diskmonitor erzeugt werden, aber ein Commodore-Laufwerk kann diesen Typ nun mal anzeigen, also gehört er in die Liste.

Seit dem Druck der C128-Handbücher sind noch diverse andere Typen hinzugekommen:

Mit dem 1581-Laufwerk wurde der Typ `CBM` für Partitionen eingeführt; und das DOS in den diversen CMD-Geräten kennt auch noch `DIR` für Verzeichnisse. Modernere Massenspeicher könnten noch weitere Typen bereitstellen.

6.3 Seite 6-10

`APPEND` kann nicht nur auf *SEQ*- und *USR*-Dateien angewendet werden, sondern auch auf *PRG*-Dateien.

Unter „`APPEND` [...]“ fehlt die Teilüberschrift „Datei öffnen im C64-Modus“, da die weiteren Beispiele den 64er-Modus betreffen.

6.4 Seite 6-11

Unter „`DCLOSE`“ fehlt die Teilüberschrift „Datei schließen im C64-Modus“.

6.5 Seite 6-18

Die Syntax des *Backup*-Befehls wird als

"D*Ziellaufwerk*=D*Quelllaufwerk*"

angegeben; korrekt wäre

"D*Ziellaufwerk*=*Quelllaufwerk*"

In der Praxis stünde da also entweder `"D0=1"` oder `"D1=0"`.

Kapitel 7

CP/M-Modus

Das separate CP/M-Handbuch wird hier nur erwähnt, damit die Kapitelnummern in diesem Text denen des Originalhandbuchs entsprechen. :)

Kapitel 8

Fehlermeldungen

8.1 Seite 8-1

Es fehlt „`37 BEND NOT FOUND`“.

8.2 Seite 8-2

Es fehlt „`41 FILE READ`“.

8.3 Seite 8-3

Es fehlt „`38 LINE NUMBER TOO LARGE`“. Dieser Fehler tritt auf, falls durch `RENUMBER` eine Zeilennummer über 64000 generiert werden würde.

8.4 Seite 8-5

Es fehlt „`40 UNIMPLEMENTED COMMAND`“. Dieser Fehler tritt bei Eingabe von `QUIT`, `OFF`, `KEY ON` und `KEY OFF` auf.
Es fehlt „`39 UNRESOLVED REFERENCE`“. Dieser Fehler wird durch `RENUMBER` erzeugt, falls eine nicht vorhandene Zeile angesprochen wird.

In der numerischen Zusammenfassung fehlen alle oben genannten Fehlermeldungen natürlich ebenfalls. Der C64 kennt alle Meldungen bis einschließlich 30, die übrigen bis 41 sind 128er-spezifisch.

Die Fehlercodes bis neun entsprechen den Kernal-I/O-Fehlernummern, ab zehn sind es dann reine BASIC-Fehlernummern — mit Ausnahme von `BREAK`, dieser Fehler hat bei Kernal-I/O-Funktionen die Nummer Null.

8.5 Seite 8-6

Es fehlt „`00, OK,00,00`“.

8.6 Seite 8-9

Nummer 66 heißt richtig „`ILLEGAL TRACK OR SECTOR`“.

Die numerischen Angaben bei Nummer 70 sind äußerst fragwürdig. Meines Wissens sind es gerade mal drei sequentielle oder eine relative Datei.

8.7 Seite 8-10

Der Text von Nummer 73 ist vom verwendeten Laufwerk abhängig und lautet nicht „`DOS MISMATCH`“.

Anhang A

Zeichencode-Tabellen, Steuercodes

A.1 Seite A-1

Der 128er-Modus benutzt andere Codes für das Blockieren und Entriegeln der **C=-SHIFT**-Tastenkombination als der 64er-Modus; daran kann man sehen, dass die Tabelle auf dieser Seite für den 64er-Modus gilt. Die Codes 96 bis 127 sind übrigens nicht über die Tastatur erreichbar, es werden stattdessen die Codes 192 bis 223 erzeugt.

A.2 Seite A-7

Die Codes 28 bis 31 stehen für „Rot“, „Cursor rechts“, „Grün“ und „Blau“.

A.3 Seite A-9

Der Code 131 steht für **SHIFT-STOP**; man bekommt ihn aber nur zu sehen, wenn man die F-Tasten-Strings abgeschaltet hat — siehe dazu die Anmerkungen zur `KEY`-Anweisung (Seite 4-77 im Handbuch, hier 4.21).

Anhang B

Speicherverwaltung (MMU)

B.1 Seite B-2

Das Speicherschema ist falsch, denn beide Function-ROMs (intern und extern) können nur den Bereich von `$8000` bis `$ffff` belegen und haben daher von `$4000` bis `$7fff` nichts zu suchen.

B.2 Seite B-4

Im drittletzten Absatz wird behauptet, ein PCR werde ins CR „geodert“. Falls damit ein *logisches ODER* gemeint sein sollte, ist das Unsinn. Im vorletzten Absatz sind mit „Bank 0“ und „Bank 1“ weder BASIC-`BANK`s noch CR-Werte gemeint, sondern lediglich PreConfiguration Registers. Um solchen Wirrwarr zu vermeiden, nenne ich die PCRs lieber „a“, „b“, „c“ und „d“.

Das System belegt die vier PCRs so:

PCR	Wert	Entsprechende BASIC-BANK:
a	`$3f`	0
b	`$7f`	1
c	`$01`	14
d	`$41`	keine

Der vierte Wert ähnelt der BASIC-BANK 14, basiert jedoch auf RAM 1 statt RAM 0. Mit dem BASIC-Befehl `BANK` lässt sich diese Konfiguration also nicht ansprechen.

B.3 Seite B-5

Im zweiten Absatz ist „aufgeteilt“ nicht wörtlich zu nehmen; tatsächlich hat jeder der beiden Teile des gemeinsamen RAM-Bereichs die konfigurierte Größe.

B.4 Seite B-8

Der I/O-Bereich umfasst den Bereich von `$d000-$dfff`, nicht `$d500-$d5ff`.

B.5 Seite B-11

Im letzten Absatz wird erneut der I/O-Bereich falsch angegeben; richtig ist `$d000-$dfff`.

B.6 Seite B-12

In der ersten Zeile ist der Zusammenhang zwischen Bitpositionen und MS-Leitungen falsch angegeben, korrekt ist:

Bit 5 korrespondiert mit MS0.
Bit 4 korrespondiert mit MS1.

Eine Seite zuvor, bei den Bits 3 und 2, stimmen die Angaben.

B.7 Seite B-13

Die Erklärung zu Bit 0 des MCR ist invertiert, korrekt ist:

Ist das Bit gelöscht, ist der Z80 aktiv.
Ist das Bit gesetzt, ist der 8502 aktiv.

Beim Reset wird dieses Bit gelöscht, so dass das System mit dem Z80 startet. Wenn dieser kein C64-Modul am Expansionsport findet, übergibt er die Kontrolle an den 8502.

B.8 Seite B-14

Der vierte Absatz ist falsch, denn die Umschaltung des Farbspeichers wird per Prozessorport kontrolliert. Weder *GAME* noch *EXROM* werden vom System als Ausgabebit genutzt.

B.9 Seite B-15

Im zweiten Absatz wird wieder die missverständliche Formulierung mit dem „Aufteilen“ des gemeinsamen Speicherbereichs benutzt. Richtig ist: Wenn man sowohl am unteren als auch am oberen Ende des Speichers gemeinsamen Speicher konfiguriert, so bekommt jeder der beiden Teile die konfigurierte Größe.

B.10 Seite B-16

In der letzten Zeile muss es nicht „`$01ff`“ heißen, sondern „`$0000`–`$01ff`“.

Anhang C

Maschinensprache-Monitor

C.1 Seite C-1

Als Banknummern können im Monitor natürlich alle Werte von `$0` bis `$f` verwendet werden, diese haben exakt dieselbe Konfiguration zur Folge wie die BASIC-BANKs. Unten beim Statusregister könnte man noch die Reihenfolge der Flags erwähnen, es ist `NV-BDIZC`.

C.2 Seite C-2

Hier fehlt der `J`(Jump)-Befehl, mit dem man ein Programm aufrufen kann.

C.3 Seite C-3

Es sollte erwähnt werden, dass der Monitor Zahlen in vier Formaten nicht nur verarbeiten, sondern auch umrechnen kann: Nach Eingabe einer Zahl mit Zahlensystem-Präfix (`$`, `+`, `&`, `%`) wird diese Zahl in allen vier Formaten angezeigt.

C.4 Seite C-4 (Assemble)

Der Operand muss nicht unbedingt hexadezimal angegeben werden — ohne Präfix ist das allerdings die Erwartung.

C.5 Seite C-6 (Disassemble)

Wird die zweite Adressangabe weggelassen, so richtet sich die Anzeige nicht nach der Anzahl der Bildschirmzeilen, sondern nach der Länge der disassemblierten Befehle: Es werden mindestens 21 Bytes disassembliert; je nach

Länge des letzten Befehls können es dann 22 oder 23 werden.
Im Beispiel steht fälschlicherweise ein Punkt vor dem Befehl.

C.6 Seite C-8 (Go)

Die im Beispiel genannte Adresse stimmt offensichtlich nicht mit der im Text überein.

C.7 Seite C-8/9 (Jump)

Hier fehlt `J` (Jump). Im Gegensatz zu `G` (Go) muss das Programm nicht auf *BRK*, sondern auf *RTS* enden.

C.8 Seite C-12 (Memory dump)

Der Versuch, ROM-Bereiche zu modifizieren, wird nicht mit einem Fragezeichen quittiert, sondern ausgeführt. Natürlich klappt es nicht, aber das sieht man nur an der erneuten Ausgabe des alten Inhalts.
Im Beispiel steht fälschlicherweise ein Punkt vor dem Befehl.

C.9 Seite C-13 (Registers)

Der Interruptvektor wird nicht angezeigt. Das machen zwar viele Monitore, aber nicht dieser.
Im Beispiel steht fälschlicherweise ein Punkt vor dem Befehl.

C.10 Seite C-14 (Save)

Bei „Bemerkungen“ ist die Reihenfolge der Bytes falsch angegeben, denn die ist bekanntlich Low-Byte/High-Byte.

C.11 Seite C-17 (Verify)

Im Beispiel steht fälschlicherweise ein Punkt vor dem Befehl.

C.12 Seite C-18 (eXit)

Im Beispiel steht fälschlicherweise ein Punkt vor dem Befehl.

Anhang D

Besonderheiten der DIN-Tastatur

Dieses Kapitel wurde nur angelegt, damit die Kapitelnummern in diesem Text denen des Originalhandbuchs entsprechen. :)

Anhang E

Registerzuordnungen beim SID und VIC

Genau genommen ist hier schon der Name des Kapitels falsch, denn schließlich werden auch die VDC-Register erläutert...

E.1 Seite E-1

Die Register 25 und 26 des SID enthalten die Werte der A/D-Wandler, also die Paddle/Mauskoordinaten.

E.2 Seite E-3

Der VIC des 128ers hat noch zwei Register mehr, diese fehlen im Schema:

Register 47 (`$d02f`) steuert mit den unteren drei Bits die zusätzlichen Ausgabeleitungen für die erweiterte Tastaturmatrix an.

Register 48 (`$d030`) kontrolliert über Bit 0 den Prozessortakt (0 steht für 1 MHz, 1 steht für 2 MHz) und über Bit 1 eine spezielle VIC-Testfunktion (wer keine Demos programmiert, sollte dieses Bit tunlichst auf Null belassen).

E.3 Seite E-4

Hier fehlt die Bedeutung der von `$d600` gelesenen Bits:

`%7.......`	*READY-Flag*: Gesetzt, wenn RAM-Zugriff abgeschlossen.
`%.6......`	Light pen: Gesetzt, wenn Lichtgriffel aktiviert wurde.
`%..5.....`	Border: Gesetzt, wenn sich der Rasterstrahl über oder unter dem Textfenster befindet.
`%...43210`	Version: VDC-Version (0, 1 oder 2)

E.4 Seite E-5

Bit 6 von Register 24 bezieht sich auf den ganzen Textscreen.

E.5 Seite E-6

Die Beschreibung für die unteren vier Bit von Register 25 stimmt nur für die VDC-Version 0. Spätere VDC-Versionen (1 und 2) schieben den Bildschirm nicht nach links, sondern nach rechts. Entsprechend müssen diese Bits auch anders initialisiert werden (Version 0 will `%00000`, die anderen wollen `%00111`). Das Kernal prüft bei der Initialisierung des VDC-Bildschirms die VDC-Version (die unteren fünf Bits von `$d600`), und schreibt dann den passenden Wert.

E.6 Seite E-7

Der zweite DRAM-Typ (Bit 4 von Register 28) lautet nicht „4164“, sondern „41464“.

Version 2 des VDC (nur im C128DCR zu finden) hat noch ein weiteres Register. Mit diesem Register 37 können die SYNC-Signale invertiert werden:

`%7.......`	Löschen des Bits invertiert HSync
`%.6......`	Löschen des Bits invertiert VSync

Das Kernal ignoriert dieses Register, der Defaultwert ist `$ff`.

Anhang F

Musiknotentabelle

Dieses Kapitel wurde nur angelegt, damit die Kapitelnummern in diesem Text denen des Originalhandbuchs entsprechen. :)

Anhang G

Besonderheiten im C64-Modus

Dieses Kapitel wurde nur angelegt, damit die Kapitelnummern in diesem Text denen des Originalhandbuchs entsprechen. :)

Anhang H

Organisation der Zero Page

H.1 Seite H-7

Die Speicherstellen 0 und 1 bilden den Prozessorport der 8502-CPU, wobei die einzelnen Bits diese Bedeutungen haben:

`%7.....` (nicht nach außen geführt)

`%.6.....` ASCII/DIN-Umschaltung (0: DIN, 1: ASCII)

Hier kann man den Zustand der Taste lesen bzw. selbst die Umschaltung auf DIN forcieren. Das geht auch im 64er-Modus und ist damit eine der wenigen Inkompatibilitäten des 128ers.

`%..5....` Datassette, Motor (0: an, 1: aus)
`%...4...` Datassette, Tasten (0: mind. eine gedrückt, 1: keine)
`%....3..` Datassette, Data Output

`%.....2.` VIC Charset (0 = ROM, 1 = RAM)

Beim C64 wird in der VIC-Speicherkonfiguration ab den Adressen `$1000` und `$9000` der ROM-Zeichensatz eingeblendet, so dass dort abgelegte eigene Grafikdaten nicht vom VIC „gesehen“ werden können. Im 128er-Modus erfolgt diese Einblendung sogar in allen vier VIC-Bänken, d.h. auch an den Adressen `$5000` und `$d000` — mit diesem Prozessorport-Bit lässt sich das Einblenden aber komplett deaktivieren.
Das Bit wird ständig vom System-Interrupt beeinflusst, ändern kann man die Information in Speicherstelle `$d9`.

`%......1.`	Color-RAM-Layer für VIC (0 = Grafik, 1 = Text)
`%.......0`	Color-RAM-Layer für CPU (normal 1)

Die untersten beiden Bits bestimmen, auf welchen Layer des Color-RAMs die CPU und der VIC zugreifen. Normalerweise sind beide Bits gesetzt. Im Grafikmodus wird das VIC-Bit gelöscht, damit die MC-Grafik ihre eigenen Farbkacheln hat.
Das CPU-Bit wird immer nur kurz geändert, wenn Grafikbefehle mit Farbquelle 3 ausgeführt werden. Will man also die MC-Grafik speichern, so sollte man vor dem Auslesen des Color-RAMs das CPU-Bit hier löschen, da man sonst den Farbspeicher des Textschirms ausliest.

H.2 Seite H-15

Adresse `$ee` (Label **COLUMNS**) gibt natürlich nicht die maximale Anzahl der Bildschirmfarben, sondern die maximale Bildschirmspalte an (39/79).

H.3 Seite H-17

Zwischen `$200` und `$2fc` befinden sich Unterprogramme zum Laden aus jeder Bank.

H.4 Seite H-18 und H-19

Hier fehlen Infos:

`$334`	Quote-Escape-Flag
`$33e`–`$349`	Tastaturtabellenzeiger
`$34a`	Tastaturpuffer
`$354`	Tab-Tabelle
`$35e`	Zeilenverknüpfungstabelle
`$380`	**CHRGET/CHRGOT**

Bei `$7f8` (2040) befinden sich die im Textmodus benutzten Spritezeiger.

H.5 Seite H-21

Hier ist die Reihenfolge der Adressen durcheinander gekommen: Zwischen `$0a2a` und `$1000` scheinen einige Informationen zu fehlen; diese finden sich aber ab der Seite H-23. Auf Seite H-24 gibt es dann einen Sprung von Adresse `$0fff` zu `$1158`; die dort fehlenden Informationen beginnen hier auf Seite H-21 und gehen bis H-22.

H.6 Seite H-23

Achtung, die Labels **SAV80C** und **SAV80D** gibt es nicht. Die Labels **CURCOL** bis **PALCNT** rücken mitsamt ihren Erklärungen zwei Zeilen hoch (also zwei Adressen runter).
In den Adressen `$0a37`, `$0a38` und `$0a39` werden während `LOAD/SAVE` der Systemtakt, das Sprite-Enable-Register und der Blanking Status gepuffert. `$0ac5` beeinflusst die automatische ASC/DIN-Umschaltung im Interrupt.

H.7 Seite H-24

Der Kassettenpuffer belegt den Speicher von `$0b00` bis `$0bbf`.

H.8 Seite H-26

`$11eb` und `$11ec` sind die Adress-High-Bytes des Text-Zeichensatzes und des von `CHAR` genutzten Zeichensatzes.

H.9 Seite H-29

Ein `$ff` in `$12fd` blockiert den Teil des System-Interrupts, der für die automatisch ablaufenden BASIC-Dinge zuständig ist: `MOVSPR`, `SOUND`, etc. Wenn bei `$1c00` das Video-RAM liegt, gilt dort:

HiRes-Modus: Das High-Nibble enthält die Farbe für die in der Bitmap gesetzten Bits („Farbquelle 1“, Vordergrund), das Low-Nibble enthält die Farbe für die in der Bitmap gelöschten Bits („Farbquelle 0“, Hintergrund).

MultiColor-Modus: Das High-Nibble enthält die Farbe für die Bitkombination `%01` („Farbquelle 1“), das Low-Nibble enthält die Farbe für die Bitkombination `%10` („Farbquelle 2“).
Die Farbe für die Bitkombination `%11` („Farbquelle 3“) kommt aus dem separaten Color-RAM, während die Farbe für die Bitkombination `%00` („Farbquelle 0, Hintergrund“) auf dem ganzen Bildschirm einheitlich ist und über das VIC-Register `$d021` festgelegt wird.

Im Grafikmodus liegen natürlich auch die Spritezeiger ab Adresse `$1ff8`.

Anhang I

BASIC-Abkürzungen

I.1 Seite I-2

Hier hätte explizit erwähnt gehört, dass man `PRINT#` nicht mit `?#` abkürzen kann. Wenn man das macht, sieht es nach dem `LIST`en zwar aus wie `PRINT#`, ist es intern aber nicht — und das Programm erzeugt dann einen `SYNTAX ERROR`.

I.2 Seite I-3

Die Abkürzung `getkE` für `GETKEY` scheint nicht viel Ersparnis zu bringen. Das liegt daran, dass der Befehl immer aus `GET` und `KEY` zusammengesetzt wird. Man kann ihn also auch mit `gEkE` abkürzen.

Anhang J

Definition von Tastaturbelegungen

Im Inhaltsverzeichnis heißt dieses Kapitel „Definition von anwenderspezifischen Zeichensätzen“, was aber ein ganz anderes Thema ist.

Bei der Auflistung der Tasten ist zu beachten, dass die Liste mit *Eins* statt Null beginnt, für Tabellen-Offsets muss man also entsprechend umrechnen.

J.1 Seite J-2

Tabellenplatz 16 ist für die linke **SHIFT**-Taste reserviert.

J.2 Seite J-3

Tabellenplatz	Taste
53	**SHIFT** rechts
59	**CONTROL**
62	**C=**
81	**ALT**
88	**NO SCROLL**

J.3 Seite J-4

Bei Adresse `$fbe4` befindet sich eine weitere Tastaturtabelle, nämlich die vom deutschen 128er-Kernal ungenutzte *CAPS-LOCK*-Tabelle. Die US-Version des 128ers benutzt diese Tabelle, wenn die **ASCII/DIN**-Taste eingerastet ist — die hat dort nämlich die Bedeutung **CAPS LOCK** (ähnlich **SHIFT LOCK**, aber die Ziffern bleiben Ziffern).

Anhang K

Abgeleitete mathematische Funktionen

Dieses Kapitel wurde nur angelegt, damit die Kapitelnummern in diesem Text denen des Originalhandbuchs entsprechen. :)

Anhang L

Steckerbelegungen

L.1 Seite L-1

Die Ansichten der beiden Joyports widersprechen sich: Das obere Pinout zeigt die Belegung des Joyports von außen, die untere ist seitenverkehrt (zeigt also quasi das Ende eines Joystickkabels von außen).

L.2 Seite L-3

Modulanschluss und Videobuchse sind ebenfalls als Außenansicht dargestellt.

L.3 Seite L-4

Auch der Userport wird von außen dargestellt.

Anhang M

Übertragung von BASIC4-Programmen

M.1 Seite M-1

Zeile 191 endet auf `GOTO 195`; das Programm hat aber keine Zeile 195. In Zeile 192 steht ein Doppelkreuz anstelle eines `=`.