MEGA 65

BASIC 65-Referenzhandbuch

MEGA
MUSEUM OF ELECTRONIC GAMES & ART

MEGA65
BASIC 65-REFERENZHANDBUCH

Herausgeber:
MEGA Museum of Electronic Games & Art e.V., Deutschland.

Hinweis zur Übersetzung:
Für Vollständigkeit und Korrektheit der Angaben in diesem übersetzten Handbuch wird keine Haftung übernommen und im Zweifelsfall auf das dem MEGA65 beiliegende englische Original-Handbuch verwiesen.

Übersetzung und Bearbeitung der deutschen Fassung: Günther Reiter *(Snoopy)*

Vielen Dank an die Mitglieder des Forum64 für die tatkräftige Unterstützung!

29. April 2022

Inhaltsverzeichnis

KAPITEL 1

Einleitung

- Willkommen beim MEGA65!

WILLKOMMEN BEIM MEGA65!

Herzlichen Glückwunsch zu Ihrem Kauf eines der lang ersehntesten Computer in der Geschichte der Informatik. Der MEGA65 ist ein von der Community entwickelter Rechner, der auf dem Commodore® 65[1] basiert. Dieser wurde 1989 entwickelt und sollte 1991 auf den Markt kommen. Es wurden damals allerdings nur ein paar wenige Prototypen verkauft. Jahrzehnte später erinnert nun der MEGA65 an diese gute, alte Zeit, als Computer noch einfach gestrickt und freundlich waren. Sie waren nicht nur einfach zu bedienen und in ihrer Funktionsweise zu verstehen, sondern sie boten auch einen freundlichen und leichten Zugang.

Diese Computer aus den 1980er Jahren inspirierten eine ganze Generation von Fachleuten, sich für die spannenden und lohnenden technischen Berufe zu entscheiden, die sie heute ausüben. Stellen Sie sich nur die Freude dieser Menschen vor, als sie erfuhren, dass sie mit ihrem neuen Computer Probleme lösen, einen Brief schreiben, Steuern vorbereiten, neue Dinge erfinden oder sogar herausfinden konnten, wie das Universum funktioniert. Wir wollen diese Begeisterung, die es in der modernen Computerwelt nicht mehr gibt, wieder aufleben lassen und haben deshalb den **MEGA65** entwickelt.

Das MEGA65-Team ist der Meinung, dass der Besitz eines Computers wie der Besitz eines Hauses ist. Ein Haus wird nicht einfach nur genutzt, sondern es werden große und kleine Dinge verändert, um es zu einem eigenen, individuellen Lebensraum zu machen. Nach einer Weile, wenn Sie sich eingelebt haben, entscheiden Sie sich vielleicht, Ihr Haus zu renovieren oder zu erweitern, um es komfortabler zu machen oder mehr Nutzen zu bieten. Betrachten Sie den MEGA65 als ein "Computerhaus".

In diesem Handbuch lernen Sie nicht nur, wie Sie Bilder an die Wand hängen, sondern auch, wie Sie Ihr Traumhaus bauen können. Während Sie dieses Benutzerhandbuch lesen, werden Sie lernen, wie man den MEGA65 bedient, Programme schreibt, zusätzliche Software hinzufügt und die Hardwarefunktionen erweitert. Was nicht sofort auffällt, ist, dass Sie auf dieser Reise, wenn Sie das BASIC 65 und die Befehle des Betriebssystems erkunden, auch etwas über die Geschichte des Computers erfahren.

Mit Computergrafiken und Musik macht das Rechnen mehr Spaß. Und wir haben den MEGA65 entwickelt, um mit ihm Spaß zu haben! In diesem Benutzerhandbuch erfahren Sie, wie Sie mit den integrierten Funktionen des MEGA65 **Grafik** und **Musik** programmieren können. Aber Sie müssen kein Programmierer sein, um Spaß mit dem MEGA65 zu haben. Da der MEGA65 einen kompletten Commodore® 64™[2] enthält, kann er auch Tausende von Spielen, Dienstprogrammen und Business-Softwarepaketen aus der Vergangenheit sowie neue Programme ausführen, die heute von Commodore-Computerenthusiasten geschrieben werden. Die Begeisterung für den MEGA65 wird

[1] Commodore ist eine Marke von C= Holdings

[2] Commodore 64 ist eine Marke von C= Holdings.

in dem Maße wachsen, in dem klar wird, welche Programme durch die Leistung und die Funktionen dieses modernen Commodore-Computers erschaffen werden können. Gemeinsam werden wir eine neue "Homebrew"-Community aufbauen, um Software und Projekte zu entwickeln, von denen wir nicht dachten, dass sie auf dem MEGA65 möglich sind.

Wir heißen Sie auf dieser Reise willkommen! Vielen Dank, dass Sie ein Teil der Gemeinschaft werden, die den MEGA65 begeistert nutzt und programmiert!

KAPITEL 2

BASIC 65-Befehlsreferenz

- Befehle, Funktionen und Operatoren
- BASIC 65-Konstanten
- BASIC 65-Variablen
- BASIC 65-Arrays
- BASIC 65-Operatoren
- BASIC 65-Befehlsreferenz

BEFEHLE, FUNKTIONEN UND OPERATOREN

Dieses Referenzhandbuch beschreibt alle Befehle, Funktionen und andere aufrufbaren Elemente von BASIC 65, einer erweiterten Version von BASIC 10. Einige von ihnen können ein oder mehrere Argumente annehmen, d.h. Eingaben die Sie als Teil des Befehls oder Funktionsaufrufs bereitstellen. Einige erfordern auch die Verwendung spezieller Schlüsselwörter. Hier ist ein Beispiel dafür, wie Befehle, Funktionen und Operatoren in diesem Referenzhandbuch beschrieben werden:

KEY <numerischer Ausdruck>,<String-Ausdruck>

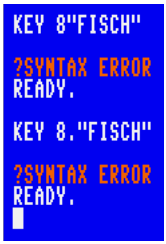
In diesem Fall ist KEY das, was wir als **Schlüsselwort** bezeichnen. Das bedeutet einfach ein spezielles Wort, das BASIC versteht. Schlüsselwörter werden immer in GROSSBUCHSTABEN geschrieben, so dass Sie sie leicht erkennen können.

Die Zeichen **<** und **>** bedeuten, dass alles, was zwischen ihnen steht, vorhanden sein muss, damit der Befehl, die Funktion oder der Operator funktioniert. In diesem Fall bedeutet es, dass wir einen **numerischen Ausdruck** an einer Stelle und einen sogenannten **String-Ausdruck** (eine Zeichenkette wird als "String" bezeichnet) an einer anderen Stelle haben. Was das ist, werden wir gleich näher erläutern.

Sie können manchmal auch eckige Klammern um etwas herum sehen. Zum Beispiel, **[,numerischer Ausdruck]**. Das bedeutet, dass alles, was zwischen den eckigen Klammern steht, optional ist, d. h. Sie können es bei Bedarf einfügen, aber dass der Befehl, die Funktion oder der Operator auch ohne sie funktioniert. Der Befehl **CIRCLE** hat zum Beispiel ein optionales numerisches Argument, das angibt, ob der Kreis beim Zeichnen ausgefüllt werden soll.

Das Komma und einige andere Symbole und Satzzeichen stehen nur für sich selbst. In diesem Fall bedeutet es, dass ein Komma zwischen dem **numerischen Ausdruck** und dem **String-Ausdruck** stehen muss. Das ist die sogenannte Syntax: Wenn Sie etwas auslassen oder das Falsche an die falsche Stelle setzt, nennt man das einen Syntaxfehler. Der Computer zeigt Ihnen einen Syntaxfehler mit der Meldung **?SYNTAX ERROR** an ("Error" heißt auf Deutsch "Fehler").

Es gibt keinen Grund zur Sorge, wenn Sie eine Fehlermeldung vom Computer erhalten. Vielmehr ist es nur die Art und Weise, wie der Computer Ihnen mitteilt, dass etwas nicht in Ordnung ist, so dass Sie das Problem leichter finden und beheben können. Fehlermeldungen wie diese schaden dem Computer nicht und beschädigen auch Ihr Programm nicht, es besteht also kein Grund zur Sorge. Wenn wir zum Beispiel versehentlich ein Komma weglassen oder es durch einen Punkt ersetzen, antwortet der Computer mit SYNTAX ERROR, ähnlich wie unten dargestellt:



Es ist allgemein üblich, dass Befehle, Funktionen und Operatoren einen oder mehrere **"Ausdrücke"** verwenden. Ein Ausdruck ist nur ein schicker Name für etwas, das einen Wert hat. Dabei kann es sich um einen String (Zeichenkette) (**"HALLO"**), eine Zahl (**23.7**) oder eine Berechnung, die eine oder mehrere Funktionen oder Operatoren (**LEN("HALLO") * (3 XOR 7)**) beinhaltet, handeln. Im Allgemeinen können Ausdrücke entweder einen String oder ein numerisches Ergebnis liefern. In diesem Fall nennen wir die Ausdrücke entweder String-Ausdrücke oder numerische Ausdrücke. Zum Beispiel ist **"HALLO"** ein **String-Ausdruck**, während **23.7** ein **numerischer Ausdruck** ist.

Es ist wichtig, dass Sie beim Schreiben Ihrer Programme den richtigen Ausdruckstyp verwenden. Wenn Sie versehentlich den falschen Typ verwenden, gibt Ihnen der Computer eine **?TYPE MISMATCH ERROR**-Fehlermeldung zurück, um Ihnen mitzuteilen, dass der Typ des von Ihnen angegebenen Ausdrucks nicht mit dem erwarteten Typ übereinstimmt. Wir erhalten zum Beispiel eine **?TYPE MISMATCH ERROR**-Fehlermeldung, wenn wir den folgenden Befehl eingeben, denn **KARTOFFEL** ist ein String-Ausdruck und kein numerischer Ausdruck:

Wenn Sie möchten, können Sie versuchen, dies selbst einzutippen.

Befehle sind Anweisungen, die Sie direkt von der Eingabeaufforderung **READY.** oder innerhalb eines Programms verwenden können:

BASIC 65-KONSTANTEN

Typ	Beispiel	Beispiel
Dezimale ganze Zahl (integer)	32000	-55
Dezimale Festkommazahl (fixed point)	3.14	-7654.321
Dezimale Fließkommazahl (floating point)	1.5E03	7.7E-02
Hexadezimale Zahl (hex)	$D020	$FF
String (string)	"X"	"TEXT"

BASIC 65-VARIABLEN

Jede skalare Variable verbraucht 8 Byte Speicherplatz im Speicher. Der reservierte Bereich in Bank 0 von $F700-$FEFF kann 256 Variablen speichern. Variablen müssen nicht deklariert werden und der Typ wird durch ein angehängtes Zeichen bestimmt. Alle Variablen ohne angehängtes Zeichen werden standardmäßig als reelle Zahl (REAL) betrachtet. Der Speicherplatz wird bei ihrer ersten Verwendung reserviert und sie werden mit Null initialisiert, String-Variablen werden als leere Strings initialisiert.

Alle Variablen mit nur einem Buchstaben als Variablenname werden als **schnelle** Variablen deklariert. Benutzerfunktionen mit nur einem Buchstaben als Funktionsname werden als **schnelle** Funktionen deklariert. Dies sind die 104 Variablen (A - Z), (A% - Z%), (A& - Z&) und (A$ - Z$) und die 26 Funktionen (FNA() - FNZ()).

Sie haben feste Speicheradressen im Bereich $FD00 - $FEFF und bieten dadurch einen schnellen Zugriff. Die entsprechende Adresse wird durch einen Hash-Algorithmus aus dem Variablennamen erzeugt und nicht wie bei anderen Variablen in einer Tabelle gesucht.

Typ	Anhang	Bereich	Beispiel
Byte (byte)	&	0 .. 255	BY& = 23
ganze Zahl (integer)	%	-32768 .. 32767	I% = 5
reelle Zahl (real)	keiner	-1E37 .. 1E38	XY = 1/3
String (string)	$	Länge = 0 .. 255	AB$ = "TEXT"

BASIC 65-ARRAYS

Jedes Array verbraucht die Anzahl der Elemente multipliziert mit der Elementgröße plus die Größe der Kopfzeile (header) (6 + 2 * Dimensionen) im Speicher.

Zum Beispiel hat das Array

```
100 DIM X(8,2,3) : REM (0..8, 0..2, 0..3)
```

3 Dimensionen und 9 x 3 x 4 = 108 Elemente.

Die Größe für reelle Zahlen (real) beträgt 5 Bytes, so dass die Daten für dieses Array 540 Bytes beanspruchen. Die Größe der Kopfzeile (header) beträgt 6 + 2 * 3 = 12 Bytes. Die komplette Größe im Speicher beträgt also 552 Bytes.

Arrays werden in Bank 1 ab Adresse $2000 gespeichert und erweitern sich im Speicher nach oben aufsteigend. Sie teilen sich den verfügbaren Speicher ($2000 .. $F6FF) mit dem String-Bereich, der in Bank 1 an der Adresse $F6FF beginnt und sich im Speicher nach unten absteigend erweitert. Jeder der oben genannten einfachen Variablentypen kann auch als Array verwendet werden, indem er mit einer DIM-Anweisung deklariert wird. Die Arrays werden bei der Deklaration für alle Elemente mit Null initialisiert. Wenn ein nicht deklariertes Array-Element verwendet wird, wird eine automatische implizite Deklaration durchgeführt, die die obere Grenze für jede Dimension auf 10 setzt. Zum Beispiel würde bei Verwendung eines nicht deklarierten Elements AB(3,5) automatisch ein "DIM AB(10,10)" durchgeführt. Die untere Grenze für jede Dimension ist immer 0 (Null). Ein mit DIM AB(10) initialisiertes Array besteht also aus 11 Elementen und akzeptiert Indizes von 0 bis 10.

Stringarrays sind präzise ausgedrückt Arrays von Stringbezeichnern. Jedes Element besteht aus drei Bytes, die folgende Werte enthalten: Länge des Strings und die Adresse (niederwertiges/höherwertiges Byte) des zugeordneten Strings im Stringspeicher. Die Verwendung der BASIC-Funktion POINTER (dt.: "Zeiger") mit einem String- oder Stringarray-Element als Argument, gibt die Adresse des Bezeichners zurück, nicht den String selbst.

Typ & Elementgröße		Anhang	Bereich	Beispiel
Byte-Array	1	&	0 .. 255	BY&(5,6) = 23
Ganzzahl-Array (integer)	2	%	-32768 .. 32767	I%(0,10) = 5
Reelles Array (real)	5	keinen	-1E37 .. 1E38	XY(I%) = 1/3
String-Array (string)	3	$	Länge = 0 .. 255	AB$(X) = "TEXT"

BASIC 65-OPERATOREN

BASIC 65 bietet eine Reihe von Operatoren, die für die meisten BASIC-Programmiersprachen typisch sind. Die Verwendung und der Vorrang entsprechen den Standards.

Das Symbol = wird sowohl als Zuweisungsoperator als auch als relationaler Operator zur Prüfung der Gleichheit verwendet. Zum Beispiel in einer Anweisung **`A = B = 5`** ist das erste Gleichheitszeichen der Zuweisungsoperator, während das zweite ein logischer Operator ist, der die Variable B mit 5 vergleicht. Als Ergebnis wird A entweder der Wert **-1** für **TRUE** (wahr) oder der Wert **0** für **FALSE** (falsch). Beachten Sie, dass der Wert von **-1** für **TRUE** anders ist als in anderen Programmiersprachen, wie zum Beispiel **C**, wo der Wert **1** für **TRUE** verwendet wird.

Das Symbol + kann als positives Vorzeichen für numerische Ausdrücke verwendet werden, als Additionsoperator oder für die Verkettung von Zeichenketten. Die Anzahl und der Typ der Operanden bestimmt die Operation.

Das Symbol - kann als negatives Vorzeichen für numerische Ausdrücke oder als Subtraktionsoperator verwendet werden. Die Anzahl und der Typ der Operanden bestimmt die Operation.

Die Operatoren **NOT, AND, OR, XOR** können sowohl als logische Operatoren (Beispiel 1) als auch als boolesche Operatoren (Beispiel 2) verwendet werden.

1. Beispiel: **`IF A>B AND A<0`**
2. Beispiel: **`A = B AND $7F`**

Beide Beispiele liefern intern immer ein ganzzahliges Ergebnis, das entweder numerisch oder logisch interpretiert werden kann. Wenn das Ergebnis eines Vergleichs **TRUE** ist, wird der Wert auf **-1** gesetzt, während ein falsches Ergebnis zu **0** führt. Im zweiten Beispiel wandelt der Operator **AND** beide Operanden in einen 16 Bit-Ganzzahlwert um und führt eine bitweise Verknüpfung **AND** für alle 16 Bits durch. In diesem Beispiel wird der Wert von B genommen, die oberen 9 Bits werden auf Null gesetzt und das Ergebnis in A gespeichert.

Das Ergebnis von logischen Operationen kann auch in numerischen Ausdrücken verwendet werden, zum Beispiel bei **`A = A - (B > 7)`** wird A um 1 erhöht, wenn der Vergleich (B > 7) **TRUE** (-1) ergibt.

Die Operatoren haben verschiedene Prioritäten, die jeweils in einer Tabelle unten aufgeführt sind. Zum Beispiel werden in der Anweisung **`A * A - B * B`** zuerst beide Multiplikationen durchgeführt, bevor die Subtraktion ausgeführt wird. Klammern werden verwendet, um die Rangfolge zu ändern, zum Beispiel führt **`A * (A - B) * B)`** zuerst die Subtraktion aus.

Zuweisungsoperator

Symbol	Beschreibung	Operandentyp	Beispiel
=	Zuweisung	alle	A = 42, A$ ="HALLO", A = B < 42

Unäre mathematische Operatoren

Name	Symbol	Beschreibung	Operandentyp	Beispiel
Plus	+	positives Vorzeichen	numerisch	A = +42
Minus	-	negatives Vorzeichen	numerisch	B = -42

Binäre mathematische Operatoren

Name	Symbol	Beschreibung	Operandentyp	Beispiel
Plus	+	Addition	numerisch	A = B + 42
Minus	-	Subtraktion	numerisch	B = A - 42
Stern	*	Multiplikation	numerisch	C = A * B
Schrägstrich	/	Division	numerisch	D = B / 13
Aufwärtspfeil	↑	Potenzierung	numerisch	E = 2 ↑ 10
Links schieben	<<	Nach links schieben	numerisch	A = B << 2
Rechts schieben	>>	Nach rechts schieben	numerisch	A = B >> 1

Vergleichsoperatoren

Symbol	Beschreibung	Operandentyp	Beispiel
>	größer	numerisch	A > 42
>=	größer oder gleich	numerisch	B >= 42
<	kleiner	numerisch	A < 42
<=	kleiner oder gleich	numerisch	B <= 42
=	gleich	numerisch	A = 42
<>	ungleich	numerisch	B <> 42

Logikoperatoren

Schlüsselwort	Beschreibung	Operandentyp	Beispiel
AND	Und	logisch	A > 42 AND A < 84
OR	Oder	logisch	A > 42 OR A = 0
XOR	Exklusives Oder	logisch	A > 42 XOR B > 42
NOT	Negation	logisch	C = NOT A > B

Boolesche Operatoren

Schlüsselwort	Beschreibung	Operandentyp	Beispiel
AND	Und	boolesch	A = B AND $FF
OR	Oder	boolesch	A = B OR $80
XOR	Exklusives Oder	boolesch	A = B XOR 1
NOT	Negation	boolesch	A = NOT 22

String-Operator

Name	Symbol	Beschreibung	Operandentyp	Beispiel
Plus	+	Verkettung	String	A$ = B$ + ".PRG"

Priorität der Operatoren

Priorität	Operatoren
hoch	↑
	+ - Zeichen
	* /
	+ -
	<< >> (Arithmetische Verschiebungen)
	< <= > >= = <>
	NOT
	AND
niedrig	OR XOR

BASIC 65-BEFEHLSREFERENZ

Symbol	Bedeutung
...	Sie können den Vorgang ... so oft wiederholen, wie Sie wollen, oder auch gar nicht.
[ ]	Alles, was zwischen den eckigen Klammern steht, ist optional und kann auch weggelassen werden.
< | >	Sie müssen eine Option auswählen. Die Optionen sind durch | getrennt.
[|]	Sie können eine Option auswählen. Die Optionen sind durch | getrennt. Sie können auch keine Option wählen, indem Sie sie weglassen.
{ , }	Zwischen { und } stehen eine Reihe von Ausdrücken, die durch Kommata getrennt sind. Die Ausdrücke sind optional und können weggelassen werden, aber Sie müssen mindestens einen Ausdruck enthalten. Die Kommas bleiben links vom letzten Ausdruck den Sie einfügen.
[{ , }]	Dies ist wie { , }, außer dass Sie alle Ausdrücke weglassen können. Wenn Sie das tun, lassen Sie auch alle Kommas weg.

Ablaufkontrolle
BEGIN
BEND
CONT
DEF FN
DO
ELSE
END
EXIT
FGOSUB
FGOTO
FN ()
FOR
GOSUB
GOTO
IF
LOOP
NEXT
ON
REM
RETURN
RREG
RUN
SLEEP
STEP
STOP
SYS
THEN
UNTIL
USR ()
WAIT
WHILE

Programmierung
AUTO [1]
CHANGE [1]
DELETE [1]
EDIT [1]
FIND [1]
HELP
HIGHLIGHT
LIST
NEW
RENUMBER [1]
TROFF
TRON

Math. Funktionen
ABS ()
ATN ()
COS ()
EXP ()
INT ()
LOG ()
LOG10 ()
MOD ()
RND ()
SGN ()
SIN ()
SQR ()
TAN ()

Speicher
BANK
CLR
DIM
DMA
EDMA
FRE ()
LET
PEEK ()
PEEKW ()
POINTER ()
POKE
POKEW

Math. Operatoren		
+	*	↑
-	/	

Umrechnen
ASC ()
CHR$ ()
DEC ()
HEX$ ()
STR$ ()
VAL ()

Daten
DATA
READ
RESTORE

Strings
+
ASC ()
CHR$ ()
INSTR ()
LEFT$ ()
LEN ()
MID$ ()
RIGHT$ ()

Logik-Operatoren [3]	
AND	OR
NOT	XOR

Relationale Operat.	
<	<=
=	<>
>	>=

Fehlerbehandlung
EL [2]
ER [2]
ERR$ ()
RESUME
TRAP

Zeit
DT$ [2]
TI [2]
TI$ [2]

Diskette	Abkürzung
APPEND	
BACKUP	
BLOAD	
BOOT	
BSAVE	
BVERIFY	
CATALOG	$ [1]
COLLECT	
CONCAT	
COPY	
DCLEAR	
DCLOSE	
DELETE	
DIR	$ [1]
DIRECTORY	$ [1]
DISK	@ [1]
DLOAD	
DOPEN	
DS [2]	
DS$ [2]	
DSAVE	
DVERIFY	
ERASE	
HEADER	
LIST	
LOAD	/ [1]
LOADIFF	
MERGE	
RECORD	
RENAME	
RUN	
SAVE	← [1]
SAVEIFF	
SCRATCH	
SET	
TYPE	
VERIFY	

Eingabe
GET
GETKEY
INPUT
JOY ()
LPEN ()
MOUSE
POT ()
RMOUSE

Ein- und Ausgabe
CLOSE
CMD
FREAD
FWRITE
GET#
INPUT#
LINE INPUT#
OPEN
PRINT#
PRINT# USING
ST [2]

System
FAST
GO64
KEY
MONITOR
RSPEED ()
SPEED

Sekundäres
OFF
TO

Grafik
BOX
CHAR
CIRCLE
DMODE
DPAT
ELLIPSE
GRAPHIC CLR
LINE
LOADIFF
PAINT
PALETTE
PEN
PIXEL ()
POLYGON
RGRAPHIC ()
RPALETTE ()
RPEN ()
SAVEIFF
SCNCLR
SCREEN
VIEWPORT

Bildschirm
BACKGROUND
BORDER
COLOR
CURSOR
FONT
FOREGROUND
PALETTE
POS ()
PRINT
PRINT USING
RCURSOR
RCOLOR ()
RPALETTE ()
RWINDOW ()
SCNCLR
SPC ()
TAB ()
WINDOW

Sprites
BUMP ()
COLLISION
MOVSPR
RSPCOLOR ()
RSPPOS ()
RSPRITE ()
SPRCOLOR
SPRITE
SPRSAV

Musik
ENVELOPE
FILTER
PLAY
RPLAY ()
SOUND
TEMPO
VOL

[1] Nur im Direktmodus

[2] Reservierte Variable

[3] Auch boolesche Operatoren

() Funktion

ABS

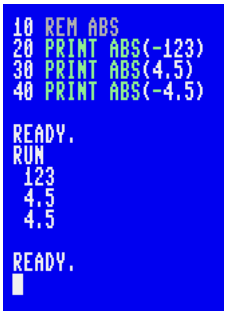
Token: $B6

Format: **ABS(x)**

Zweck: Die numerische Funktion **ABS(x)** liefert den absoluten Betrag des numerischen Arguments **x**.
x ist ein numerisches Argument (ganzzahlig oder reeller Ausduck).

Notiz: Das Ergebnis ist vom Typ real (reelle Zahl).

Beispiel: Verwendung von **ABS**:



AND

Token: $AF

Format: Operand **AND** Operand

Zweck: **AND** führt eine bitweise logische UND-Verknüpfung von zwei 16-Bit-Werten durch. Ganzzahlige Operanden werden so verwendet, wie sie sind. Reelle Operanden werden in eine vorzeichenbehaftete 16-Bit-Ganzzahl umgewandelt (unter Verlust der Genauigkeit). Logische Operanden werden in 16-Bit-Ganzzahlen umgewandelt mit $FFFF, dezimal -1 für TRUE (wahr) und $0000, dezimal 0, für FALSE (falsch).

```
0 AND 0  ->  0
0 AND 1  ->  0
1 AND 0  ->  0
1 AND 1  ->  1
```

Notiz: Das Ergebnis ist vom Typ Ganzzahl (integer). Wenn das Ergebnis in einem logischen Kontext verwendet wird, wird der Wert 0 als FALSE (falsch) betrachtet, und alle anderen Werte unterschiedlich 0 werden als TRUE (wahr) betrachtet.

Beispiel: Verwendung von **AND**:

```
PRINT 1 AND 3
 1

READY.
PRINT 128 AND 64
 0

READY.
```

In den meisten Fällen wird **AND** in **IF**-Anweisungen verwendet.

```
10 REM AND
20 INPUT C
30 IF (C >= 0 AND C < 256) THEN PRINT "WERT DES BYTES"

READY.
```

APPEND

Token: $FE $0E

Format: **APPEND# Kanal, Dateiname [,D Laufwerk (drive)] [,U Gerät (unit)]**

Zweck: Öffnet eine vorhandene sequentielle Datei vom Typ **SEQ** oder **USR** zum Schreiben und positioniert den Schreibzeiger an das Ende der Datei.

Kanal ist eine ganze Zahl, wobei:

- **1 <= Kanal <= 127**
  Die Zeile wird mit einem CR-Zeichen abgeschlossen.
- **128 <= Kanal <= 255**
  Die Zeile wird mit einem CR- und LF-Zeichen abgeschlossen.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **APPEND#** funktioniert ähnlich wie **DOPEN#**, mit der Ausnahme, dass, wenn die Datei bereits existiert, der vorhandene Inhalt der Datei beibehalten wird und alle **PRINT#**-Befehle, die in die geöffnete Datei ausgeführt werden, diese vergrößern.

Beispiel: Verwendung von **APPEND**:

```
APPEND#5,"DATEN",U9
APPEND#130,(DD$),U(UN%)
APPEND#3,"NUTZERDATEI,U"
APPEND#2,"DATENBANK"
```

ASC

Token: $C6

Format: **ASC(String)**

Zweck: Nimmt das erste Zeichen des String-Arguments und gibt dessen numerischen Codewert zurück. Der Name wurde anscheinend gewählt, um eine Eselsbrücke zu ASCII zu bilden, aber der zurückgegebene Wert ist in Wirklichkeit der sogenannte PETSCII-Code.

Notiz: **ASC** liefert bei einem leeren String den Wert 0 zurück. Dieses Verhalten unterscheidet sich zu dem des BASIC 2, bei dem ASC() einen Fehler erzeugt. Die Umkehrfunktion zu **ASC** ist **CHR$**. Weitere Informationen zum **CHR$**-Befehl finden Sie auf Seite 44.

Beispiel: Verwendung von **ASC**:

ATN

Token: $C1

Format: **ATN**(numerischer Ausdruck)

Zweck: Liefert den Arcustangens des Arguments. Das Ergebnis liegt im Bereich $(-\pi/2$ bis $\pi/2)$

Notiz: Eine Multiplikation des Ergebnisses mit $180/\pi$ wandelt den Wert in die Einheit "Grad" um. **ATN** ist die Umkehrfunktion zu **TAN**.

Beispiel: Verwendung von **ATN**:

```
10 REM ATN
20 PRINT ATN(0.5)
30 PRINT ATN(0.5) * 180 / π

READY.
RUN
 0.46364761
 26.565051

READY.
```

AUTO

Token: $DC

Format: **AUTO** [Schrittweite]

Zweck: **AUTO** ermöglicht durch die Vorgabe einer nächsten Zeilennummer ein schnelleres Eintippen von BASIC-Programmen.

Nach Übergabe einer neuen Programmzeile an den BASIC-Editor mit der RETURN-Taste, erzeugt die AUTO-Funktion eine neue BASIC-Zeilennummer für die Eingabe der nächsten Zeile.

Die neue Nummer wird errechnet, indem die **Schrittweite** zur aktuellen Zeilennummer addiert wird.

Wenn Sie **AUTO** ohne ein Argument eingeben, schalten Sie diese Funktion aus.

Beispiel: Verwendung von **AUTO**:

```
AUTO 10  : REM AKTIVIERE AUTO MIT SCHRITTWEITE 10
AUTO     : REM AUTO ABSCHALTEN
```

BACKGROUND

Token: $FE $3B

Format: **BACKGROUND** Farbwert

Zweck: **BACKGROUND** (dt. "Hintergrund") setzt die Hintergrundfarbe des Bildschirms auf das Argument, das im Bereich von 0 bis 255 liegen muss. Alle Farben innerhalb dieses Bereichs können mit dem Befehl **PALETTE** angepasst werden. Beim Start hat der MEGA65 nur die ersten 32 Farben konfiguriert, die in der folgenden Tabelle beschrieben sind.

Beispiel: Verwendung von **BACKGROUND**:

```
BACKGROUND 3 : REM WAEHLE HINTERGRUNDFARBE TUERKIS
```

Auf der folgenden Seite finden Sie eine Tabelle mit den Indizies und den RGB-Werten der standardmäßig eingestellten Farbpalette des MEGA65.

Farben: **Index und RGB-Werte der Farbpalette**

Index	Rot	Grün	Blau	Farbe (engl.)	Farbe (dt.)
0	0	0	0	black	schwarz
1	15	15	15	white	weiß
2	15	0	0	red	rot
3	0	15	15	cyan	türkis
4	15	0	15	purple	violett
5	0	15	0	green	grün
6	0	0	15	blue	blau
7	15	15	0	yellow	gelb
8	15	6	0	orange	orange
9	10	4	0	brown	braun
10	15	7	7	light red	hellrot
11	5	5	5	dark grey	dunkelgrau
12	8	8	8	medium grey	mittelgrau
13	9	15	9	light green	hellgrün
14	9	9	15	light blue	hellblau
15	11	11	11	light grey	hellgrau
16	14	0	0	guru meditation	Klatschmohn
17	15	5	0	rambutan	Abendrot
18	15	11	0	carrot	Sonnenblume
19	14	14	0	lemon tart	Kanariengelb
20	7	15	0	pandan	Frühlingsgrün
21	6	14	6	seasick green	Neues Gras
22	0	14	3	soylent green	Erbsengrün
23	0	15	9	slimer green	Eis
24	0	13	13	the other cyan	Welle
25	0	9	15	sea sky	Wasserfall
26	0	3	15	smurf blue	Gletscherblau
27	0	0	14	screen of death	Abenddämmerung
28	7	0	15	plum sauce	Blauviolett
29	12	0	15	sour grape	Dunkle Orchidee
30	15	0	11	bubblegum	Fuchsia
31	15	3	6	hot tamales	Reifer Apfel

Hinweis: Die englischen Farbbezeichnungen der alternativen Farbpalette sind Eigenkreationen der MEGA65-Entwickler. Die dort verwendeten Anspielungen und Wortwitze sind im Deutschen teilweise nur schwer wiederzugeben. Die deutschen Farbbezeichnungen sind – von der englischen Bezeichnung losgelöst – von einer erfahrenen Raumgestalterin entsprechend ihrer jeweiligen Farbe zugeordnet worden.

BACKUP

Token: $F6

Format: **BACKUP U** Quelle **TO U** Ziel
BACKUP D Quelle **TO D** Ziel [**,U** Gerät (unit)]

Zweck: Die **erste** Form von **BACKUP**, mit der Angabe von Geräten (units) für Quelle und Ziel, kann nur für die Laufwerke verwendet werden, die an den internen FDC (Floppy Disk Controller) angeschlossen sind. Die Geräte 8 und 9 sind für diesen Controller reserviert. Diese können entweder das interne Diskettenlaufwerk (Gerät 8) und ein anderes Diskettenlaufwerk (Gerät 9), das an dasselbe Flachbandkabel angeschlossen ist oder eingebundene D81-Diskettenimages sein.

Daher kann **BACKUP** verwendet werden, um

- von Diskette zu Diskette,
- von Diskette zu Diskettenimage,
- von Diskettenimage zu Diskette und
- von Diskettenimage zu Diskettenimage

zu kopieren, je nachdem, ob ein Diskettenimage eingebunden und ein zweites physisches Diskettenlaufwerk vorhanden ist.

Die **zweite** Form von **BACKUP**, mit der Angabe von Laufwerke für Quelle und Ziel, ist für zwei an den IEC-Bus angeschlossene Laufwerke. Z.B. CBM 4040, 8050, 8250 über IEEE-488-zu-IEC-Adapter. Das Backup wird hier intern von dem Doppel-Diskettenlaufwerk durchgeführt.

Quelle Nummer des Geräts (**U**nit) oder Laufwerks (**D**rive) der Quelldiskette.
Ziel Nummer des Geräts (**U**nit) oder Laufwerks (**D**rive) der Zieldiskette.

Notiz: Die Zieldiskette wird formatiert und eine identische Kopie der Quelldiskette geschrieben. **BACKUP** kann nicht für ein Backup von internen Geräten auf IEC-Geräte oder umgekehrt verwendet werden.

Beispiel: Verwendung von **BACKUP**:

```
BACKUP U8 TO U9      : REM BACKUP INTERNES LAUFWERK 8 NACH LAUFWERK 9
BACKUP U9 TO U8      : REM BACKUP INTERNES LAUFWERK 9 NACH LAUFWERK 8
BACKUP D0 TO D1, U10 : REM BACKUP AUF DOPPEL-LAUFWERK AN IEC-SCHNITTSTELLE
```

BANK

Token: $FE $02

Format: **BANK** Banknummer

Zweck: Wählt die Speicherkonfiguration für BASIC-Befehle, die 16 Bit-Adressen verwenden. Diese sind **LOAD**, **LOADIFF**, **PEEK**, **POKE**, **SAVE**, **SYS**, und **WAIT**. Nähere Informationen dazu finden Sie in der Beschreibung des Systemspeichers.

Notiz: Ein Wert > 127 wählt eine Speicherkonfiguration mit sichtbarem I/O-Bereich. Der Standardwert für die Banknummer ist 128. Diese Konfiguration hat RAM von $0000 bis $1FFF und BASIC ROM, KERNAL ROM und I/O von $2000 bis $FFFF.

Beispiel: Verwendung von **BANK**:

```
BANK 1   : REM WAEHLE SPEICHERKONFIGURATION 1
```

BEGIN

Token: $FE $18

Format: **BEGIN** ... **BEND**

Zweck: **BEGIN** und **BEND** erweitern eine IF-Klausel über mehr als eine Zeile, indem sie einen Codeblock definieren, der von der IF-Anweisung als eine Anweisung betrachtet wird, die nach **THEN** oder **ELSE** ausgeführt wird.

Dadurch wird die einzeilige Beschränkung der Standardanweisung **IF** ... **THEN** ... **ELSE**-Klausel umgangen.

Notiz: Springen Sie nicht mit **GOTO** oder **GOSUB** in eine so zusammengesetzte Anweisung. Dies kann zu unerwarteten Ergebnissen führen.

Das zweite Beispiel zeigt eine Besonderheit in der Implementierung der zusammengesetzten Anweisung. Wenn die Bedingung zu **FALSE** ausgewertet wird, wird die Ausführung nicht wie vorgesehen direkt nach **BEND** fortgesetzt, sondern am Anfang der nächsten Zeile.

Beispiel: Verwendung von **BEGIN** und **BEND**:

```
10 REM BEGIN
20 GET A$
30 IF A$>="A" AND A$<="Z" THEN BEGIN
40 PW$=PW$+A$
50 IF LEN(PW$)<7 THEN 80
60 BEND : REM IGNORIERE ALLES AUSSER (A-Z)
70 IF A$<>CHR$(13) GOTO 20
80 PRINT "PW=";PW$

READY.
RUN
PW=MEGAMEGA

READY.
```

```
10 REM BEND
20 IF Z > 1 THEN BEGIN:A$="EINS"
30 B$="ZWEI"
40 PRINT A$;" ";B$;:BEND:PRINT " QUAK!"
50 REM DIE AUSFUEHRUNG WIRD HIER FUER Z <= 1 FORTGESETZT

READY.
```

BEND

Token: $FE $19

Notiz: **BEND** ist ein Schlüsselwort, das nur in Kombination mit **BEGIN** verwendet wird.

Näheres siehe bei **BEGIN** auf Seite 22.

BIT

Token: $FE $4E

Format: **CLR BIT** Adresse**,** Bit
SET BIT Adresse**,** Bit

Zweck: **BIT** ist ein sekundäres Schlüsselwort und kann nur in Verbindung mit **CLR** oder **SET** verwendet werden.

CLR BIT löscht das angegebene **Bit** in **Adresse**.

SET BIT setzt das angegebene **Bit** in **Adresse**.

Adresse ist die Adresse, in der ein Bit gesetzt oder gelöscht werden soll.

Bit ist die Nummer des Bits, das in der angegebenen Adresse gesetzt oder gelöscht werden soll (0 - 7).

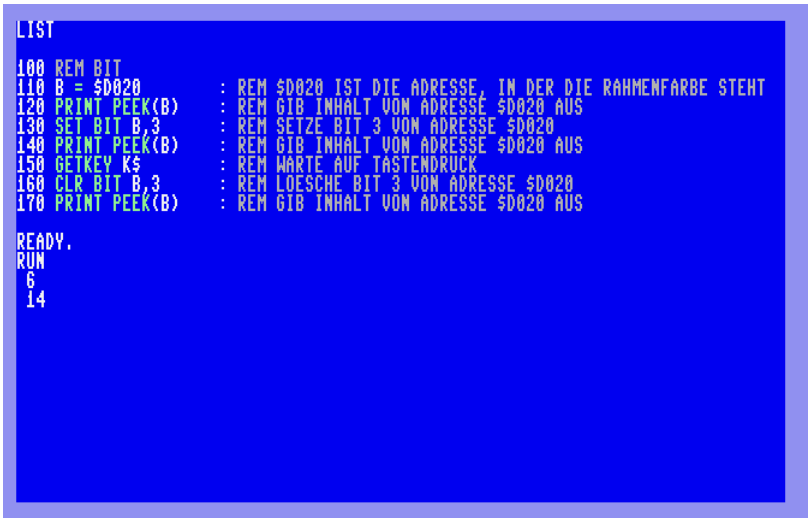
Liegt die Adresse im Bereich von $0000 bis $FFFF (0-65535), wird die mit **BANK** eingestellte Speicherbank verwendet.

Adressen, die größer oder gleich $10000 (dezimal 65536) sind, werden als flache Speicheradressen angenommen und als solche verwendet, ohne dass die Einstellung von **BANK** berücksichtigt wird.

Ein Bankwert > 127 wird für den Zugriff auf Ein- und Ausgaberoutinen die zugrunde liegende Systemhardware wie VIC, SID, FDC usw. verwendet.

Beispiel: Verwendung von **BIT**:

In dem folgenden Beispiel wird das Bit 3 der Adresse $D020, in der die Rahmenfarbe gespeichert ist, zuerst gesetzt und nach einem Tastendruck wieder gelöscht. Der Defaultwert ist 6 (blau). Bit 3 setzen erhöht den Wert auf 14 (= 6 + 2^3). Das Löschen von Bit 3 setzt den Wert wieder auf 6 zurück. Der unten abgebildete Bildschirm zeigt die Rahmenfarbe nach setzen von Bit 3.


LIST
100 REM BIT
110 B = $D020 : REM $D020 IST DIE ADRESSE, IN DER DIE RAHMENFARBE STEHT
120 PRINT PEEK(B) : REM GIB INHALT VON ADRESSE $D020 AUS
130 SET BIT B,3 : REM SETZE BIT 3 VON ADRESSE $D020
140 PRINT PEEK(B) : REM GIB INHALT VON ADRESSE $D020 AUS
150 GETKEY K$: REM WARTE AUF TASTENDRUCK
160 CLR BIT B,3 : REM LOESCHE BIT 3 VON ADRESSE $D020
170 PRINT PEEK(B) : REM GIB INHALT VON ADRESSE $D020 AUS
READY.
RUN
6
14

BLOAD

Token: $FE $11

Format: **BLOAD** Dateiname [**,B** Bank] [**,P** Adresse] [**,R**] [**,D** Laufwerk] [**,U** Gerät]

Zweck: "Binary LOAD" (dt. "binäres Laden") lädt eine Datei des Typs **PRG** in das RAM bei Adresse P.

BLOAD hat zwei Modi: Der sogenannte "flache" Speicheradressenmodus kann verwendet werden, um ein Programm an jede Adresse im 28 Bit (256 MB) Adressbereich, in dem RAM installiert ist, zu laden. Dazu gehören sowohl die Standard-RAM-Bänke 0 bis 5, als auch das sogenannte "Attic RAM" (dt. "Dachboden-RAM") ab der Adresse $8000000.

Dieser Modus wird durch die Angabe einer Adresse, die größer als $FFFF ist, beim Parameter P ausgelöst. Der Bank-Parameter wird in diesem Modus ignoriert.

Aus Kompatibilitätsgründen mit älteren BASIC-Versionen, akzeptiert **BLOAD** auch die Syntax mit einer 16-Bit-Adresse bei P und einer Banknummer bei B. Das "Attic-RAM" ist bei diesem Kompatibilitätsmodus nicht ansprechbar.

Der optionale Parameter **R** (RAW MODE) (dt. "Rohmodus") interpretiert oder verwendet die ersten beiden Bytes der Programmdatei nicht als Ladeadresse, was ansonsten das Standardverhalten ist. Im RAW MODE wird jedes Byte als Daten gelesen.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Bank gibt die zu verwendende RAM-Bank an. Wenn nichts angegeben wird, wird die aktuelle Bank, wie sie mit der letzten **BANK** Anweisung eingestellt wurde, verwendet.

Adresse kann verwendet werden, um die Ladeadresse zu überschreiben, die in den ersten beiden Bytes der **PRG**-Datei gespeichert sind.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable ver-

wendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **BLOAD** kann keine Bank-Grenzen überschreiten. Falls kein Parameter P angeben ist, verwendet **BLOAD** die Ladeadresse der Datei.

Beispiel: Verwendung von **BLOAD**:

```
BLOAD "ML DATEN", B0, U9
BLOAD "SPRITES"
BLOAD "ML ROUTINEN", B1, P32768
BLOAD (FI$), B(BA%), P(PA), U(UN%)
BLOAD "STUECKCHEN",P($8000000) : REM LADE INS ATTIC RAM
```

BOOT

Token: $FE $1B

Format: **BOOT** Dateiname [**,B** Bank] [**,P** Adresse] [**,D** Laufwerk] [**,U** Gerät]
BOOT SYS
BOOT

Zweck: **BOOT Dateiname** lädt eine Datei vom Typ **PRG** in das RAM in Adresse P und Bank B und startet den Code an der Ladeadresse.

BOOT SYS lädt den Bootsektor von Sektor 0, Spur 1 und Gerät 8 an die Adresse $0400 in Bank 0, und führt anschließend einen JSR $0400 durch (Jump To Subroutine / dt. "Sprung in das Unterprogramm") durch.

BOOT ohne Parameter versucht, eine Datei "AUTOBOOT.C65" vom Standardgerät 8 zu laden und auszuführen. Es ist ein Abkürzung für **RUN "AUTOBOOT.C65"**.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Bank gibt die zu verwendende RAM-Bank an. Wenn nichts angegeben wird, wird die aktuelle Bank, wie sie mit der letzten **BANK** Anweisung eingestellt wurde, verwendet.

Adresse kann verwendet werden, um die Ladeadresse zu überschreiben, die in den ersten beiden Bytes der **PRG**-Datei gespeichert sind.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **BOOT SYS** kopiert den Inhalt eines physischen Sektors (zwei logische Sektoren) = 512 Bytes von der Diskette in das RAM in den Speicherbereich von $0400 bis $05ff.

Beispiel: Verwendung von **BOOT**:

```
BOOT SYS
BOOT (FI$), B(BA%), P(PA), U(UN%)
BOOT
```

BORDER

Token: $FE $3C

Format: **BORDER** Farbe

Zweck: Setzt die Rahmenfarbe des Bildschirms auf den angegebenen Farbwert, der im Bereich von 0 bis 255 liegen muss. Alle Farben innerhalb dieses Bereichs können mit dem **PALETTE**-Befehl angepasst werden. Beim Start hat der MEGA65 nur die ersten 32 Farben entsprechend der folgenden Abbildung konfiguriert.

Eine detailliertere Farbtabelle ist unter **BACKGROUND** auf Seite 19 zu finden.

Beispiel: Verwendung von **BORDER**:

```
10 REM BORDER
20 BORDER  4 : REM SETZT DIE RAHMENFARBE AUF VIOLETT

READY.
```

BOX

Token: $E1

Format: **BOX** x0,y0, x2,y2 [, gefüllt]
BOX x0,y0, x1,y1, x2,y2, x3,y3 [, gefüllt]

Zweck: Die erste Form von **BOX**, mit zwei Koordinatenpaaren und einem optionalen Parameter **gefüllt**, zeichnet ein einfaches Rechteck, wobei angenommen wird, dass die Koordinatenpaare zwei diagonal gegenüberliegende Ecken deklarieren.

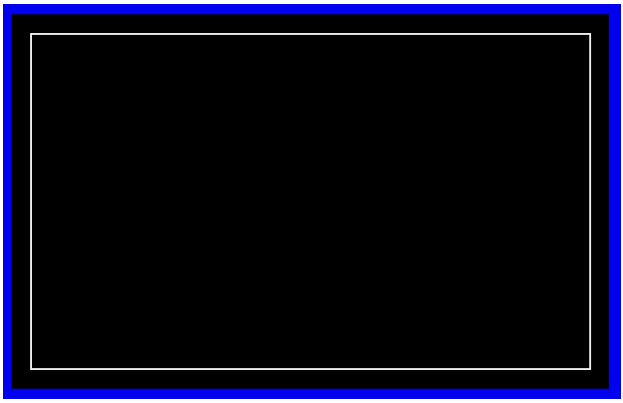
Die zweite Form, mit vier Koordinatenpaaren, deklariert einen Pfad aus vier Punkten, die durch Linien verbunden werden. Der Pfad wird geschlossen, indem der letzte Punkt mit dem ersten Punkt verbunden wird.

Das Viereck wird unter Verwendung des aktuellen Zeichenkontextes gezeichnet, der mit SCREEN, PALETTE und PEN eingestellt wurde. Das Viereck wird gefüllt, wenn der Parameter **gefüllt** ungleich 0 ist.

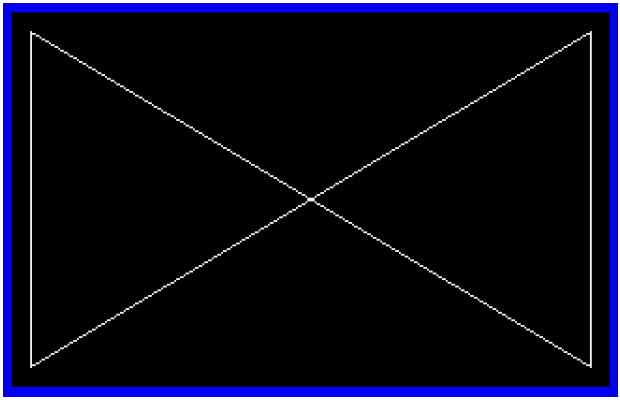
Notiz: **BOX** kann mit vier Koordinatenpaaren verwendet werden, um jede Form zu zeichnen, die mit vier Punkten definiert werden kann, nicht nur Rechtecke. Zum Beispiel Rhomben, Drachen, Trapezoide und Parallelogramme. Es ist auch möglich, Fliegeformen zu zeichnen.

Beispiel: Verwendung von **BOX**:

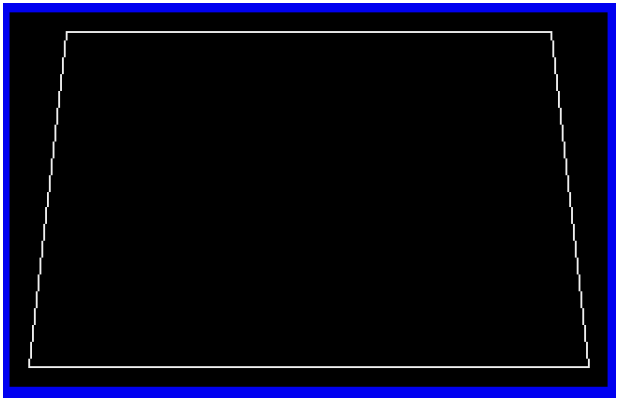
```
10 REM BOX A
20 SCREEN 320,200,2                    : REM OEFFNE GRAFIKBILDSCHIRM
30 BOX 10,10, 309,10, 309,189, 10,189 : REM ZEICHNE RECHTECK
40 GETKEY A$                           : REM WARTE AUF TASTENDRUCK
50 SCREEN CLOSE
```



```
10 REM BOX B
20 SCREEN 320,200,2                    : REM OEFFNE GRAFIKBILDSCHIRM
30 BOX 10,10, 309,189, 309,10, 10,189 : REM ZEICHNE RECHTECK
40 GETKEY A$                           : REM WARTE AUF TASTENDRUCK
50 SCREEN CLOSE
```



```
10 REM BOX C
20 SCREEN 320,200,2                    : REM OEFFNE GRAFIKBILDSCHIRM
30 BOX 30,10, 289,10, 309,189, 10,189 : REM ZEICHNE RECHTECK
40 GETKEY A$                           : REM WARTE AUF TASTENDRUCK
50 SCREEN CLOSE
```



BSAVE

Token: $FE $10

Format: **BSAVE** Dateiname**, P** Startadresse **TO P** Endadresse [**,B** Bank] [**,D** Laufwerk] [**,U** Gerät]

Zweck: "Binary SAVE" (dt. "binäres Speichern") speichert einen Speicherbereich in eine Datei des Typs **PRG**.

BSAVE hat zwei Modi: Der sogenannte "flache" Speicheradressenmodus kann verwendet werden, um einen Speicherbereich im 28 Bit (256 MB) Adressbereich, in dem RAM installiert ist, abzuspeichern. Dazu gehören sowohl die Standard-RAM-Bänke 0 bis 5, als auch das 8 MG große sogenannte "Attic RAM" (dt. "Dachboden-RAM") ab der Adresse $8000000.

Dieser Modus wird durch die Angabe einer Adresse, die größer als $FFFF ist, beim Parameter P ausgelöst. Der Bank-Parameter wird in diesem Modus ignoriert. Dieser "flache" Speicheradressenmodus ermöglicht auch Speicherbereiche von mehr als 64KB Größe abzuspeichern.

Aus Kompatibilitätsgründen mit älteren BASIC-Versionen, akzeptiert **BSAVE** auch die Syntax mit einer 16-Bit-Adresse bei P und einer Banknummer bei B. Das "Attic-RAM" ist bei diesem Kompatibilitätsmodus nicht ansprechbar. Dieser Modus kann keine Bank-Grenzen überschreiten, Start- und Endadresse müssen sich auf dieselbe Bank beziehen.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**. Wenn das erste Zeichen des Dateinamens ein At-Zeichen '@' ist, wird es als "Speichern und Ersetzen"-Vorgang interpretiert. Es wird nicht empfohlen diese Option auf den Laufwerken 1541 und 1571 zu verwenden, da diese einen "Speichern und Ersetzen"-Fehler in ihrem DOS enthalten.

Startadresse ist die erste Adresse, an der die Speicherung beginnt. Sie ist auch die Ladeadresse, die in den ersten beiden Bytes der **PRG**-Datei gespeichert wird.

Endadresse ist die Adresse, an der die Speicherung endet. **Endadresse-1** ist die letzte Adresse, die zum Speichern verwendet wird.

Bank gibt die zu verwendende RAM-Bank an. Wenn nichts angegeben wird, wird die aktuelle Bank, wie sie mit dem letzten **BANK**-Befehl gesetzt wurde, verwendet.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Die Länge der Datei ist **Endadresse - Startadresse + 2**.
Wenn die Zahl nach einem Buchstaben des Arguments keine Dezimalzahl ist, muss sie in Klammern gesetzt werden, wie in der dritten und vierten Zeile der Beispiele.

Das Dateiformat **PRG**, das von **BSAVE** verwendet wird, verlangt, dass die Ladeadresse in die ersten beiden Bytes geschrieben wird. Wenn das Speichern mit einer Banknummer erfolgt, die nicht Null ist oder einer Startadresse höher als $FFFF, passt diese Information nicht. Aus Kompatibilitätsgründen werden nur die beiden niederwertigen Bytes geschrieben. Das Laden dieser Datei mit dem Befehl **BLOAD** erfordert dann die vollständige Angabe der Ladeadresse als Parameter.

Beispiel: Verwendung von **BSAVE**:

```
BSAVE "ML DATEN", P 32768 TO P 33792, B0, U9
BSAVE "SPRITES", P 1536 TO P 2058
BSAVE "ML ROUTINEN", B1, P($9000) TO P($A000)
BSAVE (FI$), B(BA%), P(PA) TO P(PE), U(UN%)
```

BUMP

Token: $CE $03

Format: **BUMP**(Typ)

Zweck: Dient zur Erkennung von Sprite-Sprite (Typ=1)- oder Sprite-Daten (Typ=2)-Kollisionen. Der Rückgabewert ist eine 8-Bit-Maske mit einem Bit pro Sprite. Die Bitposition entspricht der Sprite-Nummer. Jedes gesetzte Bit im Rückgabewert zeigt an, dass das Sprite für seine Position seit dem letzten Aufruf von **BUMP** in eine Kollision verwickelt war. Der Aufruf von **BUMP** setzt die Kollisionsmaske zurück, so dass Sie immer eine Zusammenfassung der Kollisionen, die seit dem letzten Aufruf von **BUMP** erfolgt sind, bekommen.

Notiz: Es ist möglich, mehrere Kollisionen zu erkennen, aber Sie müssen dafür die Sprite-Koordinaten auswerten, um zu erkennen, welche Sprites zusammengestoßen sind.

Beispiel: Verwendung von **BUMP**:

```
10 REM BUMP
20 S% = BUMP(1) : REM SPRITE-SPRITE KOLLISION
30 IF (S% AND 6) = 6 THEN PRINT "SPRITE 1 & 2 KOLLISION"
40 :
50 S% = BUMP(2) : REM SPRITE-DATA KOLLISION
60 IF (S% <> 0) THEN PRINT "EINIGE SPRITES HABEN EINEN DATENBEREICH BERUEHRT"

READY.
```

Sprite	Rückgabewert	Maske
0	1	0000 0001
1	2	0000 0010
2	4	0000 0100
3	8	0000 1000
4	16	0001 0000
5	32	0010 0000
6	64	0100 0000
7	128	1000 0000

BVERIFY

Token: $FE $28

Format: **BVERIFY** Dateiname [**,P** Adresse] [**,B** Bank] [**,D** Laufwerk] [**,U** Gerät]

Zweck: "Binary VERIFY" ("binäres Überprüfen") vergleicht einen Speicherbereich mit einer Datei des Typs **PRG**.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Bank gibt die zu verwendende RAM-Bank an. Wenn nichts angegeben wird, wird die aktuelle Bank, wie sie mit der letzten **BANK** Anweisung eingestellt wurde, verwendet.

Adresse gibt die Startadresse der Überprüfung an. Wenn der Parameter P weggelassen wird, wird die Ladeadresse, die in den ersten beiden Bytes der Datei **PRG** gespeichert ist, verwendet.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **BVERIFY** kann nur auf Gleichheit testen. Es werden keine Informationen über die Anzahl oder die Position der unterschiedlich bewerteten Bytes gemeldet. Im Direktmodus beendet sich **BVERIFY** entweder mit der Meldung **OK** oder mit **VERIFY ERROR** ("Überprüfungsfehler"). Im Programmmodus wird ein **VERIFY ERROR** entweder die Ausführung beenden oder die **TRAP**-Fehlerbehandlungsroutine aufrufen, falls diese aktiv ist.

Beispiel: Verwendung von **BVERIFY**:

```
BVERIFY "ML DATEN", P 32768, B0, U9
BVERIFY "ML ROUTINEN", B1, P(DEC("9000"))
BVERIFY (FI$), B(BA%), P(PA), U(UN%)
BVERIFY "SPRITES", P 1536
```

CATALOG

Token: $FE $0C

Format: **CATALOG** [Dateimuster] [**,W**] [**,R**] [**,D** Laufwerk] [**,U** Gerät]
$ [Dateimuster] [**,W**] [**,R**] [**,D** Laufwerk] [**,U** Gerät]

Zweck: Zeigt einen Dateikatalog/Verzeichnis der angegebenen Diskette bzw. des angebenen Diskettenimage.

Der Parameter **W** (Wide, dt. "breit") listet das Verzeichnis drei Spalten breit auf dem Bildschirm auf und pausiert, wenn der Bildschirm mit einer Seite (63 Verzeichniseinträge) gefüllt ist. Durch Drücken einer beliebigen Taste wird die nächste Seite angezeigt.

Der Parameter **R** (Recoverable, dt. "Wiederherstellbar") umfasst Dateien im Verzeichnis, die als gelöscht gekennzeichnet sind, aber noch wiederherstellbar sind.

Dateimuster ist entweder ein String in Anführungszeichen, zum Beispiel: **"da*"** oder ein Stringausdruck in Klammern, z. B. **(DI$)**

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **CATALOG** ist ein Synonym für **DIRECTORY** oder **DIR** und erzeugt die gleiche Liste. **Dateimuster** kann zum Filtern der Liste verwendet werden. Die Platzhalterzeichen * und **?** können hierfür verwendet werden. Hinzufügen von **,T=** zum Dateimuster, wobei **T** Folgendes angibt einen Dateityp von **P**, **S**, **U** oder **R** (für **P**RG, **S**EQ, **U**SR, **R**EL), filtert die Ausgabe auf diesen Dateityp.

Das Abkürzungssymbol **$** kann nur im Direktmodus verwendet werden.

Beispiel: Verwendung von **CATALOG**:

```
CATALOG
   0 "C65NOTEPAD V1.01"     1D
  50 "C65NOTEPAD"           PRG
   3 "README"               PRG
  33 "NPAGES"               PRG
   8 "BPAGE1"               PRG
   8 "BPAGE2"               PRG
   8 "BPAGE3"               PRG
   8 "BPAGE4"               PRG
   8 "BTEXT1PETSCII"        SEQ
   8 "BTEXT1UTF8"           SEQ
3026 BLOCKS FREE

READY.

CATALOG "*,T=S"
   0 "C65NOTEPAD V1.01"     1D
   8 "BTEXT1PETSCII"        SEQ
   8 "BTEXT1UTF8"           SEQ
3026 BLOCKS FREE

READY.
```

Nachfolgend ein Beispiel, das zeigt, wie ein Verzeichnis mit dem Parameter **W**ide (dt. "breit") aussieht:

```
CATALOG W

   0 "BASIC EXAMPLES  "
   2 "AUTOBOOT.C65"      P     1 "DO"              P     2 "GOTO"            P
   1 "BEGIN"             P     3 "ELLIPSE"         P     2 "GRAPHIC"         P
   1 "BEND"              P     1 "ELSE"            P     1 "HELP"            P
   2 "BOX"               P     1 "EL"              P     1 "IF"              P
   1 "BUMP"              P     1 "ENVELOPE"        P     2 "INPUT#"          P
   1 "CBM-PEOPLE"        S     1 "ERR$"            P     2 "INPUT"           P
   2 "CHAR"              P     1 "ER"              P     2 "JOY"             P
   1 "CHR$"              P     2 "EXIT"            P     1 "LINE INPUT#"     P
   4 "CIRCLE"            P     1 "FGOSUB"          P     1 "LINE"            P
   1 "CLOSE"             P     1 "FGOTO"           P     1 "LOADIFF"         P
   1 "CLR"               P     2 "FILTER"          P     1 "LOOP"            P
   2 "COLLISION"         P     1 "FN"              P     1 "MID$"            P
   1 "CONT"              P     1 "FOR"             P     1 "MOD"             P
   1 "CURSOR"            P     1 "FREAD"           P     1 "MOVSPR"          P
   2 "CUT"               P     1 "FRE"             P     1 "NEXT"            P
  21 "DATA BASE"         R     1 "FWRITE"          P     2 "ON"              P
   1 "DATA"              P     2 "GCOPY"           P     3 "PAINT"           P
   1 "DEF FN"            P     2 "GET#"            P     3 "PALETTE"         P
   3 "DEMOSPRITES1"      P     1 "GETKEY"          P     1 "PEEK"            P
   1 "DIM"               P     1 "GET"             P     3 "PEN"             P
   6 "DMODE"             P     2 "GOSUB"           P     1 "PIXEL"           P
```

CHANGE

Token: $FE $2C

Format: **CHANGE /**Suchstring**/ TO /**Ersetzungsstring**/** [**,** Zeilenbereich]
CHANGE "Suchstring**" TO "**Ersetzungsstring**"** [**,** Zeilenbereich]

Zweck: **CHANGE** führt ein **Suchen und Ersetzen** in dem BASIC-Programm, das sich derzeit im Speicher befindet, durch. Ein optionaler **Zeilenbereich** begrenzt die Suche auf diesen Bereich, ansonsten wird das gesamte BASIC-Programm durchsucht. Bei jedem Auftreten des **Suchstrings** wird die Zeile aufgelistet und der Benutzer wird zu einer Aktion aufgefordert:

- Y RETURN führt die Ersetzung durch und findet den nächsten String
- N RETURN führt die Ersetzung **nicht** durch und findet den nächsten String
- * RETURN ersetzt den aktuellen und alle weiteren Suchtreffer
- RETURN beendet den Befehl und ersetzt **nicht** den aktuellen Treffer

Notiz: Jedes Zeichen, das ohne die Shift-Taste eingegeben werden kann und nicht Teil des Strings ist, kann anstelle von **/** verwendet werden. Durch die Verwendung des Anführungszeichens werden jedoch Textzeichenfolgen gefunden, die nicht tokenisiert und daher nicht Teil eines Schlüsselworts sind.
Zum Beispiel findet `CHANGE "LOOP"TO "OOPS"` nicht das BASIC-Schlüsselwort **LOOP**, da das Schlüsselwort als Token und nicht als Text gespeichert ist. Jedoch findet und ersetzt es `CHANGE /LOOP/ TO /OOPS/` (möglicherweise verursacht dies Sytnaxfehler). Dies kann nur im Direktmodus verwendet werden.

Beispiel: Verwendung von **CHANGE**:

```
CHANGE "XX$" TO "UU$", 2000-2700
CHANGE /IN/ TO /OUT/
CHANGE &IN& TO &OUT&
```

CHAR

Token: $E0

Format: **CHAR** Spalte**,** Zeile**,** Höhe**,** Weite**,** Richtung**,** String [**,** Adresse des Zeichensatzes]

Zweck: Zeigt einen Text auf einem Grafikscreen. **CHAR** kann in allen Auflösungen verwendet werden.

Spalte (in Einheiten von Zeichenpositionen) ist die Startposition der Ausgabe in horizontaler Richtung. Da jede Spalteneinheit 8 Pixel breit ist, hat eine Bildschirmbreite von 320 einen Spaltenbereich von 0-39, während eine Bildschirmbreite von 640 einen Spaltenbereich von 0-79 hat.

Zeile (in Pixel-Einheiten) ist die Startposition der Ausgabe in vertikaler Richtung. Im Gegensatz zum Parameter Spalte ist seine Einheit in Pixeln (nicht in Zeichenpositionen), wobei die oberste Zeile den Wert 0 hat.

Höhe ist ein Faktor, der auf die vertikale Größe der Zeichen angewandt wird, wobei 1 für normale Größe (8 Pixel), 2 für die doppelte Größe (16 Pixel), usw.

Weite ist ein Faktor, der auf die horizontale Größe der Zeichen angewandt wird, wobei 1 für normale Größe (8 Pixel), 2 für die doppelte Größe (16 Pixel), usw.

Richtung gibt die Ausgaberichtung an:
1: nach oben
2: nach rechts
4: nach unten
8: nach links

Die optionale **Adresse des Zeichensatzes** kann verwendet werden um einen Zeichensatz auszuwählen, der sich vom Standardzeichensatz bei Adresse $29800, der Groß- und Kleinschreibung umfasst, unterscheidet.

Es stehen drei Zeichensätze (siehe auch **FONT**) zur Verfügung:

$29000 Font A (ASCII)
$3D000 Font B (Breit)
$2D000 Font C (CBM)

Der erste Teil des Zeichensatzes (Großbuchstaben/Grafik) ist bei $xx000 - $xx7FF gespeichert.

Der zweite Teil der Zeichensatzes (Klein-/Großbuchstaben) ist bei $xx800 - $xxFFF gespeichert.

String ist eine String-Konstante oder ein String-Ausdruck der ausgegeben werden soll. Dieser String kann optional eines oder mehrere der folgenden Steuerzeichen enthalten:

Ausdruck	Tastenkombination	Beschreibung
`CHR$(2)`	CTRL+B	Leerzeichen
`CHR$(6)`	CTRL+F	Reihenfolge umdrehen
`CHR$(9)`	CTRL+I	Verknüpfe mittels AND
`CHR$(15)`	CTRL+O	Verknüpfe mittels OR
`CHR$(24)`	CTRL+X	Verknüpfe mittels XOR
`CHR$(18)`	RVSON	Inverse Darstellung an
`CHR$(146)`	RVSOFF	Inverse Darstellung aus
`CHR$(147)`	CLR	Lösche Bildschirm/Ausschnitt
`CHR$(21)`	CTRL+U	Unterstreichen
`CHR$(25)+"-"`	CTRL+Y + "-"	Nach links drehen
`CHR$(25)+"+"`	CTRL+Y + "+"	Nach rechts drehen
`CHR$(26)`	CTRL+Z	Spiegeln
`CHR$(157)`	Cursor links	Bewege nach links
`CHR$(29)`	Cursor rechts	Bewege nach rechts
`CHR$(145)`	Cursor hoch	Bewege nach oben
`CHR$(17)`	Cursor runter	Bewege nach unten

Notiz: Beachten Sie, dass die Startposition des Strings unterschiedliche Einheiten in horizontaler und vertikaler Richtung hat. Horizontal ist in Spalten und vertikal in Pixeln angegeben.

Weitere Informationen finden Sie unter dem Befehl **CHR$** auf Seite 44.

Beispiel: Verwendung von **CHAR**:

```
10 REM CHAR
20 SCREEN 640,400,2
30 CHAR 28,180,4,4,2,"MEGA65",$29000
40 GETKEY A$
50 SCREEN CLOSE

READY.
```

Gibt den Text "MEGA65" in der Mitte eines 640 x 400 Pixel großen Bildschirms aus.

CHDIR

Token: $FE $4B

Format: **CHDIR** Verzeichnisname [**,U** Gerät]

Zweck: Wechselt in ein Unterverzeichnis oder in ein übergeordnetes Verzeichnis.

Verzeichnisname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Abhängig vom **Gerät** wird **CHDIR** auf verschiedene Dateisysteme angewendet.

Gerät 12 (U12) ist für die SD-Card (FAT-Dateisystem) reserviert. Dort kann dieser Befehl verwendet werden, um in Unterverzeichnisse zu navigieren und Diskettenimages einzubinden (zu mounten), die dort gespeichert sind. **`CHDIR "..",U12`** wechselt in das übergeordnete Verzeichnis auf **Gerät 12**.

Für Geräte, die von CBDOS verwaltet werden (typischerweise 8 und 9), wird **CHDIR** verwendet, um in oder aus Unterverzeichnissen auf Disketten oder Diskettenimages vom Typ **D81** zu wechseln.

Vorhandene Unterverzeichnisse werden als Dateityp **CBM** im übergeordneten Verzeichnis angezeigt, sie werden mit dem Befehl **MKDIR** erstellt.

CHDIR "/",U Gerät wechselt in das Stammverzeichnis.

Beispiel: Verwendung von **CHDIR**:

```
CHDIR "ANWENDUNGEN" : REM WECHSLE INS UNTERVERZEICHNIS "ANWENDUNGEN"
CHDIR ".."          : REM ZURUECK ZUM UEBERGEORDNETEN VERZEICHNIS
CHDIR "SPIELE"      : REM WECHSLE INS UNTERVERZEICHNIS "SPIELE"
```

CHARDEF

Token: $E0 $96

Format: **CHARDEF** Index, Bit-Matrix

Zweck: **CHARDEF** verändert die Bit-Matrix von Zeichen

Index ist die Zeichennummer im Anzeigecode, (@:0, A:1, B:2, ...)

Bit-Matrix ist ein Satz von 8 Byte-Werten, die die Rasterdarstellung für das Zeichen von der obersten Zeile bis zur untersten Zeile definieren. Wenn mehr als 8 Werte als Argumente angegeben werden, werden die Werte 9-16 für das Zeichen (Index+1), 17-24 für (Index+2), usw. verwendet.

Notiz: Die Änderungen der Zeichen werden auf den VIC-Zeichengenerator angewendet, der sich im RAM an der Adresse $FF7E000 befindet.

Alle Änderungen sind flüchtig und der Standard-VIC-Zeichensatz kann durch einen Reset oder durch den Befehl **FONT** wiederhergestellt werden.

Beispiele: Verwendung von **CHARDEF**:

```
10 REM CHARDEF
20 CHARDEF 1,$FF,$81,$81,$81,$81,$81,$81,$FF :REM AENDERE 'A' ZUM RECHTECK
30 CHARDEF 9,$18,$18,$18,$18,$18,$18,$18,$00 :REM AENDERE 'I' ZU SANS SERIF

READY.
RUN
```

```
10 REM CH□RDEF
20 CH□RDEF 1,$FF,$81,$81,$81,$81,$81,$81,$FF :REM □ENDERE 'A' ZUM RECHTECK
30 CH□RDEF 9,$18,$18,$18,$18,$18,$18,$18,$00 :REM □ENDERE 'I' ZU S□NS SERIF

RE□DY.
RUN

RE□DY.
```

CHR$

Token: $C1

Format: **CHR$(**numerischer Ausdruck**)**

Zweck: Gibt einen String zurück, der ein Zeichen enthält, dessen PETSCII-Code gleich dem Argument ist.

Notiz: Der Argumentbereich reicht von 0 bis 255, so dass diese Funktion auch zum Einfügen von Steuercodes in Strings verwendet werden kann. Sogar das NULL-Zeichen mit dem Code 0 ist erlaubt.
CHR$ ist die Umkehrfunktion zu **ASC**. Die vollständige Tabelle der Zeichen (und ihrer PETSCII-Codes) finden Sie auf der Seite 291.

Beispiel: Verwendung von **CHR$**:

```
10 REM CHR$
20 QUOTE$   = CHR$(34)
30 ESCAPE$  = CHR$(27)
40 PRINT QUOTE$;"MEGA65";QUOTE$ : REM PRINT "MEGA65"
50 PRINT ESCAPE$;"Q";       : REM BIS ZUM ENDE DER ZEILE LOESCHEN

READY.
RUN
"MEGA65"

READY.
```

CIRCLE

Token: $E2

Format: **CIRCLE** x-Mittelpunkt**,** y-Mittelpunkt**,** Radius [**,** Status, Start, Stopp]

Zweck: Zeichnet einen Kreis mit dem Mittelpunkt (x,y) und dem Radius (Radius). Ein Sonderfall von **ELLIPSE**, bei dem für den horizontalen und vertikalen Radius derselbe Wert verwendet wird.

x-Zentrum x-Koordinate des Mittelpunkts in Pixel

y-Zentrum y-Koordinate des Mittelpunkts in Pixel

Radius Radius des Kreises in Pixel

Status steuert Füllung, Bögen und die Position des 0 Grad-Winkels. Die Voreinstellung (Null) bedeutet, dass nicht gefüllt wird, die Kreisschenkel gezeichnet werden und der 0 Grad-Winkel auf die 3 Uhr-Position zeigt.

Bit	Name	Wert	Aktion, falls gesetzt
0	Füllen	1	Füllt den Kreis oder Kreisbogen mit der aktuellen Zeichenfarbe.
1	Schenkel	2	Unterdrückt das Zeichnen der Schenkel bei einem Kreisbogen.
2	Oben	4	Der 0 Grad-Winkel soll auf die 12 Uhr-Position (oben) zeigen.

Die Einheiten für den Start- und Stopp-Winkel sind Grad im Bereich von 0 bis 360. Der 0 Grad-Winkel beginnt bei der 3 Uhr-Position und bewegt sich im Uhrzeigersinn. Das Setzen von Bit 2 des Status (Wert 4) verschiebt den 0 Grad-Winkel auf die 12-Uhr-Position.

Start Startwinkel zum Zeichnen eines Kreisbogens.

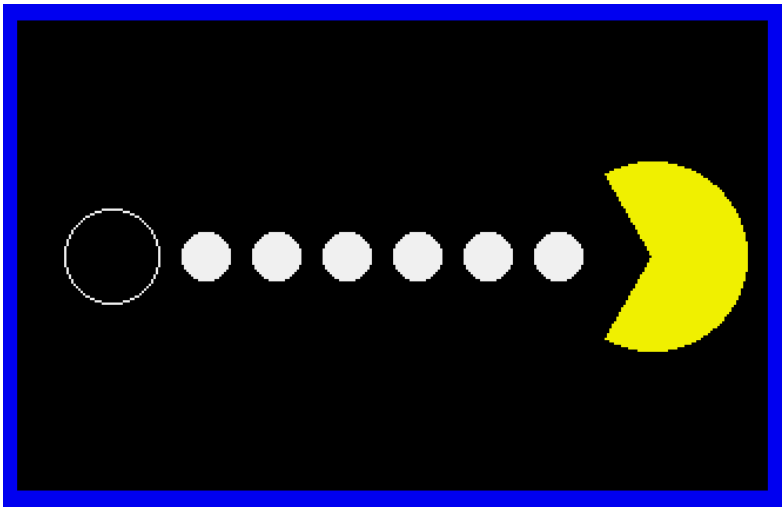
Stopp Endewinkel zum Zeichnen eines Kreisbogens.

Notiz: **CIRCLE** wird verwendet, um Kreise auf Bildschirmen mit einem Seitenverhältnis von 1:1 (z. B.: 320 x 200 oder 640 x 400) zu zeichnen. Bei der Verwendung anderer Auflösungen (z. B. 640 x 200) wird die Form stattdessen eine Ellipse sein.

Beispiel: Verwendung von **CIRCLE**:

```
100 REM CIRCLE
105 :
110 SCREEN 320,200,4            : REM OEFFNE GRAFIKSCHIRM
120 CIRCLE 40,100,20            : REM UNGEFUELLTER KREIS 40,100 MIT RADIUS 20
130 FOR X= 80 TO 250 STEP 30
140  CIRCLE X,100,10,1          : REM GEFUELLTE KREISE MIT RADIUS 10
150 NEXT X
155 :
160 PEN 7                       : REM ZEICHENFARBE 7 (GELB)
170 CIRCLE 270,100,40,1,240,120 : REM GEFUELLTERE KREISBOGEN 240 BIS 120 GRAD
175 :
180 GETKEY K$                   : REM WARTE AUF TASTENDRUCK
190 SCREEN CLOSE                : REM SCHLIESSE GRAFIKSCHIRM
200 END

READY.
```



CLOSE

Token: $A0

Format: **CLOSE** Kanal

Zweck: Schließt einen Eingangs- oder Ausgangskanal.
Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** vergeben worden ist.

Notiz: Das Schließen von Dateien, die zuvor geöffnet wurden, bevor ein Programm beendet ist, ist sehr wichtig, insbesondere für Ausgabedateien. **CLOSE** leert die Ausgabepuffer und aktualisiert die Verzeichnisinformationen auf der Diskette. Wird **CLOSE** nicht ausgeführt, können Dateien und Disketten beschädigt werden. BASIC schließt **nicht** automatisch Kanäle oder Dateien, wenn ein Programm beendet wird.

Beispiel: Verwendung von **CLOSE**:

```
10 REM CLOSE
20 OPEN 2,8,2,"TEST,S,W"
30 PRINT#2,"TESTSTRING"
40 CLOSE 2 : REM DAS WEGLASSEN VON CLOSE WUERDE EINE "SPLAT"-DATEI ERZEUGEN

READY.
```

CLR

Token: $9C

Format: **CLR**
CLR Variable
CLR BIT Adresse**,** Bit

Zweck: **CLR** ohne Parameter dient der Verwaltung von BASIC-Variablen, -Arrays und -Strings. Die Laufzeit-Stackpointer und die Tabelle der offenen Kanäle werden dabei zurückgesetzt. Nach der Ausführung eines **CLR**-Befehls sind alle Variablen und Arrays nicht deklariert. **RUN** führt jeweils **CLR** automatisch aus.

CLR Variable löscht die angegebene Variable (setzt sie auf Null). **Variable** kann sowohl eine numerische Variable als auch eine String-Variable sein, allerdings kein Array.

CLR BIT löscht das angegebene Bit in der angegebenen Adresse (siehe dazu **BIT** auf Seite 24).

Notiz: **CLR** ohne Parameter sollte nicht innerhalb von Schleifen oder Unterprogrammen verwendet werden, da hierbei die Rücksprungadresse zerstört wird. Nach einem **CLR**-Befehl ohne Parameter sind alle Variablen unbekannt und werden beim nächsten Gebrauch initialisiert.

Beispiel: Verwendung von **CLR**:

```
10 REM CLR
20 A=5: P$="MEGA65"
30 CLR
40 PRINT A;P$

READY.
RUN
 0

READY.
```

CMD

Token: $9D

Format: **CMD** Kanal [**,** String]

Zweck: **CMD** (Change Main Device, dt. etwa "ändere Standardgerät") leitet die Standardausgabe vom Bildschirm in den angegebenen Kanal um. Damit können Sie Programmlistings und Dateiverzeichnisse auf andere Ausgabekanäle auszugeben. Es ist auch möglich, diese Ausgabe in eine Datei oder ein Modem umzuleiten.

Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** vergeben worden ist.

Der optionale **String** wird, bevor die Umleitung beginnt, an den Kanal gesendet und kann zum Beispiel für Escape-Sequenzen zur Einrichtung von Druckern oder Modems verwendet werden.

Notiz: Der **CMD**-Modus wird durch einen **PRINT#**-Befehl oder Schließen eines Kanals mit **CLOSE** gestoppt. Es wird empfohlen, vor dem Schließen ein **PRINT#** zu verwenden, um sicherzustellen, dass der Ausgabepuffer auch vollständig geleert wurde.

Beispiel: Verwendung von **CMD**:

Ein Programmlisting auf Drucker ausdrucken:

```
OPEN 1,4 : REM OEFFNE KANAL #1 ZUM DRUCKER AN GERAET 4
CMD 1
LIST
PRINT#1
CLOSE 1
```

COLLECT

Token: $F3

Format: **COLLECT** [**,D** Laufwerk] [**,U** Gerät]

Zweck: Stellt die **BAM** (Block Availability Map, dt. "bla") eines Datenträgers wieder her und löscht dabei sogenannte "Splat"-Dateien (Dateien, die geöffnet, aber nicht richtig geschlossen wurden) und markiert unbenutzte Blöcke als frei.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Dieser Befehl ist zwar nützlich, um eine Diskette von "splat"-Dateien zu bereinigen, er ist aber gefährlich für Disketten mit Bootblöcken oder Dateien mit wahlfreiem Zugriff. Diese Blöcke sind nicht mit normalen Dateien auf der Diskette verbunden und werden daher als frei markiert und können durch weitere Schreibvorgänge überschrieben werden.

Beispiel: Verwendung von **COLLECT**:

```
COLLECT
COLLECT U9
COLLECT D0, U9
```

COLLISION

Token: $FE $17

Format: **COLLISION** Typ [**,** Zeilennummer]

Zweck: Aktiviert oder deaktiviert einen benutzerprogrammierten Interrupt-Handler. Ein Aufruf ohne Zeilennummer schaltet den Handler aus, während ein Aufruf mit Zeilennummer ihn aktiviert. Nach der Ausführung von **COLLISION** mit Zeilennummer, unterbricht eine Sprite-Kollision des im Aufruf angegebenen Typs das BASIC-Programm und führt einen **GOSUB**-Sprung zur Zeile **Zeilennummer** aus, ab der Anwendercode für die Behandlung von Sprite-Kollisionen erwartet wird. Dieser Handler muss die Kontrolle mit einem **RETURN** zurückgeben.

Typ spezifiziert den Kollisionstyp für diesen Interrupt-Handler:

Typ	Beschreibung
1	Sprite - Sprite-Kollision
2	Sprite - Daten-Kollision
3	Lichtgriffel

Zeilennummer muss auf eine Unterroutine verweisen, die Code zur Behandlung von Sprite-Kollisionen enthält und mit einem **RETURN** endet.

Notiz: Es ist möglich, den Interrupt-Handler für alle Typen zu aktivieren, aber es kann immer nur einer ausgeführt werden. Ein Interrupt-Handler kann nicht durch einen anderen Interrupt-Handler unterbrochen werden. Funktionen wie **BUMP**, **RSPPOS** und **LPEN** können zur Auswertung der beteiligten Sprites und deren Positionen verwendet werden.

Beispiel: Verwendung von **COLLISION**:

```
10 REM COLLISION
20 COLLISION 1,70 : REM AKTIVIERE
30 SPRITE 1,1 : MOVSPR 1,120,  0 : MOVSPR 1,0#5
40 SPRITE 2,1 : MOVSPR 2,120,100 : MOVSPR 2,180#5
50 FOR I=1 TO 50000:NEXT
60 COLLISION 1 : REM DEAKTIVIERE
70 END
80 REM SPRITE <-> SPRITE INTERRUPT-HANDLER
90 PRINT "BUMP RETURNS";BUMP(1)
100 RETURN: REM KEHRE VOM INTERRUPT ZURUECK

READY.
```

Info: **COLLISION** wurde in BASIC 10 nicht fertiggestellt. Eine funktionierende Implementierung wird in einem zukünftigen BASIC 65-Update verfügbar sein.

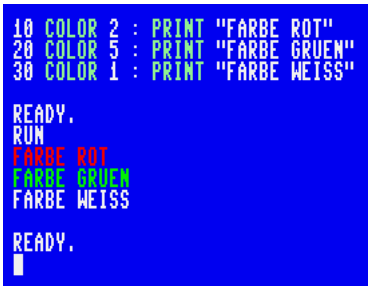
COLOR

Token: $E7

Format: **COLOR** Farbindex

Zweck: Der Befehl funktioniert auf die gleiche Weise wie **FOREGROUND**, d.h.: er setzt die Vordergrundfarbe (Textfarbe) des Bildschirms auf das Argument **Farbindex**, das im Bereich von 0 bis 31 liegen muss. Siehe dazu die Tabelle unter **BACKGROUND** auf Seite 19 für die Farbwerte und ihre entsprechenden Farben.

Beispiel: Verwendung von **COLOR**:



CONCAT

Token: $FE $13

Format: **CONCAT** Anhangsdatei [**,D** Laufwerk] **TO** Zieldatei [**,D** Laufwerk] [**,U** Gerät]

Zweck: **CONCAT** (concatenation) (dt. "Verkettung") fügt den Inhalt von **Anhangsdatei** an **Zieldatei** an. Danach enthält **Zieldatei** den Inhalt der beiden Dateien, während **Anhangsdatei** unverändert bleibt.

Anhangsdatei ist entweder ein String in Anführungszeichen (zum Beispiel: **"Daten"**) oder ein String-Ausdruck in Klammern (zum Beispiel: **(FI$)**).

Zieldatei ist entweder ein String in Anführungszeichen (zum Beispiel: **"Sicherung"**) oder ein String-Ausdruck in Klammern (zum Beispiel: **(FI$)**).

Wenn das Diskettengerät über zwei Laufwerke verfügt, ist es möglich, CONCAT auf Dateien anzuwenden, die auf verschiedenen Laufwerken gespeichert sind. In diesem Fall ist es erforderlich, die Laufwerksnummer für beide Dateien anzugeben. Dies ist auch dann erforderlich, wenn beide Dateien auf Laufwerknummer 1 gespeichert sind.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **CONCAT** wird im DOS des Laufwerks ausgeführt. Beide Dateien müssen existieren und es ist kein Mustervergleich erlaubt. Es dürfen nur Dateien des Typs **SEQ** verkettet werden.

Beispiel: Verwendung von **CONCAT**:

```
CONCAT "NEUE DATEN" TO "ARCHIV" ,U9
CONCAT "ADRESSE",D0 TO "ADRESSBUCH",D1
```

CONT

Token: $9A

Format: **CONT**

Zweck: Wiederaufnahme der Programmausführung nach einer Unterbrechung oder einem Stopp, der durch eine **END**- oder **STOP**-Anweisung oder durch Drücken der RUN STOP-Taste ausgelöst worden ist. Dies ist ein nützliches Werkzeug zur Fehlersuche. Das BASIC-Programm kann angehalten werden und die Variablen können untersucht und sogar geändert werden. Mit der Anweisung **CONT** wird die Ausführung dann wieder aufgenommen.

Notiz: **CONT** kann nicht verwendet werden, wenn ein Programm wegen eines Fehlers gestoppt wurde. Auch jede Bearbeitung eines Programms verhindert die Fortsetzung des Programms. Das Anhalten und Fortsetzen kann die Bildschirmausgabe beeinträchtigen und auch mit Ein- und Ausgabeoperationen in Konflikt geraten.

Beispiel: Verwendung von **CONT**:

COPY

Token: $F4

Format: **COPY** Quelle [**,D** Laufwerk] [**,U** Gerät] **TO** [Ziel] [**,D** Laufwerk] [**,U** Gerät]

Zweck: Kopiert den Inhalt von **Quelle** nach **Ziel**. Wird verwendet, um entweder einzelne Dateien oder, unter Verwendung von Platzhalterzeichen, mehrere Dateien zu kopieren.

Quelle ist entweder ein String in Anführungszeichen (zum Beispiel: **"Daten"**) oder ein String-Ausdruck in Klammern (zum Beispiel: **(FI$)**).

Ziel ist entweder ein String in Anführungszeichen (zum Beispiel: **"Kopie"**) oder ein String-Ausdruck in Klammern (zum Beispiel: **(FI$)**).

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Wenn keine oder nur eine Gerätenummer angegeben ist oder die Gerätenummern vor und nach dem TO-Token gleich sind, wird **COPY** auf dem Laufwerk selbst ausgeführt und die Quell- und Zieldateien befinden sich auf demselben Laufwerk.

Wenn das Quellgerät (vor TO) ein anderes ist als das Zielgerät (nach TO), wird **COPY** vom MEGA65-BASIC ausgeführt, indem es die Quelldateien in einen RAM-Puffer liest und in das Zielgerät schreibt. In diesem Fall kann der Zieldateiname nicht gewählt werden, er ist derselbe wie der Quelldateiname. Der erweiterte "Gerät-zu-Gerät"-Kopiermodus erlaubt das Kopieren von einzelnen Dateien, mit Mustern (sogenannten "Wildcards") übereinstimmenden Dateien oder allen Dateien eines Datenträgers. Jede Kombination von Geräten ist erlaubt, interne Disketten, SD-Karten-Images, IEC-Diskettenlaufwerke wie die 1541, 1571, 1581 oder CMD-Disketten- und Festplattenlaufwerke.

Notiz: Die Dateitypen **PRG**, **SEQ** und **USR** können kopiert werden. Befinden sich Quelle und Ziel auf demselben Laufwerk, muss der Zieldateiname anders lauten als der Name der Quelldatei.

COPY kann keine **DEL**-Dateien kopieren, die üblicherweise als Titel oder Trennzeichen in Diskettenverzeichnissen verwendet werden. Diese entsprechen nicht den Commodore DOS-Regeln und können nicht von den **OPEN**-Standardroutinen angesprochen werden.

REL-Dateien können nicht von Gerät zu Gerät kopiert werden.

Beispiel: Verwendung von **COPY**:

```
COPY U8 TO U9            : REM KOPIERT ALLE DATEIEN
COPY "CODES" TO "BACKUP" : REM KOPIERT EINZELNE DATEI
COPY "*.TXT",U8 TO U9    : REM KOPIERT ALLE DATEIEN "*.TXT"
COPY "M*",U9 TO U11      : REM KOPIERT ALLE DATEIEN "M*"
```

COS

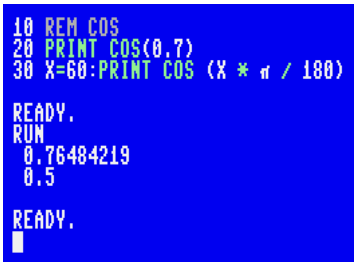
Token: $BE

Format: **COS**(numerischer Ausdruck)

Zweck: Gibt den Kosinus des Arguments zurück. Das Argument wird im Bogenmaß angegeben. Das Ergebnis liegt im Bereich (-1.0 bis +1.0)

Notiz: Ein Argument in **Grad** kann ins benötigte **Bogenmaß** umgerechnet werden, indem es mit $\pi/180$ multipliziert wird.

Beispiel: Verwendung von **COS**:



CURSOR

Format: **CURSOR** <**ON** | **OFF**> [{**,** Spalte**,** Zeile**,** Stil}]
CURSOR {Spalte**,** Zeile**,** Stil}

Zweck: Bewegt den Textcursor an die angegebene Position auf dem aktuellen Textbildschirm.

ON ("an") oder **OFF** ("aus") zeigt den Cursor oder blendet ihn aus.

Spalte und **Zeile** geben die neue Position an.

Stil definiert einen feststehenden (1) oder blinkenden (0) Cursor.

Beispiel: Verwendung von **CURSOR**:

```
10 SCNCLR
20 CURSOR ON,1,2,1       : REM ZEIGE EINEN FESTST. CURSOR IN SPALTE 1, ZEILE 2
30 PRINT "A"; : SLEEP 1
40 CURSOR ,,0            : REM WECHSLE ZU EINEM BLINKENDEN CURSOR
50 PRINT "B"; : SLEEP 1
60 CURSOR OFF            : REM VERSTECKE DEN CURSOR
70 PRINT "C"; : SLEEP 1
80 CURSOR 20,10          : REM BEWEGE DEN CURSOR ZU SPALTE 20, ZEILE 10
90 PRINT "D"; : SLEEP 1
100 CURSOR ,5            : REM BEWEGE DEN CURSOR ZU Z. 5, AENDERE DIE SP. NICHT
110 PRINT "E"; : SLEEP 1
120 CURSOR 0             : REM BEWEGE DEN CURSOR ZUM START DER AKTUELLEN ZEILE
130 PRINT "F"; : SLEEP 1
```

```
 ABC

F                  E
READY.
 
                   D
```

CUT

Token: $E4

Format: **CUT** x, y, Breite, Höhe

Zweck: **CUT** (dt. *"schneide aus"*) wird auf Grafikbildschirmen verwendet und kopiert den Inhalt des angegebenen Rechtecks mit der oberen linken Position **x, y** sowie der **Breite** und **Höhe** in einen Puffer und füllt den Bereich anschließend mit der Farbe des aktuell ausgewählten Stifts.

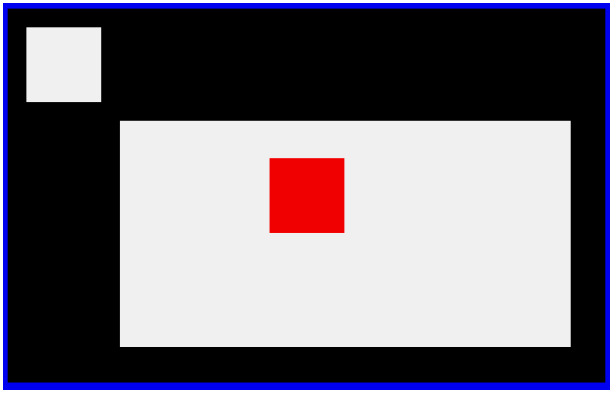
Der Ausschnitt kann mit dem Befehl **PASTE** an beliebiger Stelle eingefügt werden.

Notiz: Die Größe des Rechtecks ist durch die 1 KB-Größe des Cut/Copy/Paste-Puffers begrenzt. Der Speicherbedarf für einen ausgeschnittenen Bereich beträgt (Breite * Höhe * Anzahl der Bitebenen)/8 Bytes. Er darf höchstens 1023 Byte groß sein. Bei einem Bildschirm mit 4 Bitebenen benötigt ein 44 x 44 Pixel großer Bereich beispielsweise 968 Byte.

Beispiel: Verwendung von **CUT**:

```
10 REM CUT
20 SCREEN 320,200,2     : REM OEFFNE GRAFIKBILDSCHIRM 320 X 200 X 2
30 BOX 60,60,300,180,1  : REM ZEICHNE EIN WEISSES RECHTECK
40 PEN 2                : REM WAEHLER ROTEN STIFT AUS
50 CUT  140,80,40,40    : REM SCHNEIDE EINEN 40 X 40 PIXEL BEREICH AUS
60 PASTE 10,10,0,0      : REM FUEGE IHN AN NEUER POSITION EIN
70 GETKEY A$            : REM WARTE AUF TASTENDRUCK
80 SCREEN CLOSE         : REM SCHLIESSE GRAFIKBILDSCHIRM

READY.
```



DATA

Token: $83

Format: **DATA** [Konstante [**,** Konstante ...]]

Zweck: Dient zur Definition von Konstanten, die in einem Programm von **READ**-Anweisungen gelesen werden können. Zahlen und Strings sind erlaubt, Ausdrücke jedoch nicht. Elemente werden durch Kommas getrennt. Strings, die Kommas, Doppelpunkte oder Leerzeichen enthalten, müssen in Anführungszeichen gesetzt werden.

RUN initialisiert den Datenzeiger mit dem ersten Element der **DATA**-Anweisung und schiebt ihn bei jedem gelesenen Element weiter. Sie sind beim Programmieren dafür verantwortlich, dass der Typ der Konstanten mit dem Typ der Variable in der **READ**-Anweisung übereinstimmt. Leere Elemente ohne Konstante zwischen den Kommas sind zulässig und werden interpretiert als Null für numerische Variablen und eine leere Zeichenkette für String-Variablen.

RESTORE kann verwendet werden, um den Datenzeiger auf eine bestimmte Zeile für nachfolgendes Lesen zu setzen.

Notiz: Es ist gute Programmierpraxis, große Mengen von DATA-Anweisungen an das Ende des Programms zu setzen, damit sie die Suche nach Zeilennummern nach einem **GOTO**-Befehl oder anderen Anweisungen mit Zeilennummern als Ziele nicht verlangsamen.

Beispiel: Verwendung von **DATA**:

```
1 REM DATA
10 READ NA$, VE
20 READ N% : FOR I=1 TO N% : READ GL(I) : NEXT I
25 PRINT
30 PRINT "PROGRAMM:";NA$;"  VERSION:";VE
35 PRINT
40 PRINT "PREISENTWICKLUNG:"
50 FOR I=1 TO N%:PRINT GL(I):NEXT I
60 END
80 DATA "MEGA65",1.1
90 DATA 4,0.51,0.36,0.28,0.23

READY.
RUN

PROGRAMM:MEGA65  VERSION: 1.1

PREISENTWICKLUNG:
 0.51
 0.36
 0.28
 0.23
```

DCLEAR

Token: $FE $15

Format: **DCLEAR** [**,D** Laufwerk] [**,U** Gerät]

Zweck: Sendet einen Initialisierungsbefehl an das angegebene Gerät und das angegebene Laufwerk.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Das DOS des Laufwerks schließt alle offenen Dateien, löscht alle Kanäle, gibt die Puffer frei und liest die BAM neu ein. Alle offenen Kanäle auf dem Computer werden ebenfalls geschlossen.

Beispiel: Verwendung von **DCLEAR**:

```
DCLEAR
DCLEAR U9
DCLEAR D0, U9
```

DCLOSE

Token: $FE $0F

Format: **DCLOSE** [**U** Gerät]
DCLOSE # Kanal

Zweck: Schließt eine einzelne Datei oder alle Dateien für das angegebene Gerät.

Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** vergeben worden ist.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

DCLOSE wird entweder mit einem Kanal als Argument oder mit einer Einheitsnummer verwendet, aber nie beides gleichzeitig.

Notiz: Es ist wichtig, alle offenen Dateien zu schließen, bevor ein Programm endet. Andernfalls werden die Puffer nicht freigegeben und, was noch schlimmer ist, offene Dateien, in die geschrieben wurde, können unvollständig (gemeinhin als "Splat"-Dateien bezeichnet) und nicht mehr verwendbar sein.

Beispiel: Verwendung von **DCLOSE**:

```
DCLOSE#2  : REM DATEI, DIE DEM KANAL 2 ZUGEORDNET IST, SCHLIESSEN
DCLOSE U9 : REM ALLE AUF DEM GERAET 9 GEOEFFNETEN DATEIEN SCHLIESSEN
```

DEC

Token: $D1

Format: **DEC**(String-Ausdruck)

Zweck: Gibt den Dezimalwert des Arguments zurück, das als Hexadezimal-String geschrieben wird.

Der Argumentbereich ist "0000" bis "FFFF" (0 bis 65535 in Dezimalzahlen). Das Argument muss 1-4 Hexadezimalziffern haben.

Notiz: Erlaubte Ziffern im Großbuchstaben-/Grafikmodus sind:
0123456789ABCDEF
und im Klein-/Großbuchstabenmodus:
0123456789abcdef

Beispiel: Verwendung von **DEC**:

```
10 REM DEC
20 PRINT DEC("D000")
30 POKE DEC("600"),255

READY.
RUN
 53248

READY.
```

DEF FN

Token: $96

Format: **DEF FN** Name(reelle Variable) = [Ausdruck]

Zweck: Definiert eine Benutzerfunktion mit einer einzelnen Anweisung und einem Argument vom Typ "Real" (reelle Zahl), die einen reellen Wert zurückgibt. Die Definition muss ausgeführt werden, bevor die Funktion in Ausdrücken verwendet werden kann. Das Argument ist eine Dummy-Variable, die bei Verwendung der Funktion durch das Argument ersetzt wird, wenn die Funktion verwendet wird.

Notiz: Der Wert der Dummy-Variable wird sich nicht ändern und die Variable kann in anderen Kontexten ohne Nebeneffekte verwendet werden.

Beispiel: Verwendung von **DEF FN**:

```
1 REM DEF FN
10 PD=π/180
20 DEF FN CD(X)= COS(X*PD): REM COS FUER GRAD
30 DEF FN SD(X)= SIN(X*PD): REM SIN FUER GRAD
40 FOR D=0 TO 360 STEP 90
50 PRINT USING "###";D;
60 PRINT USING " ##.##";FNCD(D);
70 PRINT USING " ##.##";FNSD(D)
80 NEXT D

READY.
RUN
  0  1.00  0.00
 90  0.00  1.00
180 -1.00  0.00
270  0.00 -1.00
360  1.00  0.00

READY.
```

DELETE

Token: $F7

Format: **DELETE** [Zeilenbereich]
DELETE Dateiname [**,D** Laufwerk] [**,U** Gerät] [**,R**]

Zweck: Dient entweder zum Löschen eines Bereichs von Zeilen aus dem BASIC-Programm oder zum Löschen von Dateien von Diskette.

Zeilenbereich besteht aus der ersten und letzten zu löschenden Zeile oder einer einzelnen Zeilennummer. Wenn die erste Nummer weggelassen wird, wird die erste BASIC-Zeile angenommen. Die zweite Zahl im Zeilenbereich ist standardmäßig die letzte BASIC-Zeile.

Dateiname ist entweder ein String in Anführungszeichen (zum Beispiel: **"Daten"**) oder ein String-Ausdruck in Klammern (zum Beispiel: **(FI$)**).

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

R "Recover" (dt. "Wiederherstellen") Wiederherstellen einer zuvor gelöschten Datei. Dies funktioniert nur, wenn es zwischen dem Löschen und der Wiederherstellung keine Schreibvorgänge gab, die den Inhalt der Datei verändert haben.

Notiz: **DELETE Dateiname** arbeitet ähnlich wie **ERASE** und **SCRATCH**.

Nach Ausführung gibt **DELETE Dateiname** automatisch die Systemvariable **DS$** aus, die Erfolg und Anzahl der gelöschten Dateien zeigt. Die vorletzte Zahl, die im Falle eines Ein-/Ausgabefehlers normalerweise die Spurnummer angibt, gibt stattdessen die Anzahl der erfolgreich gelöschten Dateien an.

Im Direktmodus wird bei **DELETE Dateiname** vor der Befehlsausführung zur Sicherheit mit `ARE YOU SURE?` nachgefragt, ob Sie wirklich löschen wollen? Bestätigen Sie diese mit **Y** (für "Yes", dt. "Ja"). Alle anderen Buchstaben brechen den Vorgang ab.

Beispiel: Verwendung von **DELETE**:

```
DELETE 100      : REM LOESCHE ZEILE 100
DELETE 240-350  : REM LOESCHE ALLE ZEILEN VON 240 BIS 350
DELETE 500-     : REM LOESCHE VON ZEILE 500 BIS ZUM ENDE
DELETE -70      : REM LOESCHE VOM START BIS ZU ZEILE 70
DELETE "DRM",U9 : REM LOESCHE DATEI "DRM" AUF GERAET 9
```

DIM

Token: $86

Format: **DIM** Name(Grenzwert) [, Name(Grenzwert) ...]

Zweck: Deklariert die Form, Grenzen und den Typ eines BASIC-Arrays. Als Deklarationsanweisung muss sie nur einmal und vor jeder Verwendung des deklarierten Arrays ausgeführt werden. Ein Array kann eine oder mehrere Dimensionen haben. Eindimensionale Arrays werden oft als Vektoren bezeichnet, während zwei oder mehr Dimensionen eine Matrix definieren. Die untere Grenze einer Dimension ist immer Null, während die obere Grenze wie angegeben ist. Die Regeln für Variablennamen gelten auch für Array-Namen. Sie können Byte-Arrays, Integer-Arrays, Real-Arrays und String-Arrays erstellen. Es ist zulässig, denselben Bezeichner für skalare Variablen und Array-Variablen zu verwenden. Die linke Klammer nach dem Namen kennzeichnet Array-Namen.

Notiz: Byte-Arrays verbrauchen ein Byte pro Element, Integer-Arrays zwei Bytes, Real-Arrays fünf Bytes und String-Arrays drei Bytes für den String-Deskriptor plus die Länge des Strings selbst.
Wenn ein Array-Bezeichner verwendet wird, ohne dass er zuvor deklariert wurde, wird eine implizite Deklaration eines Arrays mit einem Limit von 10 je verwendeter Dimension (maximal 3) durchgeführt.

Beispiel: Verwendung von **DIM**:

```
10 REM DIM
20 DIM A%(8)    : REM ARRAY MIT 9 ELEMENTE
30 DIM XX(2,3)  : REM ARRAY MIT 3X4 = 12 ELEMENTE
40 FOR I=0 TO 8 : A%(I)=PEEK(256+I) : PRINT A%(I);: NEXT:PRINT
50 FOR I=0 TO 2 : FOR J=0 TO 3 : READ XX(I,J):PRINT XX(I,J);: NEXT J,I
60 END
70 DATA 1,-2,3,-4,5,-6,7,-8,9,-10,11,-12

READY.
RUN
 32  51  49  0  0  0  0  0  0
 1 -2  3 -4  5 -6  7 -8  9 -10  11 -12
READY.
```

DIR

Token: $EE (DIR) $FE $29 (ECTORY)

Format: **DIR** [Dateinamenmuster] [**,W**] [**,R**] [**,D** Laufwerk] [**,U** Gerät]
DIRECTORY [Dateinamenmuster] [**,W**] [**,R**] [**,D** Laufwerk] [**,U** Gerät]
$ [Dateinamenmuster] [**,W**] [**,R**] [**,D** Laufwerk] [**,U** Gerät]

Zweck: Zeigt einen Dateikatalog/Verzeichnis der angegebenen Diskette bzw. des angebenen Diskettenimage.

Der Parameter **W** (Wide, dt. "breit") listet das Verzeichnis drei Spalten breit auf dem Bildschirm auf und pausiert, wenn der Bildschirm mit einer Seite (63 Verzeichniseinträge) gefüllt ist. Durch Drücken einer beliebigen Taste wird die nächste Seite angezeigt.

Der Parameter **R** (Recoverable, dt. "Wiederherstellbar") umfasst Dateien im Verzeichnis, die als gelöscht gekennzeichnet sind, aber noch wiederherstellbar sind.

Dateimuster ist entweder ein String in Anführungszeichen, zum Beispiel: **"da*"** oder ein Stringausdruck in Klammern, z. B. **(DI$)**

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Hinweis: **DIR** ist ein Synonym für **DIRECTORY** oder **CATALOG** und erzeugt die gleiche Liste. **Dateimuster** kann zum Filtern der Liste verwendet werden. Die Platzhalterzeichen * und **?** können hierfür verwendet werden. Hinzufügen von **,T=** zum Dateimuster, wobei **T** Folgendes angibt einen Dateityp von **P**, **S**, **U** oder **R** (für **P**RG, **S**EQ, **U**SR, **R**EL), filtert die Ausgabe auf diesen Dateityp.

Das Abkürzungssymbol **$** kann nur im Direktmodus verwendet werden.

Beispiel: Verwendung von **DIR**:

```
DIR
   0 "C65NOTEPAD V1.01"     1D
  50 "C65NOTEPAD"           PRG
   3 "README"               PRG
  33 "NPAGES"               PRG
   8 "BPAGE1"               PRG
   8 "BPAGE2"               PRG
   8 "BPAGE3"               PRG
   8 "BPAGE4"               PRG
   8 "BTEXT1PETSCII"        SEQ
   8 "BTEXT1UTF8"           SEQ
3026 BLOCKS FREE

READY.

DIR "*,T=S"
   0 "C65NOTEPAD V1.01"     1D
   8 "BTEXT1PETSCII"        SEQ
   8 "BTEXT1UTF8"           SEQ
3026 BLOCKS FREE

READY.
```

Nachfolgend ein Beispiel, das zeigt, wie ein Verzeichnis mit dem Parameter **W**ide (dt. "breit") aussieht:

```
DIR W

   0 "BASIC EXAMPLES  "
   2 "AUTOBOOT.C65"      P     1 "DO"             P     2 "GOTO"           P
   1 "BEGIN"             P     3 "ELLIPSE"        P     2 "GRAPHIC"        P
   1 "BEND"              P     1 "ELSE"           P     1 "HELP"           P
   2 "BOX"               P     1 "EL"             P     1 "IF"             P
   1 "BUMP"              P     1 "ENVELOPE"       P     2 "INPUT#"         P
   1 "CBM-PEOPLE"        S     1 "ERR$"           P     2 "INPUT"          P
   2 "CHAR"              P     1 "ER"             P     2 "JOY"            P
   1 "CHR$"              P     2 "EXIT"           P     1 "LINE INPUT#"    P
   4 "CIRCLE"            P     1 "FGOSUB"         P     1 "LINE"           P
   1 "CLOSE"             P     1 "FGOTO"          P     1 "LOADIFF"        P
   1 "CLR"               P     2 "FILTER"         P     1 "LOOP"           P
   2 "COLLISION"         P     1 "FN"             P     1 "MID$"           P
   1 "CONT"              P     1 "FOR"            P     1 "MOD"            P
   1 "CURSOR"            P     1 "FREAD"          P     1 "MOVSPR"         P
   2 "CUT"               P     1 "FRE"            P     1 "NEXT"           P
  21 "DATA BASE"         R     1 "FWRITE"         P     2 "ON"             P
   1 "DATA"              P     2 "GCOPY"          P     3 "PAINT"          P
   1 "DEF FN"            P     2 "GET#"           P     3 "PALETTE"        P
   3 "DEMOSPRITES1"      P     1 "GETKEY"         P     1 "PEEK"           P
   1 "DIM"               P     1 "GET"            P     3 "PEN"            P
   6 "DMODE"             P     2 "GOSUB"          P     1 "PIXEL"          P
```

DISK

Token: $FE $40

Format: **DISK** Befehl [**,U** Gerät]
@ Befehl [**,U** Gerät]

Zweck: Sendet einen Befehlsstring an das angegebene Diskettengerät.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Befehl ist ein String-Ausdruck.

Notiz: Der Befehlsstring wird von dem Diskettengerät interpretiert und muss mit der verwendeten DOS-Version kompatibel sein. Lesen Sie das Handbuch des Laufwerks für die mögliche Befehle.

Die Verwendung von **DISK** ohne Parameter gibt den Diskettenstatus aus.

Die Abkürzung mit der @-Taste kann nur im Direktmodus verwendet werden.

Beispiel: Verwendung von **DISK**:

```
DISK "I0"   : REM INITIALISIERT DIE DISKETTE IN LAUFWERK 0
DISK "U0>9" : REM AENDERE DIE GERAETENUMMER AUF 9
```

DLOAD

Token: $F0

Format: **DLOAD** Dateiname [**,D** Laufwerk] [**,U** Gerät]
DLOAD "$[Muster=Typ]"[**,D** Laufwerk] [**,U** Gerät]
DLOAD "$$[Muster=Typ]"[**,D** Laufwerk] [**,U** Gerät]

Zweck: "Disk LOAD" (dt. "von Diskette laden") lädt in der ersten Variante eine Datei vom Typ **PRG** in den für BASIC-Programme reservierten Speicher.

Die zweite Variante von **DLOAD** lädt ein Verzeichnis in den Speicher, das dann mit **LIST** oder **LISTP** angezeigt werden kann. Es ist wie ein BASIC-Programm aufgebaut, allerdings werden hier die Dateigrößen anstelle von Zeilennummern angezeigt.

Die dritte Variante ist ähnlich wie die zweite, aber die Dateien sind hier aufsteigend nummeriert. Diese Auflistung kann wie ein BASIC-Programm mit den Tasten **F9** oder **F11** geblättert, bearbeitet, aufgelistet, gespeichert oder gedruckt werden.

Ein Filter kann durch Angabe eines Musters oder eines Musters und eines Typs angewendet werden. Der Stern ***** passt auf den Rest des Namens, während das **?** auf jedes einzelne Zeichen passt. Die Typangabe kann ein Zeichen aus (**P**, **S**, **U**, **R**) sein, d. h. **P**rogramm-, **S**equenz-, **U**ser ("Benutzer")- oder **R**EL-Datei.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Die Ladeadresse, die in den ersten beiden Bytes der Datei gespeichert ist, wird ignoriert. Das Programm wird in den BASIC-Speicher geladen. Dies ermöglicht das Laden von BASIC-Programmen, die auf anderen Computern mit unterschiedlicher Speicherkonfigurationen gespeichert wurden. Nach dem Laden ist das Programm neu verlinkt und bereit, mit **RUN** gestartet oder bearbeitet zu werden. Es ist möglich, **DLOAD** in einem

laufenden Programm zu verwenden. Dies wird Überlagerung oder Verkettung genannt. Wenn Sie dies tun, dann ersetzt das neu geladene Programm das aktuelle und die Ausführung beginnt automatisch in der ersten Zeile des neuen Programms. Variablen, Arrays und Strings aus dem aktuellen Lauf bleiben erhalten und können auch vom neu geladenen Programm verwendet werden.

Jedes **DLOAD**, ob Programm oder Verzeichnis, ersetzt (ohne Nachfrage) den Inhalt (Programme), der sich gerade im Speicher befindet. Das bedeutet zum Beispiel, dass das aktuelle BASIC-Programm im Speicher überschrieben wird, wenn Sie ein Verzeichnis mit **DLOAD** laden.

Beispiels: Verwendung von **DLOAD**:

```
DLOAD "PARADIES"
DLOAD "MEGA-TOOLS",U9
DLOAD (FI$),U(UN%)
DLOAD "$"      : REM LADE KOMPLETTES VERZEICHNIS - MIT DATEIGROESSEN
DLOAD "$$"     : REM LADE KOMPLETTES VERZEICHNIS - ALS LISTING ZUM SCROLLEN
DLOAD "$$X*=P" : REM VERZEICHNIS DER PRG-DATEIEN, DIE MIT 'X' BEGINNEN
```

DMA

Token: $FE $1F

Format: **DMA** Befehl [, Länge, Quelladresse, Quellbank, Zieladresse, Zielbank [, Unterbefehl]]

Zweck: **DMA** "Direct Memory Access" (dt. "direkter Speicherzugriff"). Die Verwendung von **DMA** ist nicht mehr üblich und wurde durch **EDMA** abgelöst.

Befehl 0: Kopieren, 1: Mischen, 2: Austauschen, 3: Füllen

Länge Anzahl der Bytes

Quelladresse 16 Bit-Adresse des Lesebereichs oder des Füll-Bytes.

Quellbank Bank-Nummer der Quelle (wird im Füll-Modus ignoriert)

Zieladresse 16 Bit-Adresse des Schreibbereichs

Zielbank Bank-Nummer des Ziels

Unterbefehl Unterbefehl

Notiz: **DMA** hat Zugriff auf den unteren 1MB-Adressbereich, der in 16 Bänken zu je 64 KB organisiert ist. Um diese Einschränkung zu vermeiden, verwenden Sie den **EDMA**-Befehl, der Zugriff auf den gesamten 256 MB-Adressbereich hat.

Beispiel: Verwendung von **DMA**:

Eine Sequenz von **DMA**-Aufrufen zur Demonstration schneller Bildschirmzeichenbefehle:

```
10 REM DMA
20 REM SPEICHERE BILDSCHIRM NACH $00000 BANK 4
30 DMA 0, 80*25, 2048, 0, 0, 4
40 REM FUELLE BILDSCHIRM MIT LEERZEICHEN
50 DMA 3, 80*25, 32, 0, 2048, 0
60 REM STELLE BILDSCHIRM VON $00000 BANK 4 WIEDER HER
70 DMA 0, 80*25, 0, 4, 2048, 0
80 REM VERTAUSCHE DEN INHALT VON ZEILE 1 UND 2 DES BILDSCHIRMS
90 DMA 2, 80, 2048, 0, 2048+80, 0

READY.
```

DMODE

Token: $FE $35

Format: **DMODE** Blockieren, Ergänzen, Schablone, Stil, Dicke

Zweck: "Display MODE" (dt. "Anzeigemodus") setzt mehrere Parameter des Grafikkontexts, der von Zeichenbefehlen verwendet wird.

Mode	Values
Blockieren	0 - 1
Ergänzen (XOR)	0 - 1
Schablone	0 - 1
Stil	0 - 3
Dicke	1 - 8

DO

Token: $EB

Format: **DO** ... **LOOP**
DO [<**UNTIL** | **WHILE**> logischer Ausdruck]
. . . Befehle [**EXIT**]
LOOP [<**UNTIL** | **WHILE**> logischer Ausdruck]

Zweck: **DO** und **LOOP** definieren den Start einer BASIC-Schleife. Die Verwendung von **DO** und **LOOP** ohne jegliche Modifikatoren erzeugt eine Endlosschleife, die nur durch die Anweisung **EXIT** verlassen werden kann. Die Schleife kann gesteuert werden, indem man **UNTIL** oder **WHILE** nach der Anweisung **DO** oder **LOOP** hinzufügt.

Notiz: **DO**-Schleifen können auch ineinander verschachtelt werden. Ein **EXIT**-Befehl verlässt nur die aktuelle Schleife.

Beispiel: Verwendung von **DO**, **EXIT**, **LOOP**, **WHILE** und **UNTIL**:

```
10 REM EXIT
20 A = 10
30 DO                      : REM SCHLEIFENBEGINN
40   PRINT A               : REM GIB A AUS
50   A = A - 1             : REM ZIEHE VON A EINS AB
60   IF A < 5 THEN EXIT    : REM VERLASSE SCHLEIFE, FALLS A KLEINER ALS 5 IST
70 LOOP                    : REM SCHLEIFENENDE
80 PRINT "ENDE"            : REM PROGRAMMENDE

READY.
RUN
 10
 9
 8
 7
 6
 5
ENDE

READY.
```

```
100 REM DO
110 PW$="":DO
120 GET A$:PW$=PW$+A$
130 LOOP UNTIL LEN(PW$)>7 OR A$=CHR$(13)
140 :
150 DO : REM WARTE AUF NUTZERENTSCHEIDUNG (J)A ODER (N)EIN
160 GET A$
170 LOOP UNTIL A$="J" OR A$="N" OR A$="J" OR A$="N"
180 :
190 DO WHILE ABS(EPS) > 0.001
200 GOSUB 280: REM AUFRUF UNTERPROGRAMM
210 LOOP
220 :
230 I%=0 : REM INTEGER-SCHLEIFE VON 1 BIS 100
240 DO
250 I%=I%+1
260 LOOP WHILE I% < 101
270 :
280 REM UNTERPROGRAMM
290 RETURN

READY.
```

DOPEN

Token: $FE $0D

Format: **DOPEN#** Kanal**,** Dateiname [**,L** [Recordlänge]] [**,W**] [**,D** Laufwerk] [**,U** Gerät]

Zweck: Öffnet eine Datei zum Lesen oder Schreiben.

Kanal Kanalnummer, wobei bei:

- **1 <= Kanal <= 127** der Zeilenabschluss CR ist.
- **128 <= Kanal <= 255** der Zeilenabschluss CR LF ist.

L zeigt an, dass die Datei eine relative Datei ist, die sowohl für den Lese-/Schreibzugriff als auch für den wahlfreien Zugriff geöffnet ist. Das Argument **Recordlänge** ist für die Erstellung relativer Dateien obligatorisch. Bei bestehenden relativen Dateien wird **Recordlänge** als Sicherheitsprüfung verwendet, falls vorhanden.

W öffnet eine Datei für den Schreibzugriff. Die Datei darf nicht existieren.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **DOPEN#** kann verwendet werden, um alle Dateitypen zu öffnen. Der sequenzielle Dateityp **SEQ** ist der Standard. Der relative Dateityp **REL** wird mit dem Parameter **L** ausgewählt. Andere Dateitypen müssen im Dateinamen angegeben werden, z.B. durch Anhängen von **,P** an den Dateinamen für **PRG**-Dateien oder **,U** für **USR**-Dateien.

Wenn das erste Zeichen des Dateinamens ein At-Zeichen '@' ist, wird es als "Speichern und Ersetzen"-Vorgang interpretiert. Es wird **nicht** empfohlen, diese Option auf den Laufwerken 1541 und 1571 zu verwenden, da diese einen "Speichern und Ersetzen"-Fehler in ihrem DOS enthalten.

Beispiel: Verwendung von **DOPEN**:

```
DOPEN#5,"DATEN",U9
DOPEN#130,(DD$),U(UN%)
DOPEN#3,"NUTZERDATEI,U"
DOPEN#2,"DATENBANK",L240
DOPEN#4,"MEINPROG,P" : REM OEFFNE PRG-DATEI
```

DOT

Token: $FE $4C

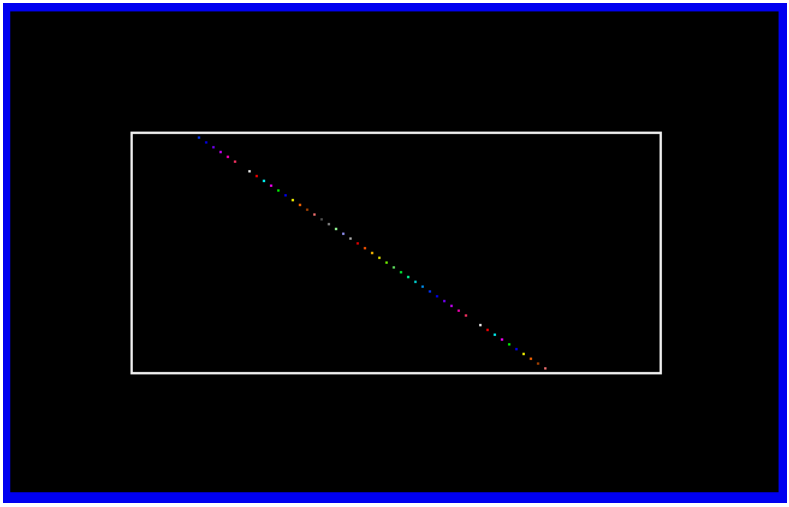
Format: **DOT** x, y [,Farbe]

Zweck: **DOT** zeichnet ein Pixel an den Bildschirmkoordinaten x und y. Der optionale dritte Parameter bestimmt die zu verwendende **Farbe**. Wird er nicht angegeben, wird die aktuelle Stiftfarbe verwendet.

Beispiel: Verwendung von **DOT**:

```
10 REM DOT
20 SCREEN 320,200,5
30 BOX 50,50,270,150
40 VIEWPORT 50,50,220,100
50 FORI=0TO127
60 DOT I+I+I,I+I,I
70 NEXT
80 GETKEY A
90 SCREEN CLOSE

READY.
```



DPAT

Token: $FE $36

Format: **DPAT** Typ [**,** Anzahl**,** Muster ...]

Zweck: "Drawing PATtern" (dt. "Zeichenmuster") setzt das Muster des Grafikkontextes für Zeichenbefehle.

Es gibt vier vordefinierte Mustertypen, die durch Angabe der Typennummer (1, 2, 3 oder 4) als einzelnen Parameter ausgewählt werden können.

Ein Wert von Null für die Typnummer bedeutet ein benutzerdefiniertes Muster. Dieses Muster kann mit Hilfe einer Bitfolge festgelegt werden, die entweder aus 8, 16, 24 oder 32 Bits besteht. Die Anzahl der verwendeten Musterbytes wird als zweiter Parameter angegeben. Er bestimmt, wie viele Musterbytes (1, 2, 3 oder 4) folgen.

- **Typ** 0-4
- **Anzahl** Anzahl der folgenden Musterbytes (1-4)
- **Muster** Musterbytes

DS

Format: **DS**

Zweck: **DS** enthält den Status der letzten Laufwerksoperation. Es handelt sich um eine flüchtige Systemvariable. Jede Verwendung löst das Lesen des Status auf dem aktuell verwendeten Gerät aus. **DS** ist mit der String-Variablen **DS$** gekoppelt, die zur gleichen Zeit aktualisiert wird. Das Lesen des Status von einem Gerät löscht automatisch den Status auf diesem Gerät, so dass nachfolgende Lesevorgänge 0 zurückgeben, wenn seitdem keine weitere Aktivitäten stattgefunden haben.

Notiz: **DS** ist eine reservierte Systemvariable.

Beispiel: Verwendung von **DS**:

```
100 DOPEN#1,"DATEN"
110 IF DS<>0 THEN PRINT"ICH KANN DIE DATEI (DATEN) NICHT OEFFNEN":STOP

READY.
```

DS$

Format: **DS$**

Zweck: **DS$** enthält den Status der letzten Diskettenoperation in folgendem Textformat: Code, Nachricht, Spur, Sektor.

DS$ ist an die numerische Variable **DS** gekoppelt. **DS$** wird aktualisiert, wenn **DS** verwendet wird.

DS$ ist auf **00,OK,00,00** gesetzt, falls kein Fehler vorliegt, ansonsten ist er auf eine DOS-Fehlermeldung gesetzt (entsprechend den in den Handbüchern der jeweiligen Laufwerke erläuterten Fehlermeldungen).

Notiz: **DS$** ist eine reservierte Systemvariable.

Beispiel: Verwendung von **DS$**:

```
100 DOPEN#1,"DATEN"
110 IF DS<>0 THEN PRINT"ICH KANN DIE DATEI (DATEN) NICHT OEFFNEN":STOP

READY.
```

DSAVE

Token: $EF

Format: **DSAVE** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: "Disk SAVE" (dt. "Speichern auf Diskette") speichert das BASIC-Programm, das sich aktuell im Speicher befindet, in eine Datei des Typs **PRG**.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Die maximale Länge des Dateinamens beträgt 16 Zeichen. Wenn das erste Zeichen des Dateinamens ein At-Zeichen '@' ist, wird dies als "Speichern und Ersetzen" interpretiert. Es wird **nicht** empfohlen, diese Option bei den Diskettenlaufwerken 1541 und 1571 zu verwenden, da diese einen "Speichern und Ersetzen"-Fehler in ihrem DOS enthalten.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **DVERIFY** kann nach **DSAVE** verwendet werden, um zu testen, ob das abgespeicherte Programm mit dem im Speicher befindlichen Programm übereinstimmt.

Beispiel: Verwendung von **DSAVE**:

```
DSAVE "ABENTEUER"
DSAVE "ZORK-I",U9
DSAVE "LABYRINTH",D1,U10
```

DT$

Format: **DT$**

Zweck: **DT$** enthält das aktuelle Datum und wird vor jeder Verwendung von der RTC (Real-Time Clock, dt. "Echtzeituhr") aktualisiert. **DT$** ist wie folgt formatiert: **DD-MON-YYYY**, zum Beispiel: **04-APR-2021** für den 4. April 2021.

Notiz: **DT$** ist eine reservierte Systemvariable.

Beispiel: Verwendung von **DT$**:

```
10 REM DT$
20 PRINT "HEUTE IST: ";DT$

READY.
RUN
HEUTE IST: 08-DEC-2021

READY.
```

DVERIFY

Token: $FE $14

Format: **DVERIFY** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: "Disk VERIFY" (dt. etwa "Überprüfung mit Diskette") vergleicht das akuelle BASIC-Programm im Speicher mit einer Datei des Typs **PRG** auf Diskette.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Der **DVERIFY**-Befehl kann nur auf Gleichheit prüfen. Er gibt keine Informationen über die Anzahl oder Position der unterschiedlich bewerteten Bytes zurück. **DVERIFY** beendet sich entweder mit der Meldung **OK** oder mit **VERIFY ERROR**.

Beispiel: Verwendung von **DVERIFY**:

```
DVERIFY "ABENTEUER"
DVERIFY "ZORK-I",U9
DVERIFY "LABYRINTH",D1,U10
```

EDIT

Format: **EDIT** <**ON** | **OFF**>

Zweck: **EDIT** schaltet den eingebauten Editor entweder mit **EDIT ON** in den Textmodus oder mit **EDIT OFF** in den BASIC-Programmeditor.

Nach dem Einschalten oder einem Reset (Zurücksetzen) wird der Editor als BASIC-Programmeditor initialisiert.

Nachdem der Editor mit **EDIT ON** in den Textmodus versetzt wurde, sind die Unterschiede zum Programmmodus folgende:

Der Editor führt keine Tokenisierung/Parsing durch. Alle Texte, die nach einer Zeilennummer eingegeben werden, bleiben reiner Text. BASIC-Schlüsselwörter wie **FOR** und **GOTO** werden nicht, wie im Programmmodus, in BASIC-Token umgewandelt.

Die Zeilennummern werden nur für die Textorganisation, wie Sortieren, Löschen, Auflisten usw., verwendet. Wenn der Text mit **DSAVE** in einer Datei gespeichert wird, wird eine sequentielle Datei (Typ **SEQ**) geschrieben, nicht eine Programmdatei (**PRG**), wie das im Programmmodus der Fall wäre. Zeilennummern werden nicht in die Datei geschrieben.

DLOAD im Textmodus kann nur sequentielle Dateien laden. Für die Bearbeitung werden automatisch Zeilennummern erzeugt.

Der Modus des Editors lässt sich an der Eingabeaufforderung erkennen: Im Programmmodus lautet die Eingabeaufforderung **READY.**, während im Textmodus die Eingabeaufforderung **OK** lautet.

Der Textmodus betrifft nur eingegebene Zeilen mit führenden Nummern, Zeilen ohne Zeilennummer werden wie übich als BASIC-Befehle ausgeführt.

Sequentielle Dateien, die mit dem Texteditor erstellt wurden, können (ohne sie zu Laden) mit **TYPE <Dateiname>** auf dem Bildschirm angezeigt werden.

Beispiel: Verwendung von **EDIT**:

EDMA

Token: $FE $21

Format: **EDMA** Befehl**,** Länge**,** Quelle**,** Ziel [**,** Unterbefehl**,** Modifikator]

Zweck: **EDMA** ("Extended Direct Memory Access", dt. "Erweiterter direkter Speicherzugriff") ist die schnellste Methode, Speicherbereiche mit Hilfe des DMA-Controllers zu manipulieren.

Befehl 0: Kopieren, 1: Mischen, 2: Austauschen, 3: Füllen.

Länge Anzahl der Bytes (Maximum = 65535).

Quelle 28 Bit-Adresse des Lesebereichs oder Füll-Byte.

Ziel 28 Bit-Adresse des Schreibbereichs.

Unterbefehl Unterbefehl

Modifikator Modifikator

Notiz: **EDMA** kann auf den gesamten Adressbereich von 256 MB zugreifen, mit bis zu 28 Bits für die Adressen von Quelle und Ziel.

Beispiel: Verwendung von **EDMA**:

```
10 REM EDMA
20 REM KOPIERE SKALARE VARIABLEN INS ATTIC RAM
30 EDMA 0, $800, $F700, $8000000
40 REM FUELLE DEN TEXTBILDSCHIRM MIT LEERZEICHEN
50 EDMA 3, 80*25, 32, 2048
60 REM KOPIERE DEN TEXTBILDSCHIRM INS ATTIC RAM
70 EDMA 0, 80*25, 2048, $8000800

READY.
```

EL

Format: **EL**

Zweck: **EL** hat den Wert der Zeile, in der der letzte BASIC-Fehler aufgetreten ist oder den Wert -1, wenn es bisher keinen Fehler gab.

Notiz: **EL** ist eine reservierte Systemvariable.

Diese Variable wird normalerweise in einer **TRAP**-Routine verwendet, wo die Fehlerzeile von **EL** übernommen wird.

Beispiel: Verwendung von **EL**:

```
10 REM EL
20 TRAP 70
30 PRINT SQR(-1)       : REM PROVOZIERE FEHLER
40 PRINT "IN ZEILE 40" : REM HIER ZUM FORTSETZEN
50 END
60 :
70 IF ER>0 THEN PRINT ERR$(ER);" FEHLER"
80 PRINT " IN ZEILE";EL
90 RESUME NEXT         : REM FORTSETZEN NACH FEHLER

READY.
RUN
ILLEGAL QUANTITY FEHLER
 IN ZEILE 30
IN ZEILE 40

READY.
```

ELLIPSE

Token: $FE $30

Format: **ELLIPSE** x-Mittelpunkt**,** y-Mittelpunkt**,** x-Radius**,** y-Radius [**,** Status **,** Start**,** Stopp]

Zweck: Zeichnet eine Ellipse

x-Zentrum x-Koordinate des Mittelpunkts in Pixel

y-Zentrum y-Koordinate des Mittelpunkts in Pixel

x-Radius x-Radius der Ellipse in Pixel

y-Radius x-Radius der Ellipse in Pixel

Status kontolliert die Füllung, das Zeichnen von Bögen und die Ausrichtung der 0 Grad-Position der Winkel. Die Standardeinstellung (Null) bedeutet, dass eine Ellipse bzw. ein elliptischer Bogen nicht gefüllt, keine Schenkel gezeichnet werden und der 0 Grad-Winkel bei der 3 Uhr-Position beginnt.

Bit	Name	Wert	Aktion, falls gesetzt
0	Füllen	1	Füllt die Ellipse oder den elliptischen Bogen mit der aktuellen Zeichenfarbe.
1	Schenkel	2	Unterdrückt das Zeichnen der Schenkel bei einem elliptischer Bogen.
2	Oben	4	Der 0 Grad-Winkel soll auf die 12 Uhr-Position (oben) zeigen.

Die Einheiten für den Start- und Stopp-Winkel sind Grad im Bereich von 0 bis 360. Der 0 Grad-Winkel beginnt bei der 3 Uhr-Position und bewegt sich im Uhrzeigersinn. Ein gesetztes "Oben" (Bit 4) verschiebt den 0 Grad-Winkel und die Startposition auf die 12 Uhr-Position.

Start Startwinkel für das Zeichnen eines elliptischen Bogens.

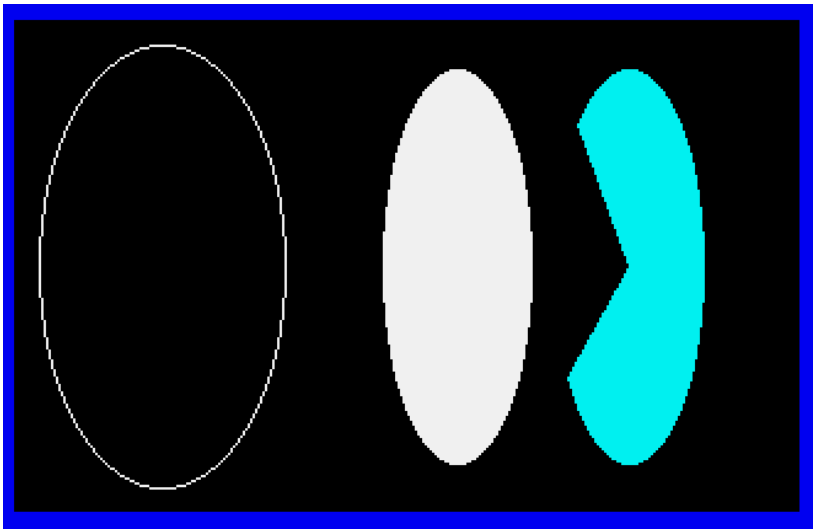
Stopp Startwinkel für das Zeichnen eines elliptischen Bogens.

Wenn eine vollständige Ellipse gezeichnet werden soll, sollten **Start** und **Stopp** entweder weggelassen oder beide auf Null gesetzt werden (nicht 0 und 360). Das Zeichnen und Füllen von vollen Ellipsen ist viel schneller als die Verwendung von Ellipsenbögen.

Beispiel: Verwendung von **ELLIPSE**:

```
100 REM ELLIPSE
110 SCREEN 320,200,4          : REM OEFFNE GRAFIKSCHIRM
120 :
130 ELLIPSE 60,100,50,90      : REM MP.:  60,100  X-RAD: 50  Y-RAD: 90
140 ELLIPSE 180,100,30,80,1   : REM MP.: 180,100  X-RAD: 30  Y-RAD: 80  GEFUELLT
150 PEN 3                     : REM ZEICHENFARBE 3
160 ELLIPSE 250,100,30,80,1,250,120 : REM MP.: 250,100  X-RAD: 30 Y-RAD: 80
165                                 : REM START: 250 GRAD  STOPP: 120 GRAD
166                                 : REM GEFUELLTER ELLIPTISCHER BOGEN
180 GETKEY K$                 : REM WARTE AUF TASTENDRUCK
190 SCREEN CLOSE              : REM SCHLIESSE GRAFIKSCHIRM
200 END

READY.
```



ELSE

Token: $D5

Notiz: **ELSE** ist ein Schlüsselwort, das nur in Kombination mit **IF** verwendet wird. Näheres siehe bei **IF** auf Seite 130.

END

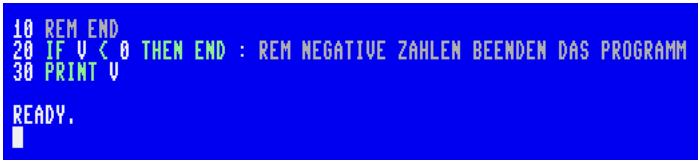
Token: $80

Format: **END**

Zweck: Beendet die Ausführung des BASIC-Programms. Die Eingabeaufforderung **READY.** erscheint, der Computer geht in den Direktmodus und wartet auf Tastatureingaben.

Notiz: Mit **END** werden weder Kanäle gelöscht noch Dateien geschlossen. Auch die Variablendefinitionen sind nach **END** noch gültig. Das Programm kann mit der Anweisung **CONT** fortgesetzt werden. Nach dem Ausführen der letzten Zeile eines Programms wird automatisch **END** ausgeführt.

Beispiel: Verwendung von **END**:



ENVELOPE

Token: $FE $0A

Format: **ENVELOPE** Nummer [{, Anschlag, Abschwell, Halt, Auskling, Wellenform, Impulsbreite}]

Zweck: **ENVELOPE** (dt. "Hüllekurve") wird verwendet, um die Parameter für die Synthese eines Musikinstruments zu definieren.

Nummer Nummer der Hüllkurve (0-9).

Anschlag Anschlagsphase (0-15).

Abschwell Abschwellphase (0-15).

Halt Haltephase (0-15).

Auskling Ausklingphase (0-15).

Wellenform 0: Dreieck, 1: Sägezahn, 2: Rechteck, 3: Rauschen, 4: Ringmodulation.

Impulsbreite Impulsbreite (0-4095) für die Wellenform.

Es gibt 10 Speicherplätze für Instrumentenparameter, voreingestellt mit den folgenden Standardwerten:

Nr.	AN	AB	HA	AU	WF	IB	Instrument
0	0	9	0	0	2	1536	Klavier
1	12	0	12	0	1		Akkordeon
2	0	0	15	0	0		Dampforgel
3	0	5	5	0	3		Trommel
4	9	4	4	0	0		Flöte
5	0	9	2	1	1		Gitarre
6	0	9	0	0	2	512	Cembalo
7	0	9	9	0	2	2048	Orgel
8	8	9	4	1	2	512	Trompete
9	0	9	0	0	0		Xylophon

Beispiel: Verwendung von **ENVELOPE**:

```
10 REM ENVELOPE
20 ENVELOPE 9,10,5,10,5,2,4000
30 VOL 9
40 TEMPO 30
50 PLAY "T904Q CDEFGAB U3T8 CDEFGAB L","T503Q H CGEQG T7 HCGEQG L"

READY.
```

ER

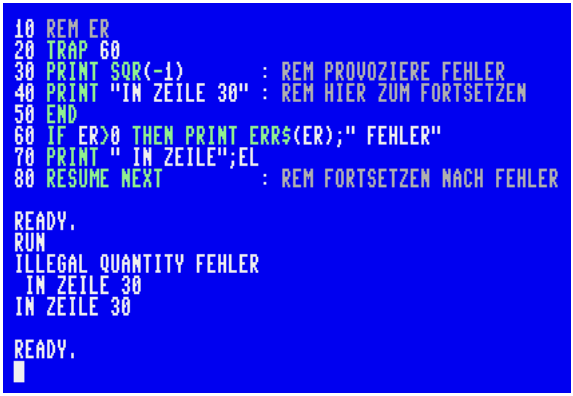
Format: **ER**

Zweck: **ER** hat den Wert des letzten aufgetretenen BASIC-Fehlers oder -1, falls kein Fehler aufgetreten ist.

Notiz: **ER** ist eine reservierte Systemvariable.

Diese Variable wird normalerweise in einer **TRAP**-Routine verwendet, wobei die Fehlernummer aus **ER** übernommen wird.

Beispiel: Verwendung von **ER**:



ERASE

Token: $FE $2A

Format: **ERASE** Dateiname [**,D** Laufwerk] [**,U** Gerät] [**,R**]

Zweck: **ERASE** (dt. "lösche") wird verwendet, um eine Datei auf einer Diskette zu löschen.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

R Stellt eine zuvor gelöschte Datei wieder her. Dies funktioniert nur, wenn zwischen dem Löschen und der Wiederherstellung keine Schreiboperationen stattgefunden haben, die den Inhalt der Diskette verändert haben könnten.

Notiz: **ERASE** arbeitet ähnlich wie **DELETE Dateiname** und **SCRATCH**.

Der Erfolg und die Anzahl der gelöschten Dateien können durch Ausgeben oder Verwendung der Systemvariablen **DS$** überprüft werden. Die vorletzte Zahl von **DS$** enthält die Anzahl der erfolgreich gelöschten Dateien (normalerweise enthält die vorletzte Zahl in **DS$** die Spurnummer im Falle eines Ein-/Ausgabefehlers).

Im Direktmodus wird bei **ERASE** vor der Befehlsausführung zur Sicherheit mit `ARE YOU SURE?` nachgefragt, ob Sie wirklich löschen wollen? Bestätigen Sie diese mit `Y` (für "Yes", dt. "Ja"). Alle anderen Buchstaben brechen den Vorgang ab.

Beispiel: Verwendung von **ERASE**:

```
ERASE "DRM",U9 : REM LOESCHE DATEI "DRM" AUF GERAET 9
ARE YOU SURE? Y
01,FILES SCRATCHED,01,00
READY.

ERASE "OLD*"   : REM LOESCHE ALLE DATEIEN, DIE MIT "OLD" BEGINNEN
ARE YOU SURE? Y
01,FILES SCRATCHED,04,00
READY.
```

ERR$

Token: $D3

Format: **ERR$**(Nummer)

Zweck: Wird verwendet, um eine Fehlernummer in einen Fehlerstring umzuwandeln.

Nummer ist eine BASIC-Fehlernummer (1-43).

Diese Funktion wird normalerweise in einer **TRAP**-Routine verwendet, wobei die Fehlernummer aus der reservierten Variable **ER** entnommen wird.

Notiz: Argumente außerhalb des Bereichs (1-43) führen zu einer Ausgabe von **OK**.

Beispiel: Verwendung von **ERR$**:

```
10 REM ERR$
20 TRAP 60
30 PRINT SQR(-1)       : REM PROVOZIERE FEHLER
40 PRINT "IN ZEILE 30" : REM HIER ZUM FORTSETZEN
50 END
60 IF ER>0 THEN PRINT ERR$(ER);" FEHLER"
70 PRINT " IN ZEILE";EL
80 RESUME NEXT         : REM FORTSETZEN NACH FEHLER

READY.
RUN
ILLEGAL QUANTITY FEHLER
 IN ZEILE 30
IN ZEILE 30

READY.
```

EXIT

Token: $FD

Notiz: **EXIT** ist ein Schlüsselwort, das nur in Kombination mit **DO** verwendet wird.

Näheres siehe bei **DO** auf Seite 76.

EXP

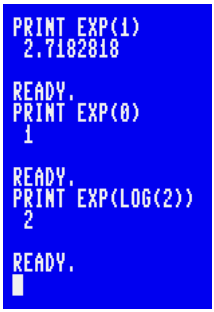
Token: $BD

Format: **EXP**(numerischer Ausdruck)

Zweck: Die **EXP**-Funktion ("exponential function", dt. "Exponentialfunktion") errechnet den Wert der Eulerschen Zahl (**2.71828183**) potenziert mit dem Wert des Argumentes.

Notiz: Ein Argument größer als 88 erzeugt einen Überlauffehler: **OVERFLOW ERROR**.

Beispiel: Verwendung von **EXP**:



FAST

Token: $FE $25

Format: **FAST** [Geschwindigkeit]

Zweck: Setzt die CPU-Geschwindkeit auf 1 MHz, 3.5 MHz oder 40 MHz.

Geschwindigkeit ist die CPU-Geschwindigkeit, wobei:

- **1** die CPU-Geschwindigkeit auf 1 MHz setzt.
- **3** die CPU-Geschwindigkeit auf 3,5 MHz setzt.
- Jeder andere Wert als **1** oder **3** setzt die CPU-Geschwindigkeit auf 40 MHz.

Notiz: Es ist zwar möglich, **FAST** mit einer beliebigen reellen Zahl aufzurufen, allerdings werden hierbei der Dezimalpunkt und alle Nachkommastellen ignoriert.

FAST ist ein Synonym für **SPEED**.

FAST hat keinen Effekt, falls vorher **POKE 0,65** verwendet wurde, um die CPU-Geschwindigkeit auf 40 MHz zu setzen.

Beispiel: Verwendung von **FAST**:

```
10 REM FAST
20 FAST      : REM SETZE DIE GESCHWINDIGKEIT AUF 40 MHZ (MAXIMUM)
30 FAST 1    : REM SETZE DIE GESCHWINDIGKEIT AUF 1 MHZ
40 FAST 3    : REM SETZE DIE GESCHWINDIGKEIT AUF 3.5 MHZ
50 FAST 3.5  : REM SETZE DIE GESCHWINDIGKEIT AUF 3.5 MHZ

READY.
```

FGOSUB

Token: $FE $48

Format: **FGOSUB** numerischer Ausdruck

Zweck: Wertet den angegebenen numerischen Ausdruck aus und ruft dann das Unterprogramm mit der daraus resultierenden Zeilennummer auf.

Notiz: Falls im Programm **FGOSUB**-Befehle vorkommen, kann die Verwendung von **RENUMBER** dazu führen, dass das Programm unter Umständen nicht mehr korrekt funktioniert. Falls dadurch die Zeilen, die mit **FGOSUB** angesprungen werden können, umnummeriert werden, zeigen die Ergebnisse des numerischen Ausdrucks zum Sprung auf die Unterprogramme bei **FGOSUB** nicht mehr auf die korrekten Zeilennummern.

Beispiel: Verwendung von **FGOSUB**:

```
10 REM FGOSUB
20 INPUT "WELCHES UNTERPROGRAMM SOLL ICH AUSFUEHREN?  100,200,300";LI
30 FGOSUB LI : REM HOFFENTLICH EXISTIERT DIESE ZEILENNUMMER
40 GOTO 10   : REM WIEDERHOLE
100 PRINT "IN ZEILE 100":RETURN
200 PRINT "IN ZEILE 200":RETURN
300 PRINT "IN ZEILE 300":RETURN

READY.
```

FGOTO

Token: $FE $47

Format: **FGOTO** numerischer Ausdruck

Zweck: Wertet den angegebenen numerischen Ausdruck aus und springt dann zu der daraus resultierenden Zeilennummer.

Notiz: Falls im Programm **FGOTO**-Befehle vorkommen, kann die Verwendung von **RENUMBER** dazu führen, dass das Programm unter Umständen nicht mehr korrekt funktioniert. Falls dadurch die Zeilen, die mit **FGOTO** angesprungen werden können, umnummeriert werden, zeigen die Ergebnisse des numerischen Ausdrucks zum Sprung bei **FGOTO** nicht mehr auf die korrekten Zeilennummern.

Beispiel: Verwendung von **FGOTO**:

FILTER

Token: $FE $03

Format: **FILTER** SID-Nr. [{, Frequenz, Tiefpass, Bandpass, Hochpass, Resonanz}]

Zweck: Legt die Parameter für einen SID-Klangfilter fest.

SID-Nr. 1: rechter SID, 2: linker SID

Frequenz Filtergrenzfrequenz (0 - 2047)

Tiefpass Tiefpassfilter (0: aus, 1: an)

Bandpass Bandpassfilter (0: aus, 1: an)

Hochpass Hochpassfilter (0: aus, 1: an)

Resonanz Resonanz (0 - 15)

Notiz: Fehlende Parameter behalten ihren aktuellen Wert. Der effektive Filter ist die Summe aller Filtereinstellungen. Dies ermöglicht Bandsperr- und Kerb-Effekte.

Beispiel: Verwendung von **FILTER**:

```
10 REM FILTER
20 PLAY "T7X1O3P9C"
30 SLEEP 0.02
40 PRINT "TIEFPASS-ABTASTUNG" :L=1:B=0:H=0:GOSUB 80
50 PRINT "BANDPASS-ABTASTUNG":L=0:B=1:H=0:GOSUB 80
60 PRINT "HOCHPASS-ABTASTUNG":L=0:B=0:H=1:GOSUB 80
70 GOTO 40
80 REM *** ABTASTUNG ***
90 FOR F = 50 TO 1950 STEP 50
100 IF F >= 1000 THEN FF = 2000-F : ELSE FF = F
110 FILTER 1,FF,L,B,H,15
120 PLAY "X1"
130 SLEEP 0.02
140 NEXT F
150 RETURN

READY.
```

FIND

Token: $FE $2B

Format: **FIND /**String**/** [**,** Zeilenbereich]
FIND "String**"** [**,** Zeilenbereich]

Zweck: **FIND** ist ein Editor-Befehl, der nur im Direktmodus verwendet werden kann. Er durchsucht einen bestimmten Zeilenbereich (falls angegeben), ansonsten wird das gesamte BASIC-Programm durchsucht.

Bei jedem Vorkommen des "Suchstrings" wird die Zeile aufgelistet, wobei der String hervorgehoben ist. Die NO SCROLL-Taste kann verwendet werden, um die Ausgabe anzuhalten.

Notiz: Jedes Zeichen, das ohne Verwendung der Shift-Taste eingegeben werden kann und das nicht Teil des Suchstrings ist, kann anstelle von **/** verwendet werden.

Die Verwendung von doppelten Anführungszeichen als Begrenzungszeichen hat jedoch eine besondere Wirkung: Der Suchtext wird nicht tokenisiert. **FIND "FOR"** sucht nach den drei Buchstaben F, O, und R und nicht nach dem BASIC-Schlüsselwort **FOR**. Daher kann es das Wort **FOR** in String-Konstanten oder REM-Anweisungen, aber nicht im Programmcode finden.

Auf der anderen Seite findet **FIND /FOR/** alle Vorkommen des BASIC-Schlüsselworts, aber nicht den Text "FOR" in Strings.

Teilweise Schlüsselwörter können nicht durchsucht werden. Zum Beispiel wird **FIND /LOO/** nicht das Schlüsselwort **LOOP** finden.

Beispiel: Verwendung von **FIND**:

```
10 REM BEISPIELPROGRAMM
20 A=1:B=2
30 C=A+B
40 PRINT "ERGEBNIS:";C
50 END

READY.
FIND /SPIEL/

10 REM BEISPIELPROGRAMM

READY.
FIND /END/

50 END

READY.
FIND "ERGEB"

40 PRINT "ERGEBNIS:";C

READY.
```

FN

Token: $A5

Format: **FN** Name(numerischer Ausdruck)

Zweck: **FN**-Funktionen sind benutzerdefinierte Funktionen, die einen numerischen Ausdruck als Argument akzeptieren und einen reellen Wert zurückgeben. Vor einer ersten Verwendung muss die Funktion mit **DEF FN** definiert werden.

Beispiel: Verwendung von **FN**:

```
1 REM DEF FN
10 PD=π/180
20 DEF FN CD(X)= COS(X*PD): REM COS FUER GRAD
30 DEF FN SD(X)= SIN(X*PD): REM SIN FUER GRAD
40 FOR D=0 TO 360 STEP 90
50 PRINT USING "###";D;
60 PRINT USING " ##.##";FNCD(D);
70 PRINT USING " ##.##";FNSD(D)
80 NEXT D

READY.
RUN
  0  1.00  0.00
 90  0.00  1.00
180 -1.00  0.00
270  0.00 -1.00
360  1.00  0.00

READY.
```

FONT

Token: $FE $46

Format: **FONT** <**A** | **B** | **C**>

Zweck: **FONT** wird verwendet, um zwischen den Schriftarten zu wechseln und den Codepages "PETSCII" und "erweitertes PETSCII". Das "erweiterte PETSCII" enthält alle ASCII-Symbole, die die in der PETSCII-Codepage fehlen, wobei die Reihenfolge immer noch PETSCII ist. Die ASCII-Symbole werden durch Drücken der Tasten in der nachstehenden Tabelle eingegeben, einige davon erfordern auch das Gedrückthalten der Taste . Die Codes für Großbuchstaben und Kleinbuchstaben sind im Vergleich zu ASCII vertauscht. Der Zeichensatz für Großbuchstaben/Grafik wird nicht geändert.

Code	Taste	PETSCII	ASCII
$5C	Pfund	£	\ (Backslash)
$5E	Pfeil nach oben (neben RESTORE)	↑	^ (Zirkumflex)
$5F	Pfeil nach links (neben der 1)	←	_ (Unterstrich)
$7B	+ Doppelpunkt	┼	{ (öffnende Klammer)
$7C	+ Punkt	▒	| (senkrechter Strich)
$7D	+ Strichpunkt	│	} (schließende Klammer)
$7E	+ Komma	π	~ (Tilde)

Notiz: Die zusätzlichen ASCII-Zeichen, die von FONT A und B bereitgestellt werden, sind nur bei Verwendung des Zeichensatzes für Klein- und Großbuchstaben verfügbar.

Beispiel: Verwendung von **FONT**:

```
10 REM FONT
20 FONT A : REM ASCII
30 FONT B : REM AEHNLICH A, ABER MIT EINER SERIFENSCHRIFT
40 FONT C : REM COMMODORE-ZEICHENSATZ (STANDARDWERT)

READY.
```

FOR

Token: $81

Format: **FOR** Index = Start **TO** Ende [**STEP** Schrittweite] ... **NEXT** [Index]

Zweck: **FOR**-Anweisungen beginnen eine BASIC-Schleife mit einer Indexvariablen.

Index wird bei jeder Iteration um einen konstanten Wert erhöht oder vermindert. Der Standardwert ist, die Variable jeweils um 1 zu erhöhen. Die Indexvariable muss eine reelle Variable sein.

Start wird verwendet, um den Index zu initialisieren. **Ende** wird am Ende einer Iteration geprüft und bestimmt, ob eine weitere Iteration durchgeführt wird oder ob die Schleife beendet wird.

Schrittweite definiert die Veränderung der Indexvariable am Ende einer Iteration. Eine positive Schrittweite erhöht die Variable, eine negative vermindert diese. Wenn keine Schrittweite angegeben wird, wird der Standardwert 1 verwendet.

Notiz: Es ist schlechte Programmierpraxis, den Wert der Variablen **Index** innerhalb der Schleife zu ändern oder mit **GOTO** in einen Schleifenkörper hineinzuspringen oder aus ihm herauszuspringen.

Die Variable **Index** nach **NEXT** ist optional. Wenn sie weggelassen wird, wird die Variable für die aktuelle Schleife angenommen. Mehrere aufeinanderfolgende **NEXT**-Anweisungen können durch Angabe der Indizes in einer durch Komma getrennten Liste kombiniert werden. Die Anweisungen `NEXT I:NEXT J:NEXT K` und `NEXT I,J,K` sind gleichwertig.

Beispiel: Verwendung von **FOR**, **NEXT**, **STEP** und **TO**:

```
10 REM FOR
20 FOR D=0 TO 360 STEP 30
30 R = D * π / 180
40 PRINT D;R;SIN(R);COS(R);TAN(R)
50 NEXT D
60 :
70 DIM M(20,20)
80 FOR I=0 TO 20
90 FOR J=I TO 20
100 M(I,J) = I + 100 * J
110 NEXT J,I

READY.
```

FOREGROUND

Token: $FE $39

Format: **FOREGROUND** Farbe

Zweck: Setzt die Vordergrundfarbe (Textfarbe) des Bildschirms auf das Argument, das im Bereich von 0 bis 31 liegen muss.

Für die Farbwerte und ihre entsprechenden Farben siehe die Tabelle unter **BACKGROUND** auf Seite 19.

Notiz: Die Vordergrundfarbe kann ebenfalls mit **COLOR** verändert werden.

Beispiel: Verwendung von **FOREGROUND**:

FORMAT

Token: $FE $37

Notiz: **FORMAT** ist ein Synonym für **HEADER**.

Näheres siehe bei **HEADER** auf Seite 125.

FRE

Token: $B8

Format: **FRE**(Bank)

Zweck: Gibt die Anzahl der freien Bytes für Bank 0 oder 1 zurück oder, wenn das Argument negativ ist, die ROM-Version.

FRE(0) liefert die Anzahl der freien Bytes in Bank 0, die für den BASIC-Programmcode verwendet wird.

FRE(1) liefert die Anzahl der freien Bytes in Bank 1, der Bank für BASIC-Variablen, Arrays und Strings. **FRE(1)** stößt auch die sogenannte "Garbage Collection" (dt. "Müllsammlung") an. Dies ist ein Prozess, der die Strings, die im oberen Adressbereich der Bank abgelegt sind, sammelt, umsortiert und dadurch den String-Speicher "aufgeräumt" (defragmentiert).

FRE(-1) liefert die sechsstellige Versionsnummer des ROM.

Beispiel: Verwendung von **FRE**:

```
10 REM FRE
20 PM = FRE(0)
30 VM = FRE(1)
40 RV = FRE(-1)
50 PRINT PM;" BYTES FREI FUER PROGRAMMCODE"
60 PRINT VM;" BYTES FREI FUER VARIABLEN"
70 PRINT RV;" NUMMER DER ROM-VERSION"

READY.
RUN
 54870  BYTES FREI FUER PROGRAMMCODE
 54980  BYTES FREI FUER VARIABLEN
 920262  NUMMER DER ROM-VERSION

READY.
```

FREAD

Token: $FE $1C

Format: **FREAD#** Kanal**,** Zeiger**,** Größe

Zweck: Liest **Größe** Bytes von Kanal **Kanal** ab der 32 Bit-Adresse **Zeiger** in den Speicher.

Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **DOPEN** oder **OPEN** verwendet worden ist.

Es muss darauf geachtet werden, dass kein Speicher überschrieben wird, der vom System oder dem Interpreter verwendet wird.

Es wird empfohlen, die Anweisung **POINTER** für das Zeigerargument zu verwenden und den Größenparameter durch Multiplikation der Anzahl der Elemente mit der Elementgröße zu berechnen.

Typ	Elementgröße
Byte-Array	1
Integer-Array	2
Real-Array	5

Beachten Sie, dass die Funktion **POINTER** mit einem String-Argument NICHT die String-Adresse zurückgibt, sondern den String-Deskriptor. Es wird nicht empfohlen, **FREAD** für Strings oder String-Arrays zu verwenden, wenn Sie nicht genau wissen, wie die Interna der Stringspeicherung zu handhaben sind.

Stellen Sie außerdem sicher, dass Sie immer einen Index angeben, wenn Sie ein Array verwenden. Die Startadresse des Arrays **XY()** ist **POINTER(XY(0))**. **POINTER(XY)** gibt die Adresse der skalaren Variablen **XY** zurück.

Beispiel: Verwendung von **FREAD**:

```
10 REM FREAD
20 N=23
30 DIM B&(N)
40 DOPEN#2,"TEXT"
50 FREAD#2,POINTER(B&(0)),N
60 DCLOSE#2

READY.
```

FREEZER

Token: $FE $4A

Format: **FREEZER**

Zweck: **FREEZER** ruft das "Freeze-Menü" des MEGA65 auf.

Notiz: Das "Freeze-Menü" kann auch über ein längeres Drücken (ca. 3 Sekunden) der **RESTORE**-Taste aufgerufen werden.

Der Freezer wird mit der **F3**-Taste wieder verlassen.

FREEZER funktioniert **nicht** mit dem Emulator "xmega65", da dieser das "Freeze-Menü" nicht emuliert.

Beispiel: Verwendung von **FREEZER**:

```
FREEZER
```

FWRITE

Token: $FE $1E

Format: **FWRITE#** Kanal**,** Zeiger**,** Größe

Zweck: Schreibt **Größe** Bytes ab der 32 Bit-Adresse **Zeiger** aus dem Speicher in Kanal **Kanal** .

Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** verwendet worden ist.

Es wird empfohlen, die Anweisung **POINTER** für das Zeigerargument zu verwenden und den Größenparameter durch Multiplikation der Anzahl der Elemente mit der Elementgröße zu berechnen.

Typ	Elementgröße
Byte-Array	1
Integer-Array	2
Real-Array	5

Beachten Sie, dass die Funktion **POINTER** mit einem String-Argument NICHT die String-Adresse zurückgibt, sondern den String-Deskriptor. Es wird nicht empfohlen, **FWRITE** für Strings oder String-Arrays zu verwenden, wenn Sie nicht genau wissen, wie die Interna der Stringspeicherung zu handhaben sind.

Stellen Sie außerdem sicher, dass Sie immer einen Index angeben, wenn Sie ein Array verwenden. Die Startadresse des Arrays **XY()** ist **POINTER(XY(0))**. **POINTER(XY)** gibt die Adresse der skalaren Variablen **XY** zurück.

Beispiel: Verwendung von **FWRITE**:

```
10 REM FWRITE
20 N=23
30 DIM B&(N),C&(N)
40 DOPEN#2,"TEXT"
50 FREAD#2,POINTER(B&(0)),N
60 DCLOSE#2
70 FORI=0TON-1:PRINTCHR$(B&(I));:NEXT
80 FORI=0TON-1:C&(I)=B&(N-1-I):NEXT
90 DOPEN#2,"RUECKWAERTS",W
100 FWRITE#2,POINTER(C&(0)),N
110 DCLOSE#2

READY.
```

GCOPY

Token: $FE $32

Format: **GCOPY** x, y, Breite, Höhe

Zweck: **GCOPY** (dt. *"grafisches Kopieren"*) wird auf Grafikbildschirmen verwendet und kopiert den Inhalt des angegebenen Rechtecks mit der oberen linken Position **x, y** sowie der **Breite** und **Höhe** in einen Puffer.

Der Ausschnitt kann mit dem Befehl **PASTE** an beliebiger Stelle eingefügt werden.

Notiz: Die Größe des Rechtecks ist durch die 1 KB-Größe des Cut/Copy/Paste-Puffers begrenzt. Der Speicherbedarf für einen ausgeschnittenen Bereich beträgt (Breite * Höhe * Anzahl der Bitebenen)/8 Bytes. Er darf höchstens 1023 Byte groß sein. Bei einem Bildschirm mit 4 Bitebenen benötigt ein 44 x 44 Pixel großer Bereich beispielsweise 968 Byte.

Beispiel: Verwendung von **GCOPY**:

```
10 REM GCOPY
20 SCREEN 320,200,2     : REM OEFFNE GRAFIKBILDSCHIRM 320 X 200 X 2
30 BOX 60,60,300,180,1  : REM ZEICHNE EIN WEISSES RECHTECK
40 PEN 2                : REM WAEHLER ROTEN STIFT AUS
50 GCOPY 140,80,40,40   : REM KOPIERE EINEN 40 X 40 PIXEL BEREICH
60 PASTE 10,10,0,0      : REM FUEGE IHN AN NEUER POSITION EIN
70 GETKEY A$            : REM WARTE AUF TASTENDRUCK
80 SCREEN CLOSE         : REM SCHLIESSE GRAFIKBILDSCHIRM

READY.
```

GET

Token: $A1

Format: **GET** Variable

Zweck: Holt das nächste Zeichen (oder den Byte-Wert des nächsten Zeichens) aus der Warteschlange der Tastatur.

Wenn die Variable vom Typ **String** ist und die Warteschlange leer ist, wird ihr ein leerer String zugewiesen. Andernfalls wird ein String mit der Länge 1 erstellt und dieser der String-Variablen zugewiesen.

Wenn der Typ der Variablen **numerisch** ist, wird ihr entweder der Byte-Wert der Taste zugewiesen oder der Wert Null (wenn die Warteschlange leer ist).

GET wartet nicht auf Tastatureingaben. Daher ist es sinnvoll, in regelmäßigen Abständen oder in Schleifen auf Tastendrücke zu prüfen.

Notiz: **GETKEY** funktioniert ähnlich, wartet aber so lange, bis eine Taste gedrückt worden ist.

Beispiel: Verwendung von **GET**:

```
10 REM GET
20 DO: GET A$: LOOP UNTIL A$ <> ""
30 IF A$ = "N" THEN 1000 : REM GEHE NACH NORDEN
40 IF A$ = "W" THEN 2000 : REM GEHE NACH WESTEN
50 IF A$ = "O" THEN 3000 : REM GEHE NACH OSTEN
60 IF A$ = "S" THEN 4000 : REM GEHE NACH SUEDEN
70 IF A$ = CHR$(13) THEN 5000 : REM ENDE
80 GOTO 10

READY.
```

GET#

Token: $A1 '#'

Format: **GET#** Kanal**,** Variable [**,** Variable ...]

Zweck: Liest ein einzelnes Byte aus dem angegebenen Kanal und weist String-Variablen Strings der Länge 1 bzw. numerischen Variablen einen 8-Bit Binärwert (0-255) zu

Dies ist nützlich, um Zeichen (oder Bytes) Byte für Byte aus einem Eingabestrom zu lesen.

Kanal Nummer des Kanals, der bei einem früheren Aufruf von Befehlen wie **DOPEN** oder **OPEN** verwendet worden ist.

Notiz: Alle Werte von 0 bis 255 sind gültig, so dass **GET#** auch zum Lesen von Binärdaten verwendet werden kann.

Beispiel: Verwendung von **GET#**, um ein Dateiverzeichnis von Diskette zu lesen:

```
1 REM GET#
10 OPEN 2,8,0,"$"            : REM OEFFNE VERZEICHNIS
15 IF DS THEN PRINT DS$: STOP: REM KANN  NICHT GELESEN WERDEN
20 GET#2,D$,D$               : REM VERWERFE DIE LADEADRESSE
25 DO                        : REM SCHLEIFENBEGINN
30   GET#2,D$,D$             : REM VERWERFE ZEILENZEIGER
35   IF ST THEN EXIT         : REM ENDE ERREICHT
40   GET#2,LO,HI             : REM LESE GROESSEN-BYTES
45   S=LO + 256 * HI         : REM BERECHNE DATEIGROESSE
50   LINE INPUT#2, F$        : REM LESE DATEINAME
55   PRINT S;F$              : REM GIB DATEI-EINTRAG AUS
60 LOOP                      : REM SCHLEIFENENDE
65 CLOSE 2                   : REM SCHLIESSE VERZEICHNIS

READY.
RUN
 0 "TESTDISK          " 3D
 1   "BEISPIEL"          SEQ
 1   "HALLO"             PRG
 1   "TSCHUESS"          PRG
 3   "GET#-DT"           PRG
 3154 BLOCKS FREE

READY.
```

GETKEY

Token: $A1 $F9 (GET-Token und KEY-Token)

Format: **GETKEY** Variable

Zweck: Holt das nächste Zeichen (oder den Byte-Wert des nächsten Zeichens) aus der Tastatur-Warteschlange. Wenn die Warteschlange leer ist, wartet das Programm, bis eine Taste gedrückt wurde. Nachdem eine Taste gedrückt wurde, wird die Variable entsprechend gesetzt und die Programmausführung fortgesetzt. Bei Verwendung einer String-Variablen wird ein String mit der Länge 1 erstellt und zugewiesen. Wenn die Variable numerisch ist, wird der Byte-Wert zugewiesen.

Beispiel: Verwendung von **GETKEY**:

GO64

Token: $CB $36 $34 (GO-Token und 64)

Format: **GO64**

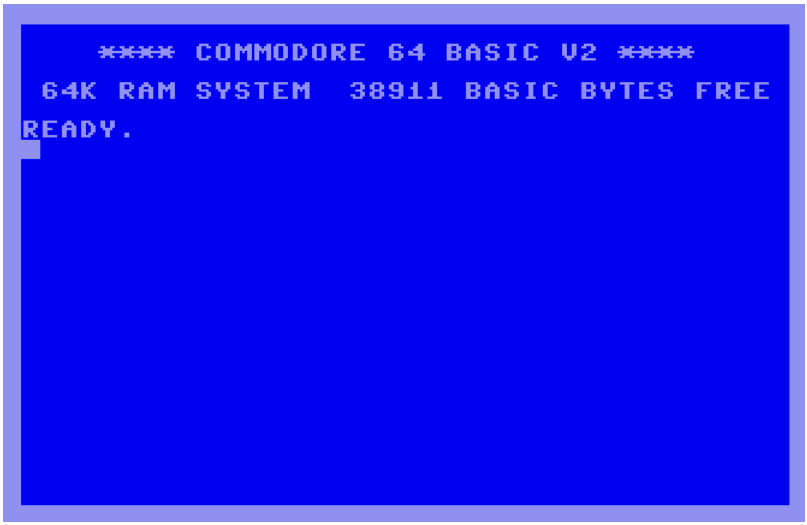
Zweck: Mit **GO64** wechselt der MEGA65 in den C64-Modus. Im Direktmodus wird hierbei ein Sicherheitshinweis **ARE YOU SURE?** (dt. "Sind Sie sicher?") ausgegeben, der zum Fortführen mit **Y** (für "Yes", dt. "Ja") bestätigt werden muss. Wenn Sie hier ein anderes Zeichen als **Y** eingeben, bleiben Sie weiterhin im C65-Modus.

Im C64-Modus wechselt der MEGA65 mit dem Befehl **SYS 58552** wieder in den C65-Modus.

Notiz: Beachten Sie, dass bei einem Moduswechsel das aktuell im Speicher vorhandene BASIC-Programm gelöscht wird.

Beispiel: Verwendung von **GO64**:

```
GO64
ARE YOU SURE? Y
```



GOSUB

Token: $8D

Format: **GOSUB** Zeilennummer

Zweck: **GOSUB** ("**Go**to **Sub**routine", dt. "Sprung ins Unterprogramm") speichert den aktuellen BASIC-Programmzähler und die Zeilennummer auf dem Laufzeitstapel und setzt die Programmausführung an der angegebenen BASIC-Zeilennummer fort. Dies ermöglicht nach der Anweisung **GOSUB** die Wiederaufnahme der Programmausführung, sobald eine **RETURN**-Anweisung in dem aufgerufenen Unterprogramm ausgeführt wird. Aufrufe von Unterprogrammen über **GOSUB** können verschachtelt werden, aber die jeweiligen Unterprogramme müssen immer mit **RETURN** enden, da sonst ein Stapelüberlauf auftreten kann.

Notiz: Im Gegensatz zu anderen Programmiersprachen unterstützt BASIC 65 bei Unterprogrammen weder Argumente noch lokale Variablen.

Programme können hinsichtlich ihrer Laufzeit optimiert werden, indem Unterprogramme am Anfang des Programmquelltextes gruppiert werden. Die **GOSUB**-Aufrufe haben dann niedrige Zeilennummern mit weniger zu dekodierenden Ziffern. Die Unterprogramme werden dadurch schneller gefunden, da die Suche nach Unterprogrammen jeweils am Anfang des Programms beginnt.

Beispiel: Verwendung von **GOSUB**:

```
1 REM GOSUB
10 GOTO 100 : REM SPRINGE ZUM HAUPTPROGRAMM
20 REM *** UNTERPROGRAMM  DISKETTENSTATUS-TEST ***
30 DD=DS : IF DD THEN PRINT "DISKETTEN-FEHLER";DS$
40 RETURN
50 REM *** UNTERPROGRAMM EINGABE J/N ***
60 DO : INPUT "FORTFAHREN (J/N)";A$
70 LOOP UNTIL A$="J" OR A$="N"
80 RETURN
90 REM *** HAUPTPROGRAMM ***
100 DOPEN#2, "DATEN"
110 GOSUB 30 : IF DD THEN DCLOSE#2 : GOSUB 60 : REM FRAGEN
120 IF A$="N" THEN STOP
130 GOTO 100 : REM NOCHMALS VERSUCHEN
```

GOTO

Token: $89 (GOTO) oder $CB $A4 (GO TO)

Format: **GOTO** Zeilennummer
GO TO Zeilennummer

Zweck: Setzt die Programmausführung an der angegebenen BASIC-Zeilennummer fort.

Notiz: Wenn die **Zeilennummer** des Ziels höher ist als die aktuelle Zeilennummer, beginnt die Suche in der aktuellen Zeile und geht von dort aus weiter zu höheren Zeilennummern. Wenn die **Zeilennummer** des Ziels niedriger ist, beginnt die Suche bei der ersten Zeilennummer des Programms. Es ist möglich, die Laufgeschwindigkeit des Programms zu optimieren, indem häufig verwendete Ziele am Anfang gruppiert werden (mit niedrigeren Zeilennummern).

GOTO (als einzelnes Wort geschrieben) wird schneller ausgeführt als die Befehlssequenz **GO TO**.

Beispiel: Verwendung von **GOTO**:

```
1 REM GOSUB
10 GOTO 100 : REM SPRINGE ZUM HAUPTPROGRAMM
20 REM *** UNTERPROGRAMM  DISKETTENSTATUS-TEST ***
30 DD=DS : IF DD THEN PRINT "DISKETTEN-FEHLER";DS$
40 RETURN
50 REM *** UNTERPROGRAMM EINGABE J/N ***
60 DO : INPUT "FORTFAHREN (J/N)";A$
70 LOOP UNTIL A$="J" OR A$="N"
80 RETURN
90 REM *** HAUPTPROGRAMM ***
100 DOPEN#2, "DATEN"
110 GOSUB 30 : IF DD THEN DCLOSE#2 : GOSUB 60 : REM FRAGEN
120 IF A$="N" THEN STOP
130 GOTO 100 : REM NOCHMALS VERSUCHEN
```

GRAPHIC

Token: $DE

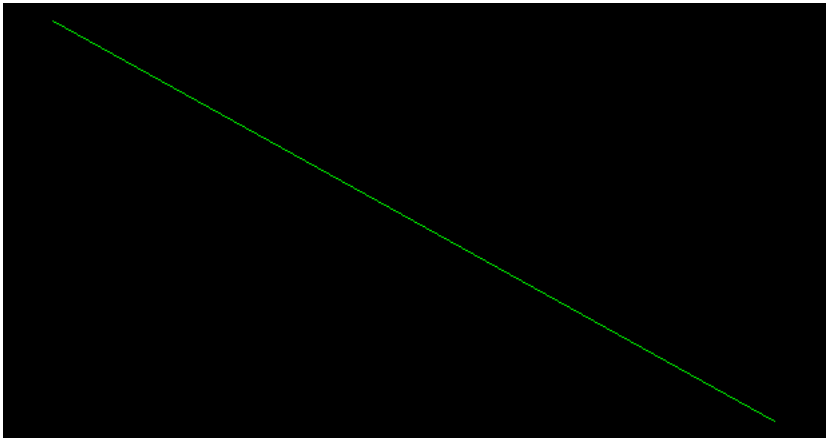
Format: **GRAPHIC CLR**

Zweck: Initialisiert das BASIC-Grafiksystem. Es löscht den Grafikspeicher und den Bildschirm und setzt alle Parameter des Grafikkontextes auf ihre Standardwerte.

Nachdem das Grafiksystem initialsiert wurde, können Befehle wie **LINE**, **PALETTE**, **PEN**, **SCNCLR** und **SCREEN** verwendet werden, um die Parameter des Grafiksystems erneut einzustellen.

Beispiel: Verwendung von **GRAPHIC**:

```
100 REM GRAPHIC
110 GRAPHIC CLR          : REM INITIALISIERUNG
120 SCREEN DEF 1,1,1,2   : REM BILDSCHIRM-MODUS AUF 640 X 400 X 2 SETZEN
130 SCREEN OPEN 1        : REM OEFFNE DIESEN BILDSCHIRM
140 SCREEN SET 1,1       : REM ZEIGE DIESEN BILDSCHIRM
150 PALETTE 1,0,0, 0,0   : REM SCHWARZ
160 PALETTE 1,1,0,15,0   : REM GRUEN
170 SCNCLR 0             : REM FUELLE DEN BILDSCHIRM MIT SCHWARZ
180 PEN 0,1              : REM WAEHLE DEN STIFT AUS
190 LINE 50,50,590,350   : REM ZEICHNE EINE LINIE
200 GETKEY A$            : REM WARTE AUF EINEN TASTENDRUCK
210 SCREEN CLOSE 1       : REM SCHLIESSE BILDSCHIRM UND SETZE PALETTE ZURUECK
```



HEADER

Token: $F1

Format: **HEADER** Diskettenname [**,I** ID] [**,D** Laufwerk] [**,U** Gerät]

Zweck: Wird verwendet um eine Diskette zu formatieren (oder zu löschen).

Diskettenname ist entweder eine String in Anführungszeichen, zum Beispiel **"DATEN"** oder ein String-Ausdruck in Klammern, zum Beispiel **(DN$)**. Die maximale Länge von **Diskettenname** beträgt 16 Zeichen.

ID ist die Disketten-ID. Einige Diskettenlaufwerke verwenden die ID-Kennung, um festzustellen, ob Sie eine Diskette in einem Laufwerk gewechselt haben. Es wird daher empfohlen, dass die ID-Kennung für jede Ihrer Disketten eindeutig ist.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Bei neuen Disketten, die noch nicht formatiert wurden, ist es erforderlich, die Disketten-ID mit dem Parameter **I** anzugeben. Dadurch wird der Formatierungsbefehl auf Vollformatierung umgestellt, bei der die Sektor-IDs geschrieben und alle Inhalte gelöscht werden. Dies nimmt einige Zeit in Anspruch, da jeder Block auf der Platte geschrieben wird.

Wenn der Parameter **I** weggelassen wird, wird eine Schnellformatierung durchgeführt. Dies ist nur möglich, wenn die Diskette bereits formatiert wurde. Bei einer Schnellformatierung wird der neue Datenträgername geschrieben, die Blockzuordnung wird gelöscht und alle Blöcke werden als frei markiert. Die Disketten-ID wird nicht geändert und Blöcke werden nicht überschrieben, so dass der Inhalt mit dem **ERASE**-Befehl in Verbindung mit dem Parameter **R** wiederhergestellt werden kann. Weitere Informationen zu **ERASE** finden Sie auf Seite 97.

Beispiel: Verwendung von **HEADER**:

```
10 REM HEADER
20 REM REM FORMATIERE DISKETTE MIT DEM NAMEN ABENTEUER UND DER ID DK
30 HEADER "ABENTEUER",IDK
40 :
50 REM FORMATIERE DISKETTE IN GERAET 9 MIT DEM NAMEN ZORK-I
60 HEADER "ZORK-I",U9
70 :
80 REM FORMAT. DISK. IN LAUFWERK 1 GERAET 10 MIT NAMEN LABYRINTH
90 HEADER "LABYRINTH",D1,U10

READY.
```

HELP

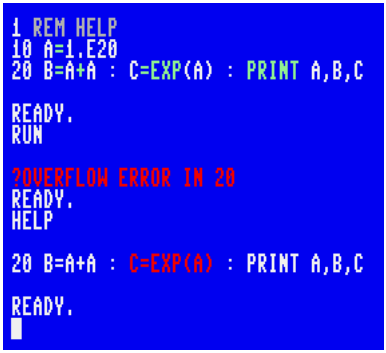
Token: $EA

Format: **HELP**

Zweck: Wenn das BASIC-Programm aufgrund eines Fehlers anhält, kann **HELP** verwendet werden, um weitere Informationen zu erhalten. Die interpretierte Zeile wird aufgelistet, wobei die fehlerhafte Anweisung hervorgehoben oder unterstrichen ist.

Notiz: **HELP** zeigt BASIC-Fehler an. Bei Fehlern im Zusammenhang mit Diskettenoperationen sollte stattdessen die Diskettenstatus-Variable **DS** oder der Diskettenstatus-String **DS$** verwendet werden.

Beispiel: Verwendung von **HELP**:



HEX$

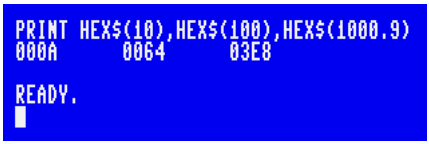
Token: $D2

Format: **HEX$(**numerischer Ausdruck**)**

Zweck: Gibt eine vierstellige hexadezimale Darstellung des Arguments zurück. Das Argument muss im Bereich von 0-65535 liegen, was den Hex-Zahlen $0000-$FFFF entspricht.

Notiz: Wenn reelle Zahlen als Argumente verwendet werden, wird der Nachkommaanteil ignoriert. Mit anderen Worten: Reelle Zahlen werden nicht gerundet.

Beispiel: Verwendung von **HEX$**:



HIGHLIGHT

Token: $FE $3D

Format: **HIGHLIGHT** Farbe [**,** Modus]

Zweck: Legt die für die Hervorhebung verwendeten Farben fest. Für Systemmeldungen, **REM**-Anweisungen und BASIC 65-Schlüsselwörter können unterschiedliche Farben eingestellt werden.

Farbe ist eine der ersten 16 Farben in der aktuellen Palette. Siehe Seite 19 für die Farben in der Standardpalette.

Modus gibt an, wofür die Farbe verwendet werden soll.

- **0** Systemmeldungen (Standardmodus)
- **1 REM**-Anweisungen
- **2** BASIC-Schlüsselwörter

Notiz: Die Farbe der Systemmeldungen wird bei der Anzeige von Fehlermeldungen und bei der Ausgabe von **CHANGE**, **FIND** und **HELP** verwendet. Die Farben für **REM**-Anweisungen und BASIC-Schlüsselwörter werden von **LIST** verwendet.

Beispiel: Verwendung von **HIGHLIGHT**, um die Farbe für BASIC-Schlüsselwörter auf Rot zu setzen:

```
LIST

10 REM *** HALLO WELT ***
20 PRINT "HALLO WELT"

READY.
HIGHLIGHT 8,2

READY.
LIST

10 REM *** HALLO WELT ***
20 PRINT "HALLO WELT"

READY.
```

IF

Token: $8B

Format: **IF** Ausdruck **THEN** wahr-Klausel [**ELSE** falsch-Klausel]

Zweck: **IF** startet eine Anweisung zur bedingten Ausführung.

Ausdruck ist ein logischer oder numerischer Ausdruck. Ein numerischer Ausdruck wird als **FALSE** ("falsch") ausgewertet, wenn der Wert Null ist und **TRUE** ("wahr") für jeden Wert ungleich Null.

wahr-Klausel ist eine oder mehrere Anweisungen, die direkt nach **THEN** in derselben Zeile beginnen. Eine Zeilennummer nach **THEN** führt stattdessen ein **GOTO** zu dieser Zeile aus.

falsch-Klausel ist eine oder mehrere Anweisungen, die direkt nach **ELSE** in der gleichen Zeile beginnen. Eine Zeilennummer nach **ELSE** führt stattdessen ein **GOTO** in dieser Zeile aus.

Notiz: Vor **ELSE** muss ein Doppelpunkt stehen. Nach **ELSE** darf weder ein Doppelpunkt noch ein Zeilenende stehen.

Wenn die **wahr-Klausel** nicht **BEGIN** und **BEND** verwendet, muss **ELSE** in der gleichen Zeile wie **IF** stehen.

Die Standardstruktur **IF ... THEN ... ELSE** ist auf eine einzige Zeile beschränkt. Die **wahr-Klausel** und die **falsch-Klausel** können jedoch auf mehrere Zeilen erweitert werden, indem die zusammengesetzte Anweisung mit **BEGIN** und **BEND** verwendet wird.

Näheres dazu siehe bei **BEGIN** auf Seite 22.

Beispiel: Verwendung von **IF**, **THEN** und **ELSE**:

```
10 REM IF
20 ROT$=CHR$(28):SCHWARZ$=CHR$(144):WEISS$=CHR$(5)
30 INPUT "GEBEN SIE EINE ZAHL EIN";V
40 IF V<0 THEN PRINT ROT$;:ELSE PRINT SCHWARZ$;
50 PRINT V : REM GEBE EINE NEGATIVE ZAHL IN ROT AUS
60 PRINT WEISS$
70 INPUT "PROGRAMM BEENDEN:(J/N)";A$
80 IF A$="J" THEN END
90 IF A$="N" THEN 30: ELSE 70

READY.
```

IMPORT

Token: $DD

Format: **IMPORT** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: **IMPORT** lädt den Inhalt einer Datei (des Typs **SEQ**), die PETSCII-kodierten Text enthält, als BASIC 65-Programm in den Speicher.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Nach dem Laden mit **IMPORT** ist das BASIC 65-Programm bereit für einen Start mit **RUN** oder zur Bearbeitung. Es ist möglich, **IMPORT** zu verwenden, um eine Programmtextdatei von der Diskette in ein bereits im Speicher befindliches Programm einzubinden. Jede aus der Datei gelesene Zeile wird auf die gleiche Weise verarbeitet, als ob sie vom Benutzer mit dem Bildschirmeditor eingegeben wird.

Um ein im Speicher befindliches BASIC 65-Programm, in für **IMPORT** geeigneter Form, auf Diskette auszugeben (**"Export"**), können Sie zum Beispiel folgende Befehlssequenz verwenden:

```
DOPEN#1,"DATEINAME",W:CMD 1:LIST:DCLOSE#1
```

Beispiel: Verwendung von **IMPORT**:

```
IMPORT "LISTING1" : REM LADE DEN TEXTINHALT DER DATEI ALS BASIC-PROGRAMM
```

INFO

Token: $FE $4D

Format: **INFO**

Zweck: **INFO** gibt eine Übersicht einiger Systemwerte und des belegten und freien Speichers des aktuellen BASIC 65-Programms aus.

Auf der linken Seite stehen von oben nach unten:

- **ROM-V** die Versionsnummer des ROMs.
- **SPEED** die aktuelle Systemgeschwindigkeit.
- **BANK4** die mit **MEM** reservierten Speicherbereiche in Bank 4 (siehe dazu **MEM** auf Seite 156).
- **BANK5** die mit **MEM** reservierten Speicherbereiche in Bank 5 (siehe dazu **MEM** auf Seite 156).

Auf der rechten Seite stehen von oben nach unten:

- **PROGRAM** die Speichergröße des aktuell im Speicher befindlichen BASIC 65-Programms.
- **SCALARS** die Speichergröße der aktuell deklarierten skalaren Variablen mit mindestens zwei Buchstaben als Variablenname.
- **STRINGS** die Speichergröße der aktuell deklarierten Strings.
- **ARRAYS** die Speichergröße der aktuell deklarierten Arrays.

Hierbei steht jeweils unter **USED** ("benutzt") die Anzahl der belegten Bytes und unter **FREE** ("frei") die Anzahl der dafür noch im Speicher verfügbaren Bytes.

Beispiel: Verwendung von **INFO**:

```
INFO

INFO                          USED /  FREE
---------------------------------------
ROM-V  V920298   PROGRAM: 4747 / 50290
SPEED  3.5 MHZ   SCALARS:  168 /  1304
BANK4 X-X---XX   STRINGS:  268 / 44968
BANK5 --X---X-   ARRAYS : 9744 / 44968

READY.
```

INPUT

Token: $85

Format: **INPUT** [Eingabeaufforderung <**,** | **;**>] Variable [**,** Variable ...]

Zweck: Gibt eine optionale Eingabeaufforderung und ein Fragezeichen auf dem Bildschirm aus, lässt den Cursor blinken und wartet auf Benutzereingaben über die Tastatur.

Die **Eingabeaufforderung** ist ein optionaler String-Ausdruck, der als Eingabeaufforderung ausgegeben werden soll. Wenn das Trennzeichen zwischen **Eingabeaufforderung** und der **Variablen**-Liste ein Komma ist, wird der Cursor direkt nach dem Prompt gesetzt. Wenn das Trennzeichen ein Semikolon ist, wird stattdessen ein Fragezeichen und ein Leerzeichen an die Eingabeaufforderung angehängt.

Die **Variablen** Liste von einer oder mehreren Variablen, die die Eingabe erhalten.

Die Eingabe wird verarbeitet, nachdem der Benutzer die RETURN-Taste gedrückt hat.

Notiz: Der Benutzer muss darauf achten, den richtigen Typ einzugeben, so dass er mit dem Typ der **Variablen**-Liste übereinstimmt. Außerdem muss die Anzahl der Eingaben mit der Anzahl der Variablen übereinstimmen. Zu viele Eingaben werden ignoriert, während zu wenige Eingaben eine erneute Aufforderung zur Eingabe mit der Aufforderung **??** auslösen. Die Eingabe von nicht numerischen Zeichen für Integer- oder Real-Variablen führt zu einer **?REDO FROM START**-Meldung (dt. etwa "Wiederholung von Anfang an") und die Eingabe wird erneut erwartet. Strings für String-Variablen müssen in doppelte Anführungszeichen (") gesetzt werden, wenn sie Leerzeichen oder Kommas enthalten. Viele Programme, die eine sichere Eingaberoutine benötigen, verwenden **LINE INPUT** und einen eigenen Parser, um Programmfehler durch falsche Benutzereingaben zu vermeiden.

Beispiel: Verwendung von **INPUT**:

```
10 REM INPUT
20 DIM N$(100),A%(100)
30 DO
40 INPUT "NAME, ALTER";NA$,AL%
50 IF NA$="" THEN40
60 IF NA$="END" THENEXIT
70 IF AL% <18 OR AG% >100 THENPRINT "ALTER?" : GOTO40
80 REM TEST OK: INS ARRAY EINTRAGEN
90 N$(N)=NA$ : A%(N)=AG% : N=N+1
100 LOOP UNTIL N=100
110 PRINT "EMPFANGEN";N;" NAMEN"

RUN
NAME, ALTER? MUELLER-LUEDENSCHEIDT, 54
NAME, ALTER? KLOEBNER, 52
NAME, ALTER? END,0
EMPFANGEN 2  NAMEN

READY.
```

INPUT#

Token: $84

Format: **INPUT#** Kanal**,** Variable [**,** Variable ...]

Zweck: Liest einen Datensatz von einem Eingabegerät, zum Beispiel aus einer Datei von Diskette, und ordnet die Daten den Variablen in der Liste zu.

Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **DOPEN** oder **OPEN** angegeben wurde.

Variablen Liste von einer oder mehreren Variablen, die die Eingabe erhalten.

Der Eingabesatz muss mit einem RETURN-Zeichen abgeschlossen werden und darf nicht länger als der Eingabepuffer (160 Zeichen) sein.

Notiz: Der Typ und die Anzahl der Daten in einem Datensatz müssen mit der Variablenliste übereinstimmen. Das Lesen von nicht numerischen Zeichen für Integer- oder Real-Variablen führt zu einem **FILE DATA ERROR** (dt. etwa "Datenfehler in der Datei"). Strings für String-Variablen müssen in Anführungszeichen gesetzt werden, wenn sie Leerzeichen oder Kommas enthalten.

LINE INPUT# kann verwendet werden, um einen ganzen Datensatz in eine einzelne String-Variable zu lesen.

Sequentielle Dateien, die mit **INPUT#** gelesen werden können, können von Programmen mit **PRINT#** oder mit dem Editor des MEGA65 erzeugt werden.

Zum Beispiel:

```
EDIT ON

OK.
10 "CHUCK PEDDLE",1937,"ENTWICKLER DES 6502"
20 "JACK TRAMIEL",1928,"GRUENDER VON COMMODORE"
30 "BILL MENSCH",1945,"HARDWARE"

DSAVE "COMMODORE-LEUTE"

SAVING 0:COMMODORE-LEUTE
READY.
EDIT OFF

READY.
```

Beispiel: Verwendung von **INPUT#**:

```
10 REM INPUT#
20 DIM N$(100),B%(100),S$(100)
30 DOPEN#2,"COMMODORE-LEUTE"     : REM OEFFNE SEQ-DATEI
40 IF DS THENPRINT DS$ : STOP    : REM FEHLER BEIM OEFFNEN
50 FOR I=1 TO 100
60 INPUT#2,N$(I),B%(I),S$(I)
70 IF ST AND 64 THEN100          : REM ENDE DER DATEI
80 IF DS THENPRINT DS$ : GOTO100 : REM DISKETTENFEHLER
90 NEXT I
100 DCLOSE#2
110 PRINT "GELESEN:";I;" EINTRAEGE"
120 FOR J=1 TO I : PRINT N$(J), B%(J), S$(J) : NEXT J

READY.
RUN
GELESEN: 3  EINTRAEGE
CHUCK PEDDLE         1937     ENTWICKLER DES 6502
JACK TRAMIEL         1928     GRUENDER VON COMMODORE
BILL MENSCH          1945     HARDWARE

READY.
```

INSTR

Token: $D4

Format: **INSTR**(Heuhaufen**,** Nadel [**,** Start])

Zweck: Sucht die Position des String-Ausdrucks **Nadel** im String-Ausdruck **Heuhaufen** und gibt den Index des ersten Vorkommens oder Null zurück, wenn es keine Übereinstimmung gibt.

Eine erweiterte Version der Stringsuche mit Muster wird verwendet, wenn das erste Zeichen des Suchstrings ein Pfundzeichen '£' ist. Das Pfundzeichen ist nicht Teil der Suche, ermöglicht aber die Verwendung des **'.'** (Punkt) als Wildcard-Zeichen, das einem beliebigen Zeichen entspricht. Das zweite spezielle Musterzeichen ist das Zeichen **'*'** (Stern). Der Stern in der Suchzeichenfolge zeigt an, dass das vorangestellte Zeichen nie, einmal oder mehrmals vorkommen darf, um als Übereinstimmung betrachtet zu werden.

Das optionale Argument **Start** ist ein ganzzahliger Ausdruck, der die Startposition für die Suche in **Heuhaufen** definiert. Wenn es nicht vorhanden ist, ist es standardmäßig 1.

Beispiel: Verwendung von **INSTR**:

```
PRINT INSTR("ABCDEF","CD")
 3

PRINT INSTR("ABCDEF","XY")
 0

PRINT INSTR("RAIIIN","£A*IN")
 5

PRINT INSTR("ABCDEF","£C.E")
 3

A$="ABC":B$="DEF":C$="CD"

PRINT INSTR(A$+B$,C$)
 3

READY.
```

INT

Token: $B5

Format: **INT(**numerischer Ausdruck**)**

Zweck: Gibt den ganzzahligen Teil des Arguments zurück. Diese Funktion ist **nicht** auf den typischen 16 Bit-Ganzzahlbereich (-32768 bis 32767) beschränkt, da sie reelle Arithmetik verwendet. Der zulässige Bereich wird daher durch die Größe der reellen Mantisse bestimmt, die 32 Bit breit ist (-2147483648 bis 2147483647).

Notiz: Es ist nicht notwendig, die Funktion **INT** für die Zuweisung von reellen Werten zu Integer-Variablen zu verwenden, da diese Umwandlung implizit erfolgt, allerdings nur für den 16 Bit-Bereich.

Beispiel: Verwendung von **INT**:

```
PRINT INT(1.9)
 1

PRINT INT(-3.1)
-3

PRINT INT(100000.5)
-100000

N% =  INT(100000.5)

?ILLEGAL QUANTITY ERROR
READY.
```

JOY

Token: $CF

Format: **JOY**(Port)

Zweck: Gibt den Status des Joysticks für den ausgewählten Controller-Port (1 oder 2) zurück. Bit 7 enthält den Status der Feuertaste. Der Joystick kann in acht Richtungen bewegt werden, die im Uhrzeigersinn nummeriert sind, beginnend mit der obersten Position.

	Links	Mitte	Rechts
Oben	8	1	2
Mitte	7	0	3
Unten	6	5	4

Beispiel: Verwendung von **JOY**:

```
1 REM JOY
10 N=JOY(1)
20 IF N AND 128 THEN PRINT "FEUER! "
30 REM                  N   NO  O   SO  S   SW  W   NW
40 ON N AND 15 GOSUB 100,200,300,400,500,600,700,800
50 GOTO 10
100 PRINT "GEHE NACH NORDEN"     : RETURN
200 PRINT "GEHE NACH NORDOSTEN"  : RETURN
300 PRINT "GEHE NACH OSTEN"      : RETURN
400 PRINT "GEHE NACH SUEDOSTEN"  : RETURN
500 PRINT "GEHE NACH SUEDEN"     : RETURN
600 PRINT "GEHE NACH SUEDWESTEN" : RETURN
700 PRINT "GEHE NACH WESTEN"     : RETURN
800 PRINT "GEHE NACH NORDWESTEN" : RETURN
```

KEY

Token: $F9

Format: **KEY**
KEY <**ON** | **OFF**>
KEY <**LOAD** | **SAVE**> Dateiname
KEY Nummer**,** String

Zweck: Liest den Zustand der Funktionstasten aus. Die Funktionstasten können entweder ihren Tastencode senden, wenn sie gedrückt werden, oder eine der Taste zugewiesene Zeichenfolge. Nach dem Einschalten oder Zurücksetzen ist diese Funktion aktiviert und die Tasten haben ihre Standardbelegung.

KEY listet die aktuellen Zuweisungen auf.

KEY ON schaltet Funktionstasten-Strings ein. Die Tasten senden die zugewiesenen Strings, wenn sie gedrückt werden.

KEY OFF schaltet Funktionstastenstrings aus. Die Tasten senden ihren Zeichencode, wenn sie gedrückt werden.

KEY LOAD lädt die Funktionstastendefinitionen aus einer Datei.

KEY SAVE speichert die Funktionstastendefinitionen in einer Datei.

KEY Nummer, String ordnet den String der Funktionstaste mit der angegebenen Nummer zu.

Standardbelegung der Funktionstasten:

```
KEY
KEY 1,CHR$(27)+"X"
KEY 2,CHR$(27)+"@"
KEY 3,"DIR"+CHR$(13)
KEY 4,"DIR "+CHR$(34)+"*=PRG"+CHR$(34)+CHR$(13)
KEY 5,"▒"
KEY 6,"KEY6"+CHR$(141)
KEY 7,"▒"
KEY 8,"MONITOR"+CHR$(13)
KEY 9,"▒"
KEY 10,"KEY10"+CHR$(141)
KEY 11,"▒"
KEY 12,"KEY12"+CHR$(141)
KEY 13,CHR$(27)+"O"
KEY 14,"▒"+CHR$(27)+"O"
KEY 15,"HELP"+CHR$(13)
KEY 16,"RUN "+CHR$(34)+"*"+CHR$(34)+CHR$(13)

READY.
```

Notiz: Die Summe der Längen aller zugewiesenen Strings darf 240 Zeichen nicht überschreiten. Sonderzeichen wie RETURN oder ANFÜHRUNGSZEICHEN werden über ihre Codes mit der Funktion **CHR$** eingegeben. Weitere Informationen finden Sie unter **CHR$** auf Seite 44.

Beispiel: Verwendung von **KEY**:

```
KEY ON                              : REM AKTIVIERE DIE FUNKTIONSTASTEN
KEY OFF                             : REM DEAKTIVIERE DIE FUNKTIONSTASTEN
KEY                                 : REM LISTE DIE AKTUELLEN ZUWEISUNGEN AUF
KEY 2,"PRINT "+CHR$(14)             : REM WEISE "PRINT PI" DER TASTE F2 ZU
KEY SAVE "MEINE TASTENBELEGUNG"     : REM SPEICHERE AKTUELLE BELEGUNG
KEY LOAD "MEINE TASTENBELEGUNG"     : REM LADE BELEGUNG AUS DER DATEI
```

LEFT$

Token: $C8

Format: **LEFT$**(String, n)

Zweck: Gibt einen String zurück, der die ersten **n** Zeichen des Arguments **String** enthält. Wenn die Länge von **String** gleich oder kleiner als **n** ist, ist die resultierende Zeichenkette identisch mit dem Argument **String**.

String ist ein String-Ausdruck.

n ist ein numerischer Ausdruck (0-255).

Notiz: Leere Strings und Strings der Länge Null sind zulässige Werte.

Beispiel: Verwendung von **LEFT$**:

```
PRINT LEFT$("MEGA65",4)
MEGA

READY.
```

LEN

Token: $C3

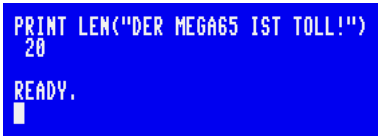
Format: **LEN**(String)

Zweck: Liefert die Länge von **String** zurück.

String ist ein String-Ausdruck.

Notiz: Es gibt kein abschließendes Zeichen, im Gegensatz zu anderen Programmiersprachen wie zum Beispiel bei C, die das NULL-Zeichen verwenden. Die Länge des Strings wird intern in einem zusätzlichen Byte des String-Deskriptors gespeichert.

Beispiel: Verwendung von **LEN**:



LET

Token: $88

Format: [**LET**] Variable = Ausdruck

Zweck: Weist der **Variable** Werte oder Ergebnisse von Ausdrücken zu.

Notiz: Die Anweisung **LET** ist obsolet und wird nicht benötigt. Die Zuweisung von Variablen kann ohne **LET** erfolgen, wurde aber aus Gründen der Abwärtskompatibilität in BASIC 65 beibehalten.

Eine Wertzuweisung ohne **LET** wird von BASIC 65 schneller bearbeitet als dieselbe Wertzuweisung mit **LET**.

Beispiel: Verwendung von **LET**:

```
LET A = 5 : REM LAENGER UND LANGSAMER
A = 5     : REM KUERZER UND SCHNELLER
```

LINE

Token: $E5

Format: **LINE** x-Start**,** y-Start [**,** x-Nächstes1**,** y-Nächstes1 ...]

Zweck: Zeichnet ein Pixel bei (x-Start, y-Start), wenn nur ein Koordinatenpaar angegeben ist.

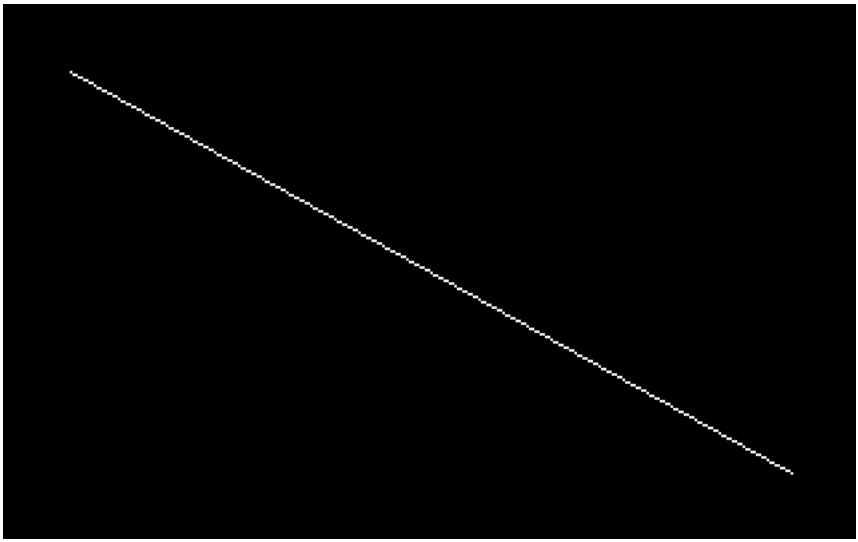
Wenn mehr als ein Koordinatenpaar definiert ist, wird auf dem aktuellen Grafikbildschirm eine Linie von der Koordinate (x-Start, y-Start) zum nächsten Koordinatenpaar gezeichnet und davon jeweils (bis zum letzten Argument) weiter zum nächsten Koordinatenpaar.

Dabei werden alle aktuell definierten Modi und Werte des Grafikkontextes unterstützt.

Beispiel: Verwendung von **LINE**:

```
1 REM LINE
10 SCREEN 320,200,2   :REM OEFFNE BILDSCHIRM MIT AUFLOESUNG: 320 X 200 X 2
20 PEN 1              :REM ZEICHENSTIFT FARBE 1 (WEISS)
30 LINE 25,25,295,175 :REM ZEICHNE LINIE
40 GETKEY A$          :REM WARTE AUF TASTENDRUCK
50 SCREEN CLOSE       :REM SCHLIESSE BILDSCHIRM UND SETZTE PALETTE ZURUECK

READY.
```



LINE INPUT#

Token: $E5 $84

Format: **LINE INPUT#** Kanal**,** Variable [**,** Variable ...]

Zweck: Liest einen Datensatz pro Variable von einem Eingabegerät (z. B. einem Diskettenlaufwerk) und ordnet die gelesenen Daten den Variablen zu. Die Datensätze müssen mit einem **RETURN**-Zeichen abgeschlossen werden, das nicht Bestandteil der String-Variable wird. Daher führt eine leere Zeile, die nur aus dem Zeichen **RETURN** besteht, dazu, dass eine leere Zeichenfolge zugewiesen wird.

Kanal Nummer, die bei einem früheren Aufruf von Befehlen wie **DOPEN** oder **OPEN** verwendet worden ist.

Variable-Liste von einer oder mehreren Variablen, die die Eingabe erhalten.

Notiz: In der Variablenliste können nur String-Variablen oder String-Array-Elemente verwendet werden. Anders als andere INPUT-Befehle interpretiert oder entfernt **LINE INPUT#** keine Anführungszeichen in der Eingabe. Sie werden als Daten akzeptiert, wie alle anderen Zeichen auch.

Die Sätze dürfen nicht länger sein als der Eingabepuffer, der 160 Zeichen beträgt.

Beispiel: Verwendung von **LINE INPUT#**:

```
10 REM LINE INPUT#
20 DIM N$(100)
30 DOPEN#2,"DATEN"
40 FOR I=0 TO 100
50 LINE INPUT#2,N$(I)
60 IF ST=64 THEN90:REM DATEIENDE
70 IF DS THENPRINT DS$ : GOTO90:REM DISKETTENFEHLER
80 NEXT I
90 DCLOSE#2
100 PRINT "GELESEN: ";I;" EINTRAEGE"

READY.
```

LIST

Token: $9B

Format: **LIST** [**P**] [Zeilenbereich]
LIST [**P**] Dateiname [**,U** Gerät]

Zweck: Wird verwendet, um eine Reihe von Zeilen aus dem BASIC-Programm aufzulisten.

Zeilenbereich besteht aus der ersten und/oder letzten aufzulistenden Zeile oder einer einzelnen Zeilennummer. Wenn die erste Zahl weggelassen wird, wird die erste BASIC-Zeile angenommen. Wenn die zweite Zahl weggelassen wird, wird die letzte BASIC-Zeile angenommen.

LIST mit einem Dateinamen als Argument wird verwendet, um ein BASIC-Programm direkt von der Diskette aufzulisten, ohne es zuerst in den Speicher zu laden. Falls kein **Gerät** angegeben wird, wird hierfür der Standardwert 8 verwendet.

Notiz: Der optionale Parameter **P** aktiviert den sogenannten Seitenmodus. Nach der Auflistung von 24 Zeilen wird die Auflistung angehalten und die Eingabeaufforderung **[MORE]** am unteren Rand des Bildschirms angezeigt. Durch Drücken von Q wird der Seitenmodus beendet, während jede andere Taste die Auflistung der nächsten Seite auslöst.

Die Ausgabe von **LIST** kann über **CMD** auf andere Geräte umgeleitet werden.

Die Tasten F9 und F11 oder Ctrl P und Ctrl V blättern eine BASIC-Liste auf dem Bildschirm nach oben oder unten.

Beispiel: Verwendung von **LIST**:

```
LIST 100       :REM LISTE ZEILE 100
LIST 240-350   :REM LISTE ALLE ZEILEN VON 240 BIS 350
LIST 500-      :REM LISTE VON 500 BIS ZUM ENDE
LIST -70       :REM LISTE VOM START BIS ZEILE 70
LIST "DEMO"    :REM LISTE DATEI "DEMO"
LIST P         :REM LISTE DAS AKTUELLE PROGRAMM IM SEITENMODUS
LIST P "DEMO"  :REM LISTE DATEI "DEMO" IM SEITENMODUS
```

LOAD

Token: $93

Format: **LOAD** Dateiname [**,** Gerät [**,** Adressübernahme]]
LOAD "$[Muster=Typ]"[**,D** Laufwerk] [**,U** Gerät]
LOAD "$$[Muster=Typ]"[**,D** Laufwerk] [**,U** Gerät]

/ Dateiname [**,** Gerät [**,** Adressübernahme]]

Zweck: **LOAD** lädt in der ersten Variante eine Datei vom Typ **PRG** in den Speicher in Bank 0, der auch für den BASIC-Programmcode verwendet wird.

Die zweite Variante von **LOAD** lädt ein Verzeichnis in den Speicher, das dann mit **LIST** oder **LISTP** angezeigt werden kann. Es ist wie ein BASIC-Programm aufgebaut, allerdings werden hier die Dateigrößen anstelle von Zeilennummern angezeigt.

Die dritte Variante ist ähnlich wie die zweite, aber die Dateien sind hier aufsteigend nummeriert. Diese Auflistung kann wie ein BASIC-Programm mit den Tasten **F9** oder **F11** geblättert, bearbeitet, aufgelistet, gespeichert oder gedruckt werden.

Ein Filter kann durch Angabe eines Musters oder eines Musters und eines Typs angewendet werden. Der Stern ***** passt auf den Rest des Namens, während das **?** auf jedes einzelne Zeichen passt. Die Typangabe kann ein Zeichen aus (**P**, **S**, **U**, **R**) sein, d. h. **P**rogramm-, **S**equenz-, **U**ser ("Benutzer")- oder **R**EL-Datei.

Dateiname ist entweder ein String in Anführungszeichen, zum Beispiel **"PROG"** oder ein String-Ausdruck.

Die Gerätenummer **Gerät** ist optional. Ist sie nicht vorhanden, wird das Standardgerät 8 verwendet.

Wenn **Adressübernahme** einen Wert ungleich Null hat, wird die Datei an die Adresse geladen, die aus den ersten beiden Bytes der Datei gelesen wird. Andernfalls wird sie an den Anfang des BASIC-Speichers geladen und die Ladeadresse in der Datei wird ignoriert.

Eine häufige Verwendung des Shortcut-Symbols **/** ist das schnelle Laden von **PRG**-Dateien. Um dies zu tun:

1. Lassen Sie ein Festplattenverzeichnis mit **DIR** oder **CATALOG** auf dem Bildschirm anzeigen.
2. Bewegen Sie den Cursor auf die gewünschte Zeile.

3. Geben Sie **/** in die erste Spalte der Zeile ein und drücken Sie die RETURN-Taste.

Nach Drücken der RETURN-Taste wird die aufgelistete Datei in der Zeile mit dem führenden **/** geladen. Zeichen vor und nach den doppelten Anführungszeichen (") des Dateinamens werden ignoriert. Dies gilt nur für **PRG**-Dateien.

Notiz: **LOAD "*"** kann verwendet werden, um das erste Programm **PRG** von **Gerät** zu laden.

LOAD "$" kann verwendet werden, um das Dateiverzeichnis von **Gerät** in den Speicher zu laden. Bei Verwendung von **LOAD "$"** kann anschließend **LIST** verwendet werden, um das Dateiverzeichnis auf dem Bildschirm auszugeben.

LOAD ist in BASIC 65 implementiert, um die Abwärtskompatibilität zu BASIC 2 (Commodore 64) zu gewährleisten.

Das Abkürzungssymbol **/** kann nur im Direktmodus verwendet werden.

Der C64 verwendet standardmäßig **Gerät** 1 für die Datasette, die an dem Kassettenport angeschlossen ist. Der MEGA65 hingegen verwendet standardmäßig **Gerät** 8, das dem internen Diskettenlaufwerk zugewiesen ist. Das bedeutet, dass Sie **LOAD**-Befehlen, die diese Einheit verwenden, nicht den Zusatz **,8** hinzufügen müssen.

Beispiel: Verwendung von **LOAD**:

```
LOAD "FREIZEITPARK" : REM LADE DATEI "FREIZEITPARK" (VON STANDARD-GERAET 8)
LOAD "MEGA-TOOLS",9 : REM LADE DATEI "MEGA-TOOLS" VON GERAET 9
LOAD "*",8,1        : REM LADE DIE ERSTE DATEI VON GERAET 8 AN DIE DORT
                      : REM  ANGEGEBENE ADRESSE IN DEN SPEICHER
LOAD "$"      : REM LADE KOMPLETTES VERZEICHNIS - MIT DATEIGROESSEN
LOAD "$$"     : REM LADE KOMPLETTES VERZEICHNIS - ALS LISTING ZUM SCROLLEN
LOAD "$$X*=P" : REM VERZEICHNIS DER PRG-DATEIEN, DIE MIT 'X' BEGINNEN
```

LOADIFF

Token: $FE $43

Format: **LOADIFF** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: **LOADIFF** lädt eine IFF-Datei in den Grafikspeicher. Das IFF (Interchange File Format) wird von vielen verschiedenen Anwendungen und Betriebssystemen unterstützt. **LOADIFF** geht davon aus, dass die Dateien Bitplane-Grafiken enthalten, die in den Grafikspeicher des MEGA65 passen.

Unterstützte Auflösungen sind:

Breite	Höhe	Bitplanes	Farben	Speicher
320	200	max. 8	max. 256	max. 64 KB
640	200	max. 8	max. 256	max. 128 KB
320	400	max. 8	max. 256	max. 128 KB
640	400	max. 4	max. 16	max. 128 KB

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Beispiel: Verwendung von **LOADIFF**:

```
10 REM LOADIFF
20 BANK 128 : SCNCLR
30 REM ZEIGE BILDER IN 320 X 200 X 7 AUFLOESUNG
40 GRAPHIC CLR : SCREEN DEF 0,0,0,7 : SCREEN OPEN 0 : SCREEN SET 0,0
50 FOR I = 1 TO 7 : READ F$
60 LOADIFF(F$+".IFF") : SLEEP 4 : NEXT I
70 DATA URLAUB,HIMBEERE,KIRSCHE,ERDBEERE,SCHIFF,SONNENUNTERGANG
80 SCREEN CLOSE 0
90 PALETTE RESTORE
```

LOCK

Token: $FE $50

Format: **LOCK** Dateiname/Muster [**,D** Laufwerk] [**,U** Gerät]

Zweck: **LOCK** wird verwendet, um Dateien zu sperren. Die angegebene Datei oder ein Satz von Dateien, die dem Muster entsprechen, wird/werden gesperrt und kann/können nicht mit den Befehlen **DELETE**, **ERASE** oder **SCRATCH** gelöscht werden.

Der Befehl **UNLOCK** hebt die Sperre auf.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Im Direktmodus wird die Anzahl der gesperrten Dateien ausgegeben. Die vorletzte Zahl in der Meldung enthält die Anzahl der gesperrten Dateien.

Beispiel: Verwendung von **LOCK**

```
LOCK "DRM",U9 : REM SPERRE DATEI "LOCK" AUF GERAET 9
03,FILES LOCKED,01,00
READY.

LOCK "BS*"    : REM SPERRE ALLE DATEIEN, DIE MIT "BS" BEGINNEN
03,FILES LOCKED,03,00
READY.
```

LOG

Token: $BC

Format: **LOG(**numerischer Ausdruck**)**

Zweck: Berechnet den Wert des natürlichen Logarithmus des Arguments. Der natürliche Logarithmus (auf Taschenrechner oft als LN bezeichnet) verwendet die Eulersche Zahl (**2.71828183**) als Basis, nicht 10, die normalerweise in LOG-Funktionen auf Taschenrechnern verwendet wird.

Notiz: Die Log-Funktion zur Basis 10 kann berechnet werden indem man das Ergebnis durch log(10) dividiert.

Beispiel: Verwendung von **LOG**:

```
PRINT LOG(2.3)
 0.83290912

READY.

PRINT LOG(0)

?ILLEGAL QUANTITY ERROR
READY.

PRINT LOG(4)
 1.3862944

READY.

PRINT LOG(100) / LOG(10)
 2

READY.
```

LOG10

Token: $CE $08

Format: **LOG10**(numerischer Ausdruck)

Zweck: Berechnet den Wert des dezimalen Logarithmus des Arguments. Der dezimale Logarithmus verwendet 10 als Basis.

Beispiel: Verwendung von **LOG10**:

```
PRINT LOG10(100)
 2

READY.

PRINT LOG10(0)

?ILLEGAL QUANTITY ERROR
READY.

PRINT LOG10(5)
 0.69897001

READY.

PRINT LOG10(100);LOG10(10);LOG10(0.1);LOG10(0.01)
 2  1 -1 -2

READY.
```

LOOP

Token: $EC

Notiz: **LOOP** ist ein Schlüsselwort, das nur in Kombination mit **DO** verwendet wird.

Näheres siehe bei **DO** auf Seite 76.

LPEN

Token: $CE $04

Format: **LPEN**(Koordinate)

Zweck: Diese Funktion erfordert die Verwendung eines Röhrenmonitors (oder Röhrenfernsehers) und eines Lightpen (dt. "Lichtgriffel"), der an Port 1 angeschlossen sein muss. Ein Lightpen funktioniert nicht mit einem LCD- oder LED-Bildschirm.

LPEN(0) gibt die x-Position des Lightpen zurück, der Bereich ist 60-320.

LPEN(1) gibt die y-Position des Lightpen zurück, der Bereich liegt bei 50-250.

Notiz: Die x-Auflösung beträgt zwei Pixel, daher liefert **LPEN(0)** nur gerade Zahlen. Eine helle Hintergrundfarbe ist erforderlich, um den Lightpen auszulösen. Die Anweisung **COLLISION** kann verwendet werden, um einen Interrupt-Handler zu aktivieren.

Beispiel: Verwendung von **LPEN**:

```
PRINT LPEN(0),LPEN(1) : REM GIB DIE KOORDINATEN DES LIGHTPEN AUS
 0         0

READY.
```

MEM

Token: $FE $23

Format: **MEM** Maske4, Maske5

Zweck: **Maske4** und **Maske5** sind Byte-Werte, die als Maske von 8 Bits interpretiert werden. Jedes auf 1 gesetzte Bit reserviert ein 8 KB-Segment des Speichers in Bank 4 (für das erste Argument) und in Bank 5 (für das zweite Argument).

Bit	Speichersegment
0	$0000 - $1FFF
1	$2000 - $3FFF
2	$4000 - $5FFF
3	$6000 - $7FFF
4	$8000 - $9FFF
5	$A000 - $BFFF
6	$C000 - $DFFF
7	$E000 - $FFFF

Notiz: Nach der Reservierung von Speicher mit **MEM** verwendet die Grafikbibliothek die reservierten Bereiche nicht, so dass sie für andere Zwecke genutzt werden können. Der Zugriff auf Bank 4 und 5 ist mit den Befehlen **PEEK, PEEKW, POKE, POKEW** und **EDMA** möglich.

Wenn ein Grafikbildschirm nicht geöffnet werden kann, weil der verbleibende Speicher nicht ausreicht, bricht das Programm mit einer **?OUT OF MEMORY ERROR**-Fehlermeldung (dt. "Speicherplatzmangel") ab.

MEM 0,0 gibt die reservierten Speichersegmente wieder frei.

Beispiel: Verwendung von **MEM**:

```
10 REM MEM
20 MEM 1,3          : REM RESERVIERT $40000 - $41FFF AND $50000 - $53FFF
30 SCREEN 320,200,2 : REM BILDSCHIRM VERWENDET DIE RESERVIERTEN SEGMENTE NICHT
40 EDMA 3,$2000,0,$4000 : REM FUELLE SPEICHER-SEGMENTE MIT NULLEN
50 GETKEY A$            : REM WARTE AUF TATSENDRUCK
60 SCREEN CLOSE         : REM SCHLIESSE GRAFIKBILDSCSHIRM
70 MEM 0,0              : REM GIBT SPEICHER-SEGMENTE WIEDER FREI

READY.
```

MERGE

Token: $E6

Format: **MERGE** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: **MERGE** lädt eine BASIC-Programmdatei von der Festplatte und fügt sie an das Programm im Speicher an.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Die Ladeadresse, die in den ersten beiden Bytes der Datei gespeichert ist, wird ignoriert. Das geladene Programm ersetzt nicht ein Programm im Speicher (wie es **DLOAD** tut), sondern wird an ein Programm im Speicher angehängt. Nach dem Laden ist das Programm neu verlinkt und kann ausgeführt oder bearbeitet werden.

Es liegt in der Verantwortung des Benutzers, sicherzustellen, dass es keine Zeilenkonflikte zwischen dem Programm im Speicher und dem zusammengefügten Programm gibt. Die erste Zeilennummer des zusammengefügten Programms muss größer sein als die letzte Zeilennummer des Programms im Speicher.

Beispiel: Verwendung von **MERGE**:

```
100 REM PROGRAMM 2
110 PRINT "PROGRAMM 2"
120 END

DSAVE"ZWEI"

SAVING 0:ZWEI
READY.
NEW

READY.
10 REM PROGRAMM 1
20 PRINT "PROGRAMM 1"
LIST

10 REM PROGRAMM 1
20 PRINT "PROGRAMM 1"

READY.
MERGE"ZWEI"

LOADING 0:ZWEI
READY.
LIST

10 REM PROGRAMM 1
20 PRINT "PROGRAMM 1"
100 REM PROGRAMM 2
110 PRINT "PROGRAMM 2"
120 END

READY.
```

MID$

Token: $CA

Format: **MID$**(String**,** Start**,** Länge**)**
MID$(String**,** Start**,** Länge**)** = String-Ausdruck

Zweck: **MID$** kann entweder als Funktion verwendet werden, die eine Zeichenkette zurückgibt, oder als Anweisung zum Einfügen von Teilzeichenketten in eine bestehende Zeichenkette.

String ist ein String-Ausdruck.

Start Startindex (0-255).

Länge Länge der Teilzeichenkette (0-255).

Notiz: Leere Zeichenketten und Längen von Null sind zulässige Werte.

Beispiel: Verwendung von **MID$**:

```
A$ = "MEGA65"

READY.
PRINT MID$(A$,3,2)
GA

READY.
MID$(A$,5,1)="7"

READY.
PRINT A$
MEGA75

READY.
```

MKDIR

Token: $FE $51

Format: **MKDIR** Verzeichnisname **,L** Größe [**,U** Gerät]

Zweck: **MKDIR** erstellt ein Unterverzeichnis auf einer Diskette oder einem D81-Diskettenimage.

Verzeichnisname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

MKDIR kann nur auf Geräten verwendet werden, die von CBDOS verwaltet werden. Dies sind das interne Diskettenlaufwerk und SD-Karten-Images vom Typ **D81**. Der Befehl kann nicht für externe Laufwerke verwendet werden, die an den seriellen IEC-Bus angeschlossen sind.

Der Parameter **Größe** gibt die Anzahl der Spuren an, die für das Unterverzeichnis reserviert werden sollen, wobei eine Spur = 40 Sektoren à 256 Byte ist. Die erste Spur des reservierten Bereichs wird als Verzeichnisspur für das Unterverzeichnis verwendet.

Die Mindestgröße beträgt 3 Spuren, die Höchstgröße 38 Spuren. Auf der Diskette (D81-Image) muss ein zusammenhängender Bereich mit leeren Spuren vorhanden sein, der groß genug für die Erstellung des Unterverzeichnisses ist. Ist ein solcher Bereich nicht vorhanden, wird die Fehlermeldung **DISK FULL** ausgegeben.

Es können mehrere Unterverzeichnisse angelegt werden, solange genügend leere Spuren vorhanden sind.

Nach erfolgreicher Erstellung des Unterverzeichnisses wird ein automatisches **CHDIR** in dieses Unterverzeichnis durchgeführt.

CHDIR "/" wechselt zurück in das Stammverzeichnis.

Beispiel: Verwendung von **MKDIR**:

```
MKDIR "VERZEICHNIS",L5 : REM ERZEUGE UNTERVERZEICHNIS MIT 5 SPUREN

READY.
DIR
   0 "TEST           " 89 2D
 200 "VERZEICHNIS"      CBM
2960 BLOCKS FREE

READY.
```

MOD

Token: $NN

Format: **MOD**(Dividend**,** Divisor**)**

Zweck: Die Funktion **MOD** gibt den Rest der Division zurück.

Notiz: In anderen Programmiersprachen, wie zum Beispiel C, ist diese Funktion als Operator (%) implementiert. In BASIC 65 ist sie als Funktion implementiert.

Beispiel: Verwendung von **MOD**:

```
FOR I = 0 TO 8 : PRINT MOD(I,4); : NEXT I
 0  1  2  3  0  1  2  3  0
READY.
```

MONITOR

Token: $FA

Format: **MONITOR**

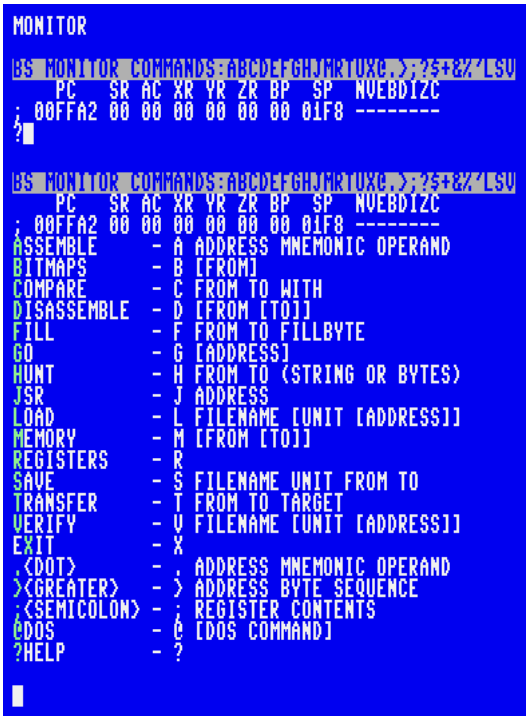
Zweck: Ruft das Monitorprogramm für Maschinensprache auf, das hauptsächlich zur Fehlersuche verwendet wird.

Notiz: Die Verwendung des **MONITOR** erfordert Kenntnisse der CSG4510- / 6502- / 6510-CPU, der verwendeten Assemblersprache und ihrer Architekturen.

Zum Verlassen des Monitors drücken Sie **X**.

Der Hilfetext kann entweder mit **?** oder **H** angezeigt werden.

Beispiel: Verwendung von **MONITOR**:



MOUNT

Token: $FE $49

Format: **MOUNT** Dateiname [**,U** Gerät]

Zweck: Bindet ein Diskette-Image vom Typ **D81** von der SD-Karte als Gerät 8 (Standard) oder Gerät 9 ein.

Wenn kein Argument angegeben wird, weist **MOUNT** das eingebaute Diskettenlaufwerk des MEGA65 dem Gerät 8 zu.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Der **MOUNT**-Befehl kann entweder im Direktmodus oder in einem Programm verwendet werden. Er sucht die angegebene Datei auf der SD-Karte und bindet (mounted) sie, wie gewünscht, als Gerät 8 oder 9 ein. Nach dem Einbinden kann das Diskettenimage wie gewohnt mit allen DOS-Befehlen verwendet werden.

Beispiel: Verwendung von **MOUNT**:

```
MOUNT "SPIELE.D81"   : REM BINDET "SPIELE.D81" ALS GERAET 8 EIN
MOUNT "BASIC.D81",U9 : REM BINDET "BASIC.D81" ALS GERAET 9 EIN
MOUNT (FI$),U(UN%)   : REM BINDET DATEI FI$ ALS GERAET UN% EIN
```

MOUSE

Token: $FE $3E

Format: **MOUSE ON** [{**,** Port**,** Sprite**,** Position}]
MOUSE OFF

Zweck: **MOUSE ON** aktiviert den Maustreiber und verknüpft die Maus am angegebenen Port mit dem Mauszeiger-Sprite.

Port Mausport 1, 2 (Voreinstellung) oder 3 (beide).

Sprite Sprite-Nummer für Mauszeiger (Voreinstellung 0).

Position Anfangsposition der Maus (x,y).

MOUSE OFF deaktiviert den Maustreiber und gibt das zugehörige Sprite frei.

Notiz: Der "Hot Spot" (dt. "Brennpunkt") des Mauszeigers ist das obere linke Pixel des Sprites. Das bedeutet, dass sich die Koordinatenangaben auf dieses obere linke Pixel beziehen.

Notiz: Verwendung von **MOUSE**:

```
10 REM LADE DATEN IN SPRITE NUMMER 0, BEVOR ES VERWENDET WIRD
20 MOUSE ON, 1 : REM AKTIVIERE MAUS MIT SPRITE NUMMER 0
30 MOUSE OFF   : REM DEAKTIVIERE MAUS

READY.
```

MOVSPR

Token: $FE $06

Format: **MOVSPR** Nummer**,** Position
MOVSPR Nummer**,** Startposition **TO** Endposition**,** Geschwindigkeit

Zweck: Bewegt ein Sprite auf dem Bildschirm. Jedes Argument von **Position** besteht aus zwei 16 Bit-Werten, die entweder eine absolute Koordinate, eine relative Koordinate, einen Winkel oder eine Geschwindigkeit angeben. Der Wertetyp wird durch ein Präfix bestimmt:

- **+Wert** relative Koordinate: positiver Offset.
- **-Wert** relative Koordinate: negativer Offset.
- **#Wert** Geschwindigkeit.

Wenn kein Präfix angegeben wird, wird die absolute Koordinate oder der absolute Winkel verwendet.

Daher kann das Argument Position entweder verwendet werden:

- um das Sprite auf eine absolute Position auf dem Bildschirm setzen.
- um eine Verschiebung relativ zur aktuellen Position anzugeben.
- um eine Relativbewegung von einer bestimmten Position aus auslösen.
- um eine Bewegung mit einem Winkel und einer Geschwindigkeit, ausgehend von der aktuellen Position, zu beschreiben.

MOVSPR Nummer, Position wird verwendet, um das Sprite sofort an die Position zu setzen oder, im Falle eines **Winkel#Geschwindigkeit**-Arguments, seine weitere Bewegung zu beschreiben.

MOVSPR Nummer, Startposition TO Zielposition, Geschwindigkeit platziert das Sprite an der Startposition, definiert die Zielposition und die Geschwindigkeit der Bewegung. Das Sprite wird an die Startposition gesetzt und bewegt sich von dort mit der angegebenen Geschwindigkeit in einer geraden Linie zum Ziel. Die Koordinaten müssen absolut oder relativ sein. Die Bewegung wird durch den BASIC-Interrupt-Handler gesteuert und geschieht gleichzeitig mit der Programmausführung.

Nummer Spritenummer (0-7).

Position x,y | xrel,y | x,yrel | xrel,yrel | Winkel#Geschwindigkeit.

x absolute x-Bildschirmkoordinate in Pixel.

y absolute y-Bildschirmkoordinate in Pixel.

xrel relative x-Bildschirmkoordinate in Pixel.

yrel relative y-Bildschirmkoordinate in Pixel.

Winkel Kompassrichtung für die Sprite-Bewegung [Grad].
0: oben, 90: rechts, 180: unten, 270: links, 45 oben rechts, usw.

Geschwindigkeit ist die Bewegungsgeschwindigkeit des Sprites, angegeben als Fließkommazahl im Bereich **`0.0`** - **`127.0`**. Die Einheit ist hierbei "Pixel pro Frame". Das heißt, dass ein Geschwindigkeitswert von **`1.0`** das Sprite im PAL-Modus um 50 Pixel pro Sekunde bewegt.

Beachten Sie, dass die PAL-Norm eine Bildwiederholfrequenz von 50 und die NTSC-Norm von 60 Hz hat. Dies bedeutet, dass bei PAL 50 Frames (Einzelbilder) pro Sekunde erzeugt werden, während es bei NTSC 60 Frames pro Sekunde sind.

Notiz: Der "Hot Spot" (dt. "Brennpunkt") des Sprites ist das obere linke Pixel des Sprites. Das bedeutet, dass sich die Koordinatenangaben auf dieses obere linke Pixel beziehen.

Wenn Sie **Geschwindigkeit** entsprechend dem BASIC 7-Befehl **MOVSPR** (vom Commodore 128) verwenden wollen, beachten Sie, dass Sie die bei BASIC 7 erwarteten **Geschwindigkeit**-Parameter (0-15) für BASIC 65 jeweils **mit 8 multiplizieren** müssen, um eine entsprechende Spritegeschwindigkeit zu erhalten.

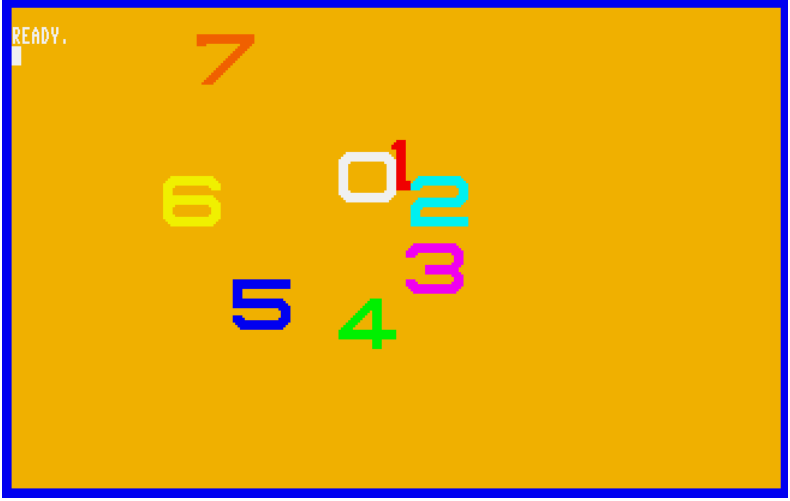
BASIC 7	BASIC 65
0	0.0
1	8.0
2	16.0
3	24.0
4	32.0
5	40.0
6	48.0
7	56.0
8	64.0
9	72.0
10	80.0
11	88.0
12	96.0
13	104.0
14	112.0
15	120.0

Beispiel: Verwendung von **MOVSPR**:

```
LIST

1 REM MOVSPR
10 CLR : SCNCLR : SPRITECLR : BACKGROUND 18
20 BLOAD "DEMOSPRITES1",B0,P1536
30 FOR I=0 TO 7: C=I+1 : SP=0.07*(I+1)
40 MOVSPR I,160,120
50 MOVSPR I,45*I#SP
60 SPRITE I,1,C,,0,0
70 NEXT
80 SLEEP 3
90 FOR I=0 TO 7 : MOVSPR I,0#0 : NEXT

READY.
RUN
```



NEW

Token: $A2

Format: **NEW**
NEW RESTORE

Zweck: **NEW** "löscht" das gegenwärtig im Speicher befindliche BASIC-Programm sowie alle Variablen. **NEW** setzt alle BASIC-Parameter auf ihre Standardwerte zurück.

Da **NEW** Parameter und Zeiger zurücksetzt (aber den Adressbereich eines BASIC-Programms, das sich im Speicher befand, nicht überschreibt), kann das Programm wiederhergestellt werden. Wenn nach **NEW** keine **LOAD**-Operationen oder Bearbeitungen durchgeführt worden sind, kann das Programm mit **NEW RESTORE** wiederhergestellt werden.

Beispiel: Verwendung von **NEW**:

```
NEW         : REM SETZE BASIC ZURUECK
NEW RESTORE : REM VERSUCHE DAS MIT NEW "GELOESCHTE" PROGRAMM WIEDERHERZUSTELLEN
```

NEXT

Token: $82

Notiz: **NEXT** ist ein Schlüsselwort, das nur in Kombination mit **FOR** verwendet wird.

Näheres siehe bei **FOR** auf Seite 110.

NOT

Token: $A8

Format: **NOT** Operand

Zweck: **NOT** führt eine bitweise logische NICHT-Operation an einem 16 Bit-Wert durch.

Ganzzahlige Operanden werden so verwendet, wie sie sind, während reelle Operanden in eine vorzeichenbehaftete 16 Bit-Ganzzahl umgewandelt werden (mit Verlust der Genauigkeit).

Logische Operanden werden in eine 16 Bit-Ganzzahl umgewandelt, wobei $FFFF (dezimal -1) für TRUE (wahr) und $0000 (dezimal 0) für FALSE (falsch) verwendet wird.

Ausdruck	Ergebnis
NOT 0	1
NOT 1	0

Notiz: Das Ergebnis ist vom Typ Integer (Ganzzahl).

Beispiel: Verwendung von **NOT**:

```
PRINT NOT 3
-4

READY.

PRINT NOT 64
-65

READY.
```

In den meisten Fällen wird **NOT** in **IF**-Anweisungen verwendet.

```
10 REM BEISPIEL 2 - NOT
20 REM PRUEFE, OB C KLEINER ALS 256 UND GROESSER GLEICH 0 IST
30 :
40 OK = C < 256 AND C >= 0
50 :
60 IF (NOT OK) THEN PRINT "KEIN GUELTIGER BYTE-WERT"
70 :

READY.
```

OFF

Token: $FE $24

Format: Schlüsselwort **OFF**

Zweck: **OFF** ist ein sekundäres Schlüsselwort, das in Kombination mit primären Schlüsselwörtern wie **COLOR**, **KEY** und **MOUSE** verwendet wird.

Notiz: **OFF** kann nicht eigenständig verwendet werden.

Beispiel: Verwendung von **OFF**:

ON

Token: $91

Format: **ON** Ausdruck **GOSUB** Zeilennummer [**,** Zeilennummer ...]
ON Ausdruck **GOTO** Zeilennummer [**,** Zeilennummer ...]
Schlüsselwort **ON**

Zweck: **ON** ruft entweder eine berechnete **GOSUB**- oder **GOTO**-Anweisung auf. Je nach Ergebnis des Ausdrucks wird das Ziel für **GOSUB** und **GOTO** aus der Tabelle der Zeilenadressen am Ende der Anweisung ausgewählt.

Bei der Verwendung als sekundäres Schlüsselwort wird **ON** in Kombination mit primären Schlüsselwörtern wie **COLOR**, **KEY** und **MOUSE** verwendet.

Ausdruck ist ein positiver numerischer Wert. Reelle Werte werden in Ganzzahlen umgewandelt (unter Verlust der Genauigkeit). Logische Operanden werden in eine 16 Bit-Ganzzahl umgewandelt, wobei $FFFF (dezimal -1) für TRUE (wahr) und $0000 (dezimal 0) für FALSE (falsch) verwendet wird.

Notiz: Negative Werte für **Ausdruck** halten das Programm mit einer Fehlermeldung an. Die Liste der **Zeilennummer**n gibt die Ziele für die Werte 1, 2, 3, usw. an.

Ein Ausdrucksergebnis von Null oder ein Ergebnis, das größer als die Anzahl der Zielzeilen ist, bewirkt nichts und das Programm setzt die Ausführung mit der nächsten Anweisung fort.

Beispiel: Verwendung von **ON**:

```
100 REM ON
110 KEY   ON :REM AKTIVIERE FUNKTIONSTASTEN-STRINGS
120 MOUSE ON :REM AKTIVIERE MOUSETREIBER
130 :
140 :
150 N = JOY(1):IF N AND 128 THEN PRINT "FEUER! ";
160 REM               N   NO  O   SO  S   SW  W   NW
170 ON N AND 15 GOSUB 190,200,210,220,230,240,250,260
180 GOTO 140
190 PRINT "GEHE NACH NORDEN"     : RETURN
200 PRINT "GEHE NACH NORDOSTEN"  : RETURN
210 PRINT "GEHE NACH OSTEN"      : RETURN
220 PRINT "GEHE NACH SUEDOSTEN"  : RETURN
230 PRINT "GEHE NACH SUEDEN"     : RETURN
240 PRINT "GEHE NACH SUEDWESTEN" : RETURN
250 PRINT "GEHE NACH WESTEN"     : RETURN
260 PRINT "GEHE NACH NORDWESTEN" : RETURN

READY.
```

OPEN

Token: $9F

Format: **OPEN** Kanal**,** Primäradresse [**,** Sekundäradresse [**,** Dateiname]]

Zweck: Öffnet einen Eingangs-/Ausgangskanal für ein Gerät.

Kanal Kanalnummer, wobei bei:

- **1 <= Kanal <= 127** der Zeilenabschluss CR ist.
- **128 <= Kanal <= 255** der Zeilenabschluss CR LF ist.

Primäradresse Gerätenummer. Bei IEC-Geräten ist die Gerätenummer die Primäradresse. Folgende Werte für die Primäradresse sind möglich:

Gerätenummer	Gerät
0	Tastatur
1	Systemvoreinstellung
2	Serieller RS232-Anschluss
3	Bildschirm
4-7	IEC-Drucker und Plotter
8-31	IEC-Diskettenlaufwerke

Die **Sekundäradresse** hat einige reservierte Werte für IEC-Diskettenlaufwerke: 0: Laden, 1: Speichern, 15: Befehlskanal. Die Werte 2-14 können für Diskettendateien verwendet werden.

Dateiname ist entweder ein String in Anführungszeichen, zum Beispiel **"DATA"** oder ein String-Ausdruck. Die Syntax unterscheidet sich von **DOPEN#**, da der **Dateiname** für **OPEN** alle Dateiattribute enthält, zum Beispiel: **"0:DATEN,S,W"**.

Notiz: Für IEC-Diskettenlaufwerke wird die Verwendung von **DOPEN#** empfohlen.

Wenn das erste Zeichen des Dateinamens ein At-Zeichen '@' ist, wird es als "Speichern und Ersetzen"-Operation interpretiert. Es wird nicht empfohlen, diese Option auf den Laufwerken 1541 und 1571 zu verwenden, da sie einen "Speichern und Ersetzen"-Fehler in ihrem DOS enthalten.

Notiz: Verwendung von **OPEN**:

```
10 REM OPEN
20 :
30 OPEN 4,4   :REM OEFFNE DRUCKER
40 CMD 4      :REM LEITE STANDARDAUSGABE AUF GERAET 4 UM
50 LIST       :REM GIB LISTING AUF GERAET 4 (DRUCKER) AUS
60 CLOSE 4
70 :
80 OPEN 3,8,3,"0:NUTZERDATEI,U"
90 OPEN 2,9,2,"0:DATEN,S,W"

READY.
```

OR

Token: $B0

Format: Operand **OR** Operand

Zweck: Führt eine bitweise logische ODER-Operation an zwei 16 Bit-Werten durch. Ganzzahlige Operanden werden so verwendet, wie sie sind. Reelle Operanden werden in eine vorzeichenbehaftete 16 Bit-Ganzzahl umgewandelt (mit Verlust der Genauigkeit). Logische Operanden werden unter Verwendung von $FFFF (dezimal -1) für TRUE (wahr) und $0000 (dezimal 0) für FALSE (falsch) in eine 16 Bit-Ganzzahl umgewandelt.

Ausdruck	Ergebnis
0 OR 0	0
0 OR 1	1
1 OR 0	1
1 OR 1	1

Notiz: Das Ergebnis ist vom Typ Integer. Wenn das Ergebnis in einem logischen Kontext verwendet wird, wird der Wert 0 als FALSE und alle anderen Werte ungleich Null als TRUE angesehen.

Beispiel: Verwendung von **OR**:

```
PRINT 1 OR 3
 3

PRINT 128 OR 64
 192
```

In den meisten Fällen wird **OR** in **IF**-Anweisungen verwendet.

```
10 REM OR
20 INPUT C
30 :
40 REM PRUEFE, OB C KLEINER ALS 0 ODER GROESSER ALS 255 IST
50 :
60 IF (C < 0 OR C > 255) THEN PRINT "KEIN BYTE-WERT"

READY.
```

PAINT

Token: $DF

Format: **PAINT** x, y, Modus [, Farbe]

Zweck: Füllt einen eingeschlossenen Grafikbereich in der aktuellen Stiftfarbe aus.

x, y ist ein Koordinatenpaar, das innerhalb des zu füllenden Bereichs liegen muss.

Modus gibt den Füllmodus an:

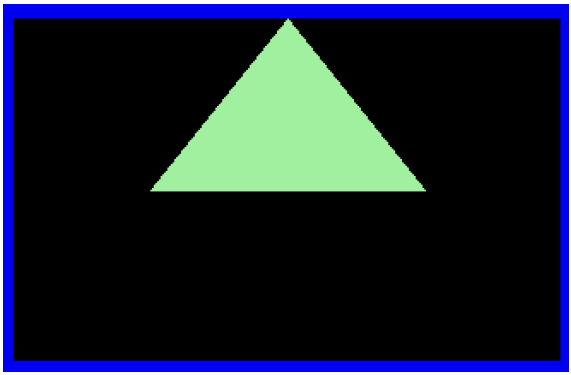
- **0** Die Farbe des Pixels (x,y) definiert die Farbe, die durch die Stiftfarbe ersetzt wird.
- **1 Farbe** definiert die Farbe des **Rahmens** der Region, die mit dem aktuellen Zeichenstift gefüllt werden soll.
- **2** Füllt die Region, die mit dem Pixel (x,y) verbunden ist.

Farbe ist der Farbindex für Modus 1 an.

Beispiel: Verwendung von **PAINT**:

```
100 REM ELLIPSE
110 SCREEN 320,200,2     :REM OEFFNE GRAFIKSCHIRM (320X200)
120 PALETTE 0,1,10,15,10 :REM SETZE FARBE 1 AUF HELLGRUEN
130 PEN 1                :REM VERWENDE STIFT 1
140 LINE 160,0,240,100   :REM ERSTE LINIE
150 LINE 240,100,80,100  :REM ZWEITE LINIE
160 LINE 80,100,160,0    :REM DRITTE LINIE
170 PAINT 160,10         :REM FUELLE DAS DREIECK MIT DER AKTUELLEN ZEICHENFARBE
180 GETKEY A$            :REM WARTE AUF TASTENDRUCK
190 SCREEN CLOSE         :REM SCHLIESSE GRAFIKSCHIRM

READY.
```



PALETTE

Token: $FE $34

Format: **PALETTE** Bildschirmnummer**,** Farbe**,** Rot**,** Grün**,** Blau
PALETTE COLOR Farbe**,** Rot**,** Grün**,** Blau
PALETTE RESTORE

Zweck: **PALETTE** kann verwendet werden, um einen Eintrag der Systemfarbpalette oder die Palette eines Bildschirms zu ändern.

PALETTE RESTORE setzt die Systempalette auf die Standardwerte zurück.

Bildschirmnummer ist die Bildschirmnummer (0-3).

COLOR Schlüsselwort zum Ändern der Systempalette.

Farbe Index zur Palette (0-255).

Rot rote Intensität (0-15).

Grün grüne Intensität (0-15).

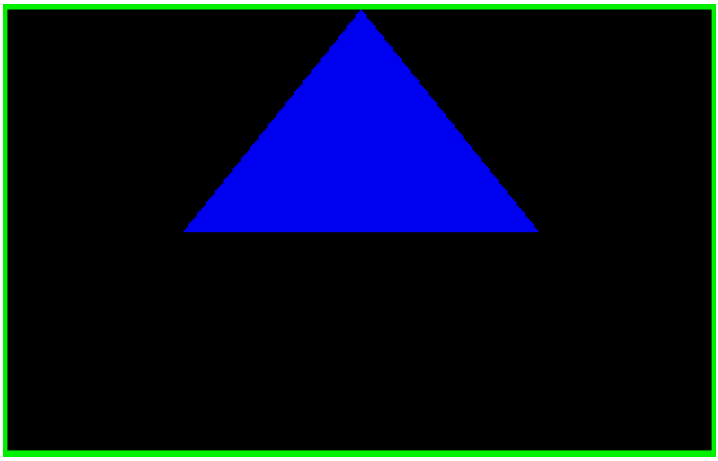
Blau blaue Intensität (0-15).

Beispiel: Verwendung von **PALETTE**:

```
10 REM SYSTEMFARBINDEX AENDERN
20 REM --- INDEX  9 (BRAUN) ZU (DUNKELBLAU)
30 PALETTE COLOR 9,0,0,7
```

```
10 GRAPHIC CLR                : REM INITIALISIERE GRAFIK
20 SCREEN DEF 1,0,0,2         : REM WAEHLE 320 X 200
30 SCREEN OPEN 1              : REM OEFFNE GRAFIKSCHIRM
40 SCREEN SET 1,1             : REM AKTIVIERE SCHIRM
50 PALETTE 1,0,  0, 0, 0      : REM 0 = SCHWARZ
60 PALETTE 1,1, 15, 0, 0      : REM 1 = ROT
70 PALETTE 1,2,  0, 0,15      : REM 2 = BLAU
80 PALETTE 1,3,  0,15, 0      : REM 3 = GRUEN
90 PEN 2                      : REM SETZE ZEICHENSTIFT AUF BLAU (2)
100 LINE 160,0,240,100        : REM ZEICHNE ERSTE LINIE
110 LINE 240,100,80,100       : REM ZEICHNE ZWEITE LINIE
120 LINE 80,100,160,0         : REM ZEICHNE DRITTE LINIE
130 PAINT 160,10,0,2          : REM FUELLE DREIECK MIT BLAU (2)
140 GETKEY K$                 : REM WARTE AUF TASTENDRUCK
150 SCREEN CLOSE 1            : REM BEENDE UND SCHLIESSE GRAFIK

READY.
```



PASTE

Token: $E3

Format: **PASTE** x, y, 0, 0

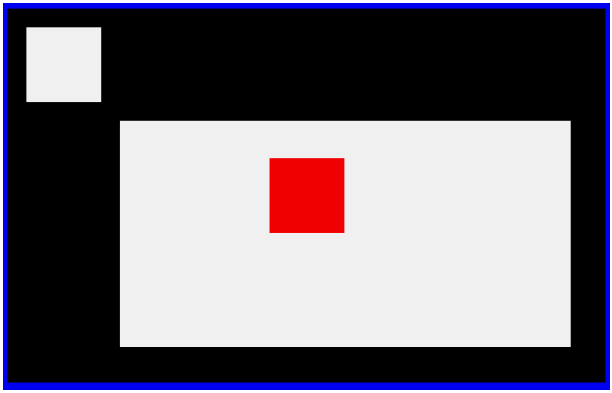
Zweck: **PASTE** (dt. *"einfügen"*) wird auf Grafikbildschirmen verwendet und fügt den Inhalt des Cut/Copy/Paste-Puffers auf dem Bildschirm an der Position **x, y** (Koordinaten für die linke obere Ecke) ein.

Notiz: Die beiden **0** als Argumente dürfen nicht weggelassen werden, weil dies einen **SYNTAX ERROR** erzeugen würde.

Beispiel: Verwendung von **PASTE**:

```
10 REM CUT
20 SCREEN 320,200,2     : REM OEFFNE GRAFIKBILDSCHIRM 320 X 200 X 2
30 BOX 60,60,300,180,1  : REM ZEICHNE EIN WEISSES RECHTECK
40 PEN 2                : REM WAEHLER ROTEN STIFT AUS
50 CUT  140,80,40,40    : REM SCHNEIDE EINEN 40 X 40 PIXEL BEREICH AUS
60 PASTE 10,10,0,0      : REM FUEGE IHN AN NEUER POSITION EIN
70 GETKEY A$            : REM WARTE AUF TASTENDRUCK
80 SCREEN CLOSE         : REM SCHLIESSE GRAFIKBILDSCHIRM

READY.
```



PEEK

Token: $C2

Format: **PEEK**(Adresse)

Zweck: Gibt einen vorzeichenlosen 8 Bit-Wert (Byte) zurück, der von **Adresse** gelesen wurde.

Wenn die Adresse im Bereich von $0000 bis $FFFF (0-65535) liegt, wird die Speicherbank, die durch **BANK** festgelegt ist, verwendet.

Adressen, die größer oder gleich $10000 (dezimal 65536) sind, werden als flache Speicheradressen verwendet, wobei die Einstellung **BANK** ignoriert wird.

Notiz: **BANK**-Werte von 0 bis 127 ermöglichen den Zugriff auf RAM- oder ROM-Bänke. **BANK**-Werte über 127 werden für den Zugriff auf Ein-/Ausgabe und die zugrunde liegende Systemhardware wie VIC (Grafik), SID (Musik), FDC (Diskettenkontroller) usw. verwendet.

Beispiel: Verwendung von **PEEK**:

```
10 BANK 128                :REM WAEHLE SYSTEM BANK AUS
20 L = PEEK($02F8)         :REM USR SPRUNGADRESSE (NIEDRIGES BYTE)
30 H = PEEK($02F9)         :REM USR SPRUNGADRESSE (HOEHERES BYTE)
40 T = L + 256 * H         :REM 16-BIT SPRUNGADRESSE
50 PRINT "USR-FUNKTION RUFT DIESE ADRESSE AUF: ";T

READY.
RUN
USR-FUNKTION RUFT DIESE ADRESSE AUF:  22316

READY.
```

PEEKW

Token: $C2 'W'

Format: **PEEKW(**Adresse**)**

Zweck: Gibt einen vorzeichenlosen 16 Bit-Wert (Wort) zurück, der von **Adresse** (low byte, dt. "niederwertiges Byte") und **Adresse**+1 (high byte, , dt. "höherwertiges Byte") gelesen wurde.

Liegt die Adresse im Bereich von $0000 bis $FFFF (0-65535), wird die mit **BANK** eingestellte Speicherbank verwendet.

Adressen, die größer oder gleich $10000 (dezimal 65536) sind, werden als flache Speicheradressen verwendet, wobei die Einstellung **BANK** ignoriert wird.

Notiz: **BANK**-Werte von 0 bis 127 ermöglichen den Zugriff auf RAM- oder ROM-Bänke. **BANK**-Werte über 127 werden für den Zugriff auf Ein-/Ausgabe und die zugrunde liegende Systemhardware wie VIC (Grafik), SID (Musik), FDC (Diskettenkontroller) usw. verwendet.

Beispiel: Verwendung von **PEEKW**:

```
10 UA = PEEKW($02F8)   :REM USR SPRUNGZIELADRESSE
20 PRINT "USR-FUNCTION ZIELADRESSE: ";UA

READY.
RUN
USR-FUNCTION ZIELADRESSE:  22316

READY.
```

PEN

Token: $FE $33

Format: **PEN** Farbe

Zweck: **PEN** (dt. "Stift") legt die Farbe des Grafikstifts fest.

Farbe Palettenindex. In der Tabelle unter **BACKGROUND** auf Seite 19 finden Sie die Farbwerte und die entsprechenden Farben.

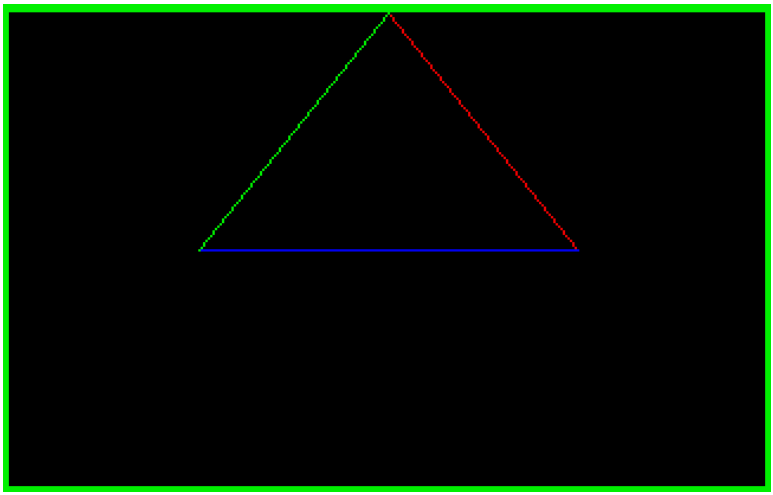
Notiz: Die mit **PEN** ausgewählte Farbe wird von allen nachfolgenden Grafik- und Zeichenbefehlen verwendet.

Es ist auch möglich, **PEN** mit zwei Argumenten aufzurufen. Das erste Argument steht für den zu verwendenen Zeichenmodus und darf 0,1 oder 2 sein. Das zweite Argument ist die **Farbe**. Zum aktuellen Zeitpunkt wird nur der Zeichenmodus 0 von BASIC 65 unterstützt, so dass das erste Argument entweder 0 sein muss oder weggelassen werden kann (dann wird der Zeichenmodus 0 als Standardwert verwendet).

Beispiel: Verwendung von **PEN**:

```
10 REM PEN
20 GRAPHIC CLR              : REM INITIALISIERE GRAFIK
30 SCREEN DEF 1,0,0,2       : REM WAEHLE 320 X 200
40 SCREEN OPEN 1            : REM OEFFNE GRAFIKSCHIRM 1
50 SCREEN SET 1,1           : REM AKTIVIERE GRAFIKSCHIRM 1
55 SCNCLR 0                 : REM LOESCHE GRAFIKSCHIRM MIT SCHWARZ (0)
60 PALETTE 1,0,  0, 0, 0    : REM 0 = SCHWARZ
70 PALETTE 1,1, 15, 0, 0    : REM 1 = ROT
80 PALETTE 1,2,  0, 0,15    : REM 2 = BLAU
90 PALETTE 1,3,  0,15, 0    : REM 3 = GRUEN
100 PEN 1                   : REM SETZE ZEICHENSTIFT (0) AUF ROT (1)
110 LINE 160,0,240,100      : REM ZEICHNE ROTE LINIE
120 PEN 2                   : REM SETZE ZEICHENSTIFT (0) AUF BLAU (2)
130 LINE 240,100,80,100     : REM ZEICHNE BLAUE LINIE
140 PEN 3                   : REM SETZE ZEICHENSTIFT (0) AUF GRUEN (3)
150 LINE 80,100,160,0       : REM ZEICHNE GRUENE LINIE
160 GETKEY K$               : REM WARTE AUF TASTENDRUCK
170 SCREEN CLOSE 1          : REM BEENDE UND SCHLIESSE GRAFIK

READY.
```



PIXEL

Token: $CE $0C

Format: **PIXEL**(x, y)

Zweck: Gibt die Farbe eines Pixels an der angegebenen Position zurück.

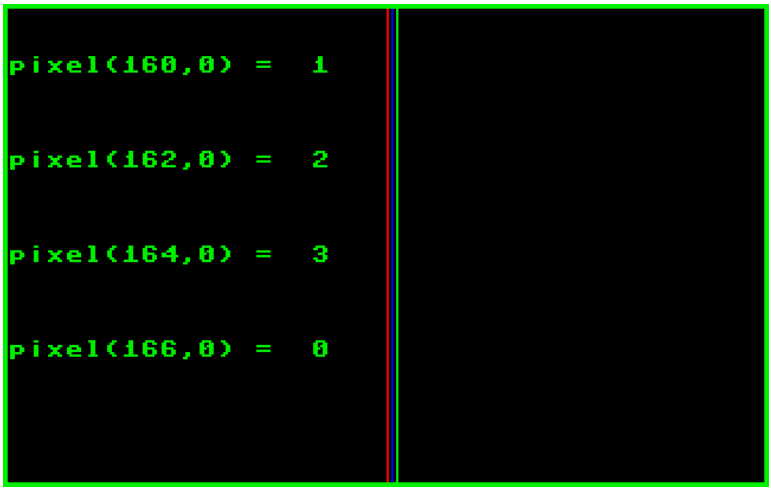
x absolute Bildschirmkoordinate.

y absolute Bildschirmkoordinate.

Beispiel: Verwendung von **PIXEL**:

```
10 REM PIXEL
20 GRAPHIC CLR              : REM INITIALISIERE GRAFIK
30 SCREEN DEF 1,0,0,2       : REM WAEHLE 320 X 200
40 SCREEN OPEN 1            : REM OEFFNE GRAFIKSCHIRM 1
50 SCREEN SET 1,1           : REM AKTIVIERE GRAFIKSCHIRM 1
60 SCNCLR 0                 : REM LOESCHE GRAFIKSCHIRM MIT SCHWARZ (0)
70 PALETTE  1,0,  0, 0, 0   : REM 0 = SCHWARZ
80 PALETTE  1,1, 15, 0, 0   : REM 1 = ROT
90 PALETTE  1,2,  0, 0,15   : REM 2 = BLAU
100 PALETTE 1,3, 0,15, 0    : REM 3 = GRUEN
110 PEN 1                   : REM SETZE ZEICHENSTIFT AUF ROT (1)
120 LINE 160,0,160,199      : REM ZEICHNE ROTE LINIE
130 PEN 2                   : REM SETZE ZEICHENSTIFT AUF BLAU (2)
140 LINE 162,0,162,199      : REM ZEICHNE BLAUE LINIE
150 PEN 3                   : REM SETZE ZEICHENSTIFT AUF GRUEN (3)
160 LINE 164,0,164,199      : REM ZEICHNE GRUENE LINIE
170 A=PIXEL(160,0)          : REM LESE FARBE VON PIXEL(160,0)
180 B=PIXEL(162,0)          : REM LESE FARBE VON PIXEL(162,0)
190 C=PIXEL(164,0)          : REM LESE FARBE VON PIXEL(164,0)
200 D=PIXEL(166,0)          : REM LESE FARBE VON PIXEL(166,0)
210 CHAR 0, 20,1,1,2,"PIXEL(160,0) = "+STR$(A)
220 CHAR 0, 60,1,1,2,"PIXEL(162,0) = "+STR$(B)
230 CHAR 0,100,1,1,2,"PIXEL(164,0) = "+STR$(C)
240 CHAR 0,140,1,1,2,"PIXEL(166,0) = "+STR$(D)
250 GETKEY K$               : REM WARTE AUF TASTENDRUCK
260 SCREEN CLOSE 1          : REM BEENDE UND SCHLIESSE GRAFIK
270 END

READY.
```


pixel(160,0) = 1
pixel(162,0) = 2
pixel(164,0) = 3
pixel(166,0) = 0

PLAY

Token: $FE $04

Format: **PLAY** [{String1**,** String2**,** String3**,** String4**,** String5**,** String6}]

Zweck: **PLAY** ohne Argumente führt dazu, dass alle Stimmen stumm geschaltet werden und alle BASIC-Variablen des Musiksystems zurückgesetzt werden (zum Beispiel **TEMPO**).

Auf **PLAY** können bis zu sechs durch Komma getrennte String-Argumente folgen, wobei jedes Argument die Abfolge von Noten und Direktiven angibt, die auf einer bestimmten Stimme auf den beiden verfügbaren SID-Chips gespielt werden sollen, was eine bis zu sechskanalige Polyphonie ermöglicht.

Eine Musiknote ist ein Zeichen (C, D, E, F, G, A oder B), dem ein optionaler Modifikator vorangestellt werden kann. Hierbei ist zu beachten, dass der Buchstabe 'B' für die Note 'H' steht (im Englischen wird die Note 'H' mit dem Buchstaben 'B' bezeichnet).

Mögliche Modifikatoren sind:

Zeichen	Effekt
#	spielt die Note einen Halbton höher
$	spielt die Note einen Halbton tiefer
.	spielt die Note als punktierte Note (1,5-fache Dauer)
S	spielt die Note als sechzehntel Note
I	spielt die Note als achtel Note
Q	spielt die Note als viertel Note
H	spielt die Note als halbe Note
W	spielt die Note als ganze Note
R	setzt eine Pause

Eingebettete Direktiven bestehen aus einem Buchstaben, gefolgt von einer Ziffer:

Zeichen	Direktive	Argumentbereich
O	Oktave	0 - 6
T	Instrumentenhüllkurve	0 - 9
U	Lautstärke	0 - 9
X	Filter	0 - 1
M	Modulation	0 - 9
P	Portamento	0 - 9
L	Schleife	*entfällt*

Die Modulations-Direktive moduliert die Note um die von Ihnen angegebene Stärke (1-9). Mit dem Argument 0 wird die Modulation deaktiviert.

Die Portamento-Direktive gleitet sanft zwischen aufeinanderfolgenden Noten mit der von Ihnen angegebenen Geschwindigkeit (1-9). Mit dem Argument 0 wird das Portamento deaktiviert. Bei aktiviertem Portamento entfällt die Ausklingphase der Hüllkurve.

Fügen Sie eine Anweisung **L** (kein Argument erforderlich) am Ende Ihres Strings ein, wenn Sie möchten, dass nach seiner Abarbeitung an den Anfang des Strings zurückgekehrt wird.

Sie sind sehr flexibel, wenn es darum geht, auf welchen Stimmkanälen Sie Ihre Melodien abspielen wollen. Sie können zum Beispiel beschließen, nur Stimme 1 und Stimme 4 für Ihre Melodie zu verwenden und die anderen Kanäle für Soundeffekte zu nutzen, die von **SOUND** erzeugt werden. Überspringen Sie einfach die Stimmen, die Sie mit **PLAY** nicht verwenden, indem Sie diese Argumente leer lassen:

```
PLAY "O4EDCDEEERL",,,"O2CGEGCGEGL"
```

Sie können sogar **PLAY** erneut aufrufen, um die oben erwähnten ungenutzten Kanäle zu verwenden, um eine andere Melodie neben Ihrer ersten Melodie zu spielen. Verwenden Sie dieses Mal zum Beispiel Stimme 2 und Stimme 5:

```
PLAY ,"O5T2IGAGFEDCEGO6.QCL",,,"O3T2.QG.B O4ICO3GE.QCL"
```

Wenn Sie feststellen möchten, ob eine Melodie auf einem Stimmen-Kanal abgespielt wird, können Sie dies herausfinden, indem Sie den von **RPLAY(Stimme)** zurückgegebenen Wert überprüfen, wobei der Stimmen-Parameter ein Wert von 1 bis 6 ist, der den Stimmen-Kanal angibt. Es wird hierbei entweder 1 (Wiedergabe) oder 0 (keine Wiedergabe) zurückgegeben.

Es ist zu beachten, dass BASIC-Strings eine maximale Länge von 255 Bytes haben. Wenn Ihre Melodie diese Länge überschreiten würde, sollten Sie in Erwägung ziehen, Ihre Melodie in mehrere Strings aufzuteilen und dann **RPLAY(voice)** zu verwenden, um festzustellen, wann der erste String fertig ist, um dann den nächsten String zu spielen.

Die Hüllkurven-Speicherplätze der Instrumente können mit der Anweisung **ENVELOPE** geändert werden. Die Standardeinstellungen für die Hüllkurven finden Sie auf Seite 95.

Notiz: Die Anweisung **PLAY** verwendet eine Interrupt-gesteuerte Routine, die mit dem Abarbeiten des Strings und dem Abspielen der Melodie beginnt. Die Programmausführung wird mit der nächsten Anweisung fortgesetzt und blockiert das Programm nicht.

Beispiel: Verwendung von **PLAY**:

```
10 REM *** PLAY -  EINFACHES SCHLEIFEN-BEISPIEL ***
20 ENVELOPE 9,10,5,10,5,0,300
30 VOL 8
40 TEMPO 30
50 PLAY "O5T9HCIDCDEHCG IGAGFEFDEWCL", "O2T0QCGEGCGEG DBGB CGEGL"

READY.
```

```
10 REM *** PLAY - MODULATION + PORTAMENTO BEISPIEL ***
20 TEMPO 20
30 M$ = "M5 T2O5P0QD P5FP0RP5QG .AI#AQA HGQE.C IDQE HFQD .DI#CQD HEQ#CQO4HA"
40 M$ = M$ + "O5QDHFQG.AI#AQA HGQE.C IDQEFED#CO4BO5#C DO4AFD P0R L"
50 B$ = "T0QRO2H.D.F.CO1.A.#A.G.A QAIO2AGFE H.D.F.CO1.A.#A.AO2 .D DL"
60 PLAY M$,B$

READY.
```

POINTER

Token: $CE $0A

Format: **POINTER**(Variable)

Zweck: Gibt die aktuelle Adresse einer Variablen oder eines Array-Elements als 32 Bit-Zeiger zurück. Bei String-Variablen ist es die Adresse des String-Deskriptors, nicht die des Strings selbst. Der String-Deskriptor besteht aus drei Bytes (Länge, niedrige String-Adresse und hohe String-Adresse).

Notiz: Die Adresswerte von Arrays und ihren Elementen sind während der Ausführung des Programms konstant.

Die Adressen von Strings (nicht ihre Deskriptoren) können sich jedoch aufgrund der sogenannten "Garbage Collection" jederzeit ändern.

Beispiel: Verwendung von **POINTER**:

```
100 REM POINTER
110 BANK 0                  :REM SKALARE VARIABLEN LIEGEN IN BANK 0
120 H$="HELLO"              :REM VERKNUEPFE STRING MIT H$
130 P=POINTER(H$)           :REM HOLE DESKRIPTOR-ADRESSE
140 PRINT "DESKRIPTOR BEI: $";HEX$(P)
150 L=PEEK(P):SP=PEEKW(P+1) :REM LAENGE UND STRING-POINTER
160 PRINT "LAENGE = ";L     :REM GIB LAENGE AUS
170 BANK 1                  :REM STRINGS LIEGEN IN BANK 1
180 FOR I%=0 TO L-1:PRINT PEEK(SP+I%);:NEXT:PRINT
190 FOR I%=0 TO L-1:PRINT CHR$(PEEK(SP+I%));:NEXT:PRINT

READY.
RUN
DESKRIPTOR BEI: $FD75
LAENGE =  5
 72  69  76  76  79
HELLO

READY.
```

POKE

Token: $97

Format: **POKE** Adresse**,** Byte [**,** Byte ...]

Zweck: Schreibt ein oder mehrere Bytes in den Speicher oder in die speichereingebunden Ein-/Ausgabe, beginnend bei **Adresse**.

Wenn die Adresse im Bereich von $0000 bis $FFFF (0-65535) liegt, wird die Speicherbank, die durch **BANK** festgelegt ist, verwendet.

Adressen, die größer oder gleich $10000 (dezimal 65536) sind, werden als flache Speicheradressen verwendet, wobei die Einstellung **BANK** ignoriert wird.

Byte ist ein Wert im Bereich von 0 bis 255.

Notiz: Die Adresse wird für jedes Datenbyte um Eins erhöht, so dass ein Speicherbereich mit einem einzigen **POKE** beschrieben werden kann.

BANK-Werte von 0 bis 127 ermöglichen den Zugriff auf RAM- oder ROM-Bänke. **BANK**-Werte über 127 werden für den Zugriff auf Ein-/Ausgabe und die zugrunde liegende Systemhardware wie VIC (Grafik), SID (Musik), FDC (Diskettenkontroller) usw. verwendet.

Beispiel: Verwendung von **POKE**:

```
10 REM POKE
20 BANK 128          :REM WAEHLE SYSTEM BANK AUS
30 POKE $02F8,0,24 :REM SETZE USR-ZEIGER AUF $1800

READY.
```

POKEW

Token: $97 'W'

Format: **POKEW** Adresse**,** Wort [**,** Wort ...]

Zweck: Schreibt ein oder mehrere Wörter (zwei Bytes) in den Speicher oder in die speichereingebunden Ein-/Ausgabe, beginnend bei **Adresse**.

Wenn die Adresse im Bereich von $0000 bis $FFFF (0-65535) liegt, wird die Speicherbank, die durch **BANK** festgelegt ist, verwendet.

Adressen, die größer oder gleich $10000 (dezimal 65536) sind, werden als flache Speicheradressen verwendet, wobei die Einstellung **BANK** ignoriert wird.

Wort ist ein Wert im Bereich von 0 bis 65535.

Das erste Wort wird unter der Adresse (niederwertiges Byte) und der Adresse+1 (höherwertiges Byte) gespeichert. Das zweite Wort wird unter Adresse+2 (niederwertiges Byte) und Adresse+3 (höherwertiges Byte) gespeichert, usw.

Notiz: Die Adresse wird für jedes Datenwort um Zwei erhöht, so dass ein Speicherbereich mit einem einzigen **POKEW** beschrieben werden kann.

BANK-Werte von 0 bis 127 ermöglichen den Zugriff auf RAM- oder ROM-Bänke. **BANK**-Werte über 127 werden für den Zugriff auf Ein-/Ausgabe und die zugrunde liegende Systemhardware wie VIC (Grafik), SID (Musik), FDC (Diskettenkontroller) usw. verwendet.

Beispiel: Verwendung von **POKEW**:

```
10 REM POKEW
20 BANK 128                :REM WAEHLE SYSTEM-BANK AUS
30 POKEW $02F8,$1800       :REM SETZE USR-ZEIGER AUF $1800

READY.
```

POLYGON

Token: $FE $2F

Format: **POLYGON** x, y, x-Radius, y-Radius, Polygonseiten [{, Zeichenseiten, Schneiden, Winkel, gefüllt}]

Zweck: Zeichnet ein regelmäßiges n-seitiges Polygon. Das Polygon wird unter Verwendung des aktuellen Zeichenkontextes, der mit **SCREEN**, **PALETTE** und **PEN** gesetzt worden ist, gezeichnet.

x,y Mittelpunktskoordinaten.

x-Radius,y-Radius Radius in x- und y-Richtung.

Polygonseiten Anzahl der Polygonseiten.

Zeichenseiten Seiten zu zeichnen.

Schneiden Linie von der Mitte zum Anfang ziehen (1).

Winkel Anfangswinkel.

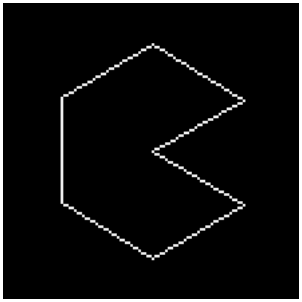
gefüllt gefüllt (1) oder Umrisslinie (0).

Notiz: Ein regelmäßiges Vieleck ist sowohl gleichwinklig als auch gleichseitig. Das heißt, dass alle Winkel und alle Seiten gleich sind.

Beispiel: Verwendung von **POLYGON**:

```
10 REM POLYGON
20 SCREEN 320,200,1                 : REM OEFFNE 320 X 200 GRAFIKSCHIRM
30 POLYGON 160,100,40,40,6,5,1,330 : REM ZEICHNE POLYGON
40 GETKEY A$                        : REM WARTE AUF TASTENDRUCK
50 SCREEN CLOSE                     : REM SCHLIESSE GRAFIKSCHIRM

READY.
```



POS

Token: $B9

Format: **POS**(Dummy)

Zweck: Gibt die Cursorspalte relativ zum aktuell verwendeten Fenster zurück.

Dummy ein numerischer Wert, der ignoriert wird.

Notiz: **POS** gibt die Spaltenposition für den Bildschirmcursor an. Sie funktioniert nicht bei umgeleiteter Ausgabe.

Beispiel: Verwendung von **POS**:

POT

Token: $CE $02

Format: **POT**(Paddle)

Zweck: Gibt die Position eines Paddels zurück.

Paddle Paddlenummer (1-4).

Das niederwertige Byte des Rückgabewerts ist der Paddle-Wert, wobei 0 die Grenze im Uhrzeigersinn und 255 die Grenze gegen den Uhrzeigersinn darstellt.

Ein Wert größer als 255 zeigt an, dass auch die Feuertaste gedrückt wird.

Notiz: Analoge Paddles sind verrauscht und ungenau. Der Bereich kann kleiner als 0-255 sein und die von **POT** zurückgegebenen Werte können etwas schwanken.

Beispiel: Verwendung von **POT**:

```
10 REM POT
20 X = POT(1)        : REM LESE PADDLE 1 EIN
30 B = X > 255       : REM TRUE (-1) FALLS FEUERKNOPF GEDRUECKT IST
40 V = X AND 255     : REM PADDLE 1-WERT

READY.
```

PRINT

Token: $99

Format: **PRINT** Argumentliste

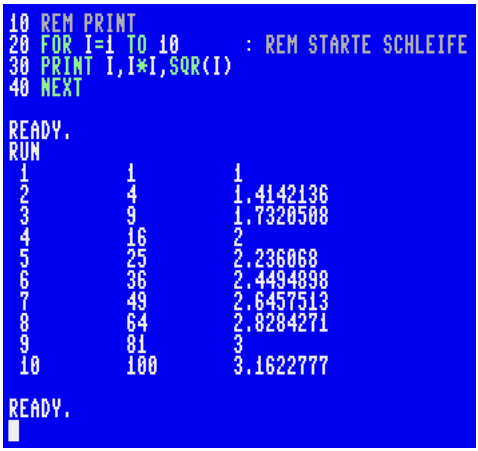
Zweck: Wertet die Argumentliste aus und gibt die Werte formatiert im aktuellen Bildschirmfenster aus. Je nach Argumenttyp wird die Standardformatierung verwendet. Für benutzergesteuerte Formatierung siehe **PRINT USING**.

Die folgenden Argumenttypen werden ausgewertet:

- **numerisch** Der Ausdruck beginnt mit einem Leerzeichen für positive und Nullwerte bzw. einem Minuszeichen für negative Werte. Ganzzahlige Werte werden mit der erforderlichen Anzahl von Ziffern gedruckt. Reelle Werte werden entweder in Festkommaform (normalerweise 9 Stellen) oder in wissenschaftlicher Form gedruckt, wenn der Wert außerhalb des Bereichs von 0,01 bis 999999999 liegt.
- **String** Die Zeichenfolge kann aus druckbaren Zeichen und Steuercodes bestehen. Darstellbare Zeichen werden an der Cursorposition ausgegeben, während Steuercodes ausgeführt werden.
- **,** ein Komma dient als Tabulator.
- **;** ein Semikolon fungiert als Trennzeichen zwischen den Argumenten der Liste. Anders als das Komma fügt es keine weiteren Zeichen ein. Ein Semikolon am Ende der Argumentliste unterdrückt den automatischen Rücklauf ("carriage return").

Notiz: Die Funktionen **SPC** und **TAB** können in der Argumentliste zur Positionierung verwendet werden. **CMD** kann für die Umleitung verwendet werden.

Beispiel: Verwendung von **PRINT**:



```
10 REM PRINT
20 FOR I=1 TO 10      : REM STARTE SCHLEIFE
30 PRINT I,I*I,SQR(I)
40 NEXT

READY.
RUN
 1          1          1
 2          4          1.4142136
 3          9          1.7320508
 4          16         2
 5          25         2.236068
 6          36         2.4494898
 7          49         2.6457513
 8          64         2.8284271
 9          81         3
 10         100        3.1622777

READY.
```

PRINT#

Token: $98

Format: **PRINT#** Kanal**,** Argumentliste

Zweck: Wertet die Argumentliste aus und gibt die formatierten Werte auf dem Gerät aus, das **Kanal** zugewiesen ist. Es wird die Standardformatierung verwendet, abhängig vom Argumenttyp. Für benutzergesteuerte Formatierung siehe **PRINT# USING**.

Kanalnummer Kanalnummer, die bei einem früheren Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** verwendet wurde.

Die folgenden Argumenttypen werden ausgewertet:

- **numerisch** Der Ausdruck beginnt mit einem Leerzeichen für positive und Nullwerte bzw. einem Minuszeichen für negative Werte. Ganzzahlige Werte werden mit der erforderlichen Anzahl von Ziffern gedruckt. Reelle Werte werden entweder in Festkommaform (normalerweise 9 Stellen) oder in wissenschaftlicher Form gedruckt, wenn der Wert außerhalb des Bereichs von 0,01 bis 999999999 liegt.
- **String** Die Zeichenfolge kann aus druckbaren Zeichen und Steuercodes bestehen. Darstellbare Zeichen werden an der Cursorposition ausgegeben, während Steuercodes ausgeführt werden.
- **,** ein Komma dient als Tabulator.
- **;** ein Semikolon fungiert als Trennzeichen zwischen den Argumenten der Liste. Anders als das Komma fügt es keine weiteren Zeichen ein. Ein Semikolon am Ende der Argumentliste unterdrückt den automatischen Rücklauf ("carriage return").

Notiz: Die Funktionen **SPC** und **TAB** sind nicht für andere Geräte als den Bildschirm geeignet.

Beispiel: Verwendung von **PRINT#** zum Schreiben einer Datei auf Gerät 8:

```
10 REM PRINT#
20 DOPEN#2,"TABELLE",W,U8
30 FOR I=1 TO 10
40 PRINT#2,I,I*I,SQR(I)
50 NEXT
60 DCLOSE#2

READY.
RUN
```

Sie können überprüfen, ob die Datei **'TABELLE'** geschrieben wurde, indem Sie **DIR "TABELLE"** eingeben und sich dann den Inhalt der Datei anzeigen lassen, indem Sie **TYPE "TABELLE"** eingeben.

```
DIR "TABELLE"
   0 "BASIC EXAMPLES  " BS 3D
   2 "TABELLE"          SEQ
2226 BLOCKS FREE

READY.

TYPE "TABELLE"
 1            1            1
 2            4            1.4142136
 3            9            1.7320508
 4            16            2
 5            25           2.236068
 6            36           2.4494898
 7            49           2.6457513
 8            64           2.8284271
 9            81           3
 10            100           3.1622777


READY.
```

PRINT USING

Token: $98 $FB or $99 $FB

Format: **PRINT**[**#** Kanal**,**] **USING** Format**;** Argument

Zweck: Gibt unter Berücksichtigung von **Format** das **Argument** aus.

Das Argument kann entweder ein String oder ein numerischer Wert sein. Das Format der Ausgabe richtet sich nach dem String **Format**.

Kanal ist die Kanalnummer, die bei einem vorherigen Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** verwendet worden ist. Wenn kein Kanal angegeben wird, erfolgt die Ausgabe auf dem Bildschirm.

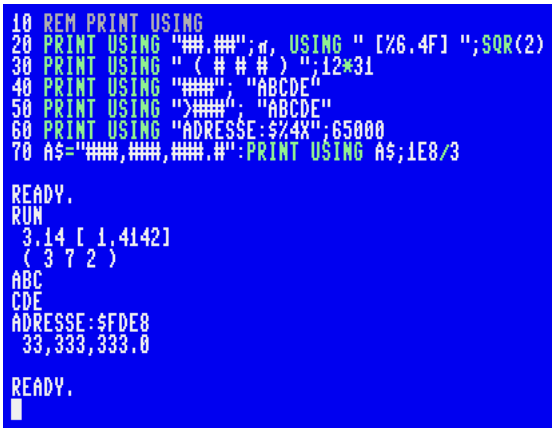
Format ist eine String-Variable oder String-Konstante, die die Regeln für die Formatierung definiert. Bei Verwendung einer Zahl als **Argument** kann die Formatierung entweder im CBM-Stil mit einem Muster wie **###.##** oder im C-Stil mit einem <Breite.Genauigkeit>-Spezifikator wie **%3D %7.2F %4X** erfolgen.

Argument ist die zu formatierende Zahl. Wenn das Argument nicht in das Format passt, zum Beispiel wenn versucht wird, eine vierstellige Variable in eine Reihe von drei Rautezeichen auszugeben (**###**), werden stattdessen Sterne (*****) verwendet.

Notiz: Der Formatstring wird nur für ein Argument verwendet. Es ist allerdings möglich, weitere **USING Format;Argument**-Sequenzen anzuhängen.

Argument kann aus darstellbaren Zeichen und Steuercodes bestehen. Darstellbare Zeichen werden an der Cursorposition ausgegeben, während Steuercodes ausgeführt werden. Die Anzahl der **#**-Zeichen bestimmt die Breite der Ausgabe. Ist das erste Zeichen des Formatstrings ein Gleichheitszeichen **=**, wird der Argumentstring zentriert. Ist das erste Zeichen des Formatstrings ein **>**-Zeichen, wird der Argumentstring rechtsbündig ausgerichtet.

Beispiel: Verwendung von **PRINT USING**:



```
10 REM PRINT USING
20 PRINT USING "##.##";π, USING " [%6.4F] ";SQR(2)
30 PRINT USING " ( # # # ) ";12*31
40 PRINT USING "!###"; "ABCDE"
50 PRINT USING "_!###"; "ABCDE"
60 PRINT USING "ADRESSE:$%4X";65000
70 A$="####,###,###.#":PRINT USING A$;1E8/3

READY.
RUN
 3.14 [ 1.4142]
 ( 3 7 2 )
ABC
CDE
ADRESSE:$FDE8
 33,333,333.0

READY.
```

RCOLOR

Token: $CD

Format: **RCOLOR**(Farbquelle)

Zweck: Gibt den aktuellen Farbindex für die ausgewählte Farbquelle zurück.

Farbquellen sind:

- **0** Hintergrundfarbe (**BACKGROUND**) (VIC $D021).
- **1** Textfarbe (**FOREGROUND**) ($F1).
- **2** Hervorhebungsfarbe (**HIGHLIGHT**) ($2D8).
- **3** Rahmenfarbe (**BORDER**) (VIC $D020).

Beispiel: Verwendung von **RCOLOR**:

```
10 REM RCOLOR
20 C = RCOLOR(3)  : REM C = FARBINDEX DES RAHMENS
30 PRINT C

READY.
RUN
 6

READY.
```

RCURSOR

Token: $FE $42

Format: **RCURSOR** {Spaltenvariable**,** Zeilenvariable}

Zweck: Liest die aktuelle Spalte und Zeile des Cursors in die **Spaltenvariable** und **Zeilenvariable**.

Notiz: Die Zeilen- und Spaltenwerte beginnen bei Null, wobei die Spalte ganz links den Wert Null und die oberste Zeile den Wert Null hat.

Beispiel: Verwendung von **RCURSOR**:

```
10 REM RCURSOR
20 CURSOR ON,20,10
30 PRINT "[HIER]";
40 RCURSOR X,Y
50 PRINT " SPALTE:";X;" ZEILE:";Y

READY.
RUN
                    [HIER] SPALTE: 26  ZEILE: 10

READY.
```

READ

Token: $87

Format: **READ** Variable [**,** Variable ...]

Zweck: Liest Werte aus der Programmquelle in Variablen.

Variable Liste von beliebigen gültigen Variablen.

Alle Arten von Konstanten (Integer, Real und Strings) können gelesen werden, allerdings keine Ausdrücke. Elemente werden durch Kommas getrennt. Strings, die Kommas, Doppelpunkte oder Leerzeichen enthalten, müssen in Anführungszeichen gesetzt werden.

RUN initialisiert den Datenzeiger mit dem ersten Element der **DATA**-Anweisung und schiebt ihn bei jedem gelesenen Element auf das nächste zu lesende Element weiter. Sie sind beim Programmieren dafür verantwortlich, dass der Typ der Konstanten mit dem Typ der Variable in der **READ**-Anweisung übereinstimmt. Leere Elemente ohne Konstante zwischen den Kommas sind zulässig und werden interpretiert als Null für numerische Variablen und eine leere Zeichenkette für String-Variablen.

RESTORE kann verwendet werden, um den Datenzeiger für nachfolgende **READ**-Anweisungen auf eine bestimmte Zeile zu setzen.

Notiz: Es ist gute Programmierpraxis, große Mengen von DATA-Anweisungen an das Ende des Programms zu setzen, damit sie die Suche nach Zeilennummern nach einem **GOTO**-Befehl oder anderen Anweisungen mit Zeilennummern als Ziele nicht verlangsamen.

Beispiel: Verwendung von **READ**:

```
10 REM READ
20 READ NA$, VE
30 READ N% : FOR I=1 TO N% : READ GL(I) : NEXT I
40 PRINT
50 PRINT "PROGRAMM:";NA$;"  VERSION:";VE
60 PRINT
70 PRINT "PREISENTWICKLUNG:"
80 FOR I=1 TO N%:PRINT GL(I):NEXT I
90 END
100 DATA "MEGA65",1.1
110 DATA 4,0.51,0.36,0.28,0.23

READY.
RUN

PROGRAMM:MEGA65  VERSION: 1.1

PREISENTWICKLUNG:
 0.51
 0.36
 0.28
 0.23

READY.
```

RECORD

Token: $FE $12

Format: **RECORD#** Kanal**,** Datensatz [**,** Byte]

Zweck: **RECORD** (dt. "Datensatz") positioniert den Lese-/Schreibzeiger einer relativen Datei.

Kanal ist eine Kanalnummer, die bei einem früheren Aufruf von Befehlen wie **DOPEN** oder **OPEN** verwendet wurde.

Datensatz ist der Zieldatensatz im Bereich 1 - 65535.

Byte Byte-Position im Datensatz.

RECORD kann nur für Dateien des Typs **REL** verwendet werden, bei denen es sich um relative Dateien handelt, die einen direkten Zugriff ermöglichen.

RECORD positioniert den Dateizeiger auf die angegebene Datensatznummer. Wenn diese Datensatznummer nicht existiert und auf der Platte, auf die **RECORD** schreibt, genügend Platz vorhanden ist, wird die Datei durch Hinzufügen leerer Datensätze auf die angeforderte Datensatzanzahl erweitert. In diesem Fall gibt der Plattenstatus die Meldung **RECORD NOT PRESENT** (dt. "Datensatz existiert nicht") aus, was aber kein Fehler ist.

Nach einem Aufruf von **INPUT#** oder **PRINT#** springt der Dateizeiger auf die nächste Datensatzposition.

Notiz: Die Commodore-Diskettenlaufwerke haben einen Fehler in ihrem DOS, der Daten durch die Verwendung relativer Dateien zerstören kann. Als Abhilfe wird empfohlen, den Befehl **RECORD** zweimal zu verwenden, vor und nach der Ein-/Ausgabe-Operation.

Beispiel: Verwendung von **RECORD**:

```
100 REM RECORD
110 DOPEN#2,"DATENBANK",L240          : REM OEFFNE ODER ERSTELLE DATEI
120 FOR I%=1 TO 20                    : REM SCHREIB-SCHLEIFE
130 PRINT#2,"DATENSATZ NR.";I%        : REM SCHREIBE DATENSATZ
140 NEXT I%                           : REM SCHLEIFENENDE
150 DCLOSE#2                          : REM SCHLIESSE DATEI
160                                   : REM NUN DAS TESTEN
170 DOPEN#2,"DATENBANK",L240          : REM WIEDEROEFFNEN
180 FOR I%=20 TO 2 STEP -2            : REM LESE DATEI RUECKWAERTS
190 RECORD#2,I%                       : REM POSITIONIERE AUF DATENSATZ
200 INPUT#2,A$                        : REM LESE DATENSATZ
210 PRINT A$;:IF I% AND 2 THENPRINT
220 NEXT I%                           : REM SCHLEIFENENDE
230 DCLOSE#2                          : REM SCHLIESSE DATEI

READY.
RUN
DATENSATZ NR. 20 DATENSATZ NR. 18
DATENSATZ NR. 16 DATENSATZ NR. 14
DATENSATZ NR. 12 DATENSATZ NR. 10
DATENSATZ NR. 8 DATENSATZ NR. 6
DATENSATZ NR. 4 DATENSATZ NR. 2

READY.
```

REM

Token: $8F

Format: **REM**

Zweck: Markiert alle Zeichen nach **REM** in der gleichen Zeile als Kommentar, ohne Einfluss auf den Programmablauf.

Notiz: Zeichen nach **REM** werden von BASIC 65 nicht ausgeführt.

Beispiel: Verwendung von **REM**:

```
10 REM REM-BEISPIEL
20 REM
30 REM REM KANN FUER KOMMENTARE IM PROGRAMM VERWENDET WERDEN.
40 REM ALLE ZEICHEN HINTER EINEM REM WERDEN IGNORIERT
50 :
60 PRINT "HALLO"

READY.
RUN
HALLO

READY.
```

RENAME

Token: $F5

Format: **RENAME** Alt **TO** Neu [**,D** Laufwerk] [**,U** Gerät]

Zweck: Benennt eine Diskettendatei um.

Alt ist entweder ein String in Anführungszeichen, zum Beispiel **"DATEN"** oder ein String-Ausdruck in Klammern, zum Beispiel **(FI$)**).

Neu ist entweder ein String in Anführungszeichen, zum Beispiel **"BACK-UP"** oder ein String-Ausdruck in Klammern, zum Beispiel **(FS$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **RENAME** wird im DOS des Laufwerks ausgeführt. Es kann alle regulären Dateitypen (**PRG**, **REL**, **SEQ**, **USR**) umbenennen. Die alte Datei muss existieren und die neue Datei darf nicht existieren. Es können nur einzelne Dateien umbenannt werden, Platzhalterzeichen wie ***** und **?** sind nicht zulässig. Der Dateityp kann nicht geändert werden.

Beispiel: Verwendung von **RENAME**:

```
DIR "DATEN*"
   0 "BASIC EXAMPLES  " BS 3D
  21 "DATENBANK"       REL
   1 "DATEN"           PRG
2184 BLOCKS FREE

READY.

RENAME "DATEN" TO "DATEN-BACKUP" : REM NENNE EINZELNE DATEI UM

READY.
DIR "DATEN*"
   0 "BASIC EXAMPLES  " BS 3D
  21 "DATENBANK"       REL
   1 "DATEN-BACKUP"    PRG
2184 BLOCKS FREE

READY.
```

RENUMBER

Token: $F8

Format: **RENUMBER** [{Neu**,** Schrittweite**,** Bereich}]

Zweck: **RENUMBER** wird verwendet, um alle Zeilen oder einen Bereich von Zeilen eines BASIC-Programms neu zu nummerieren.

Neu ist die neue Anfangszeile des neu zu nummerierenden Zeilenbereichs. Der Standardwert ist 10.

Schrittweite ist die zu verwendende Schrittweite bei der Neunummerierung. Der Standardwert ist 10.

Bereich ist der neu zu nummerierende Zeilenbereich. Die Standardwerte sind von der ersten bis zur letzten Zeile.

RENUMBER wird entweder im "erhaltenden" Modus oder im "optimierenden" Modus ausgeführt. Im "optimierenden" Modus werden Leerzeichen zum Beispiel vor den Zeilennummern oder zwischen **THEN** und **GOTO**/**GOSUB** entfernt, wodurch die Codegröße verringert und die Ausführungszeit verkürzt wird. Im "erhaltenden" Modus werden alle Leerzeichen im Programmcode beibehalten. Der "optimierende" Modus wird ausgelöst, indem das erste Argument, die **neue** Startnummer, neben dem Schlüsselwort **RENUMBER** ohne Leerzeichen dazwischen eingegeben wird.

RENUMBER ändert alle Zeilennummern im gewählten Bereich und ändert auch alle Verweise in Anweisungen, die **GOSUB**, **GOTO**, **RESTORE**, **RUN**, **TRAP**, usw. verwenden.

RENUMBER kann nur im direkten Modus ausgeführt werden. Wenn es ein Problem wie Speicherüberlauf, nicht aufgelöste Referenzen oder Zeilenzahlüberlauf (mehr als 64000 Zeilen) feststellt, bricht es mit einer Fehlermeldung ab und lässt das Programm unverändert.

RENUMBER kann mit 0 bis 3 Parametern aufgerufen werden. Nicht spezifizierte Parameter verwenden ihre Standardwerte.

Notiz: **RENUMBER** kann bei großen Programmen mehrere Minuten zur Ausführung benötigen.

Beispiel: Verwendung von **RENUMBER**:

```
RENUMBER              : REM ERHALTEND - ZEILENNUMMERN WERDEN 10,20,30,...
RENUMBER 100,5        : REM ERHALTEND - ZEILENNUMMERN WERDEN 100,105,110,...
RENUMBER 100,5,120-180 : REM ERHALTEND - ÄENDERT ZEILEN 120-180 NACH 100,105,...
RENUMBER601,1,500     : REM OPTIMIEREND - STARTET VON 500 NACH 601,602,...

10 GOTO  20
20 GOTO 10

RENUMBER 100,10   : REM ERHALTENDE VARIANTE

LIST

100 GOTO  110
110 GOTO 100

RENUMBER100,10    : REM OPTIMIERENDE VARIANTE

LIST

100 GOTO110
110 GOTO100

READY.
```

RESTORE

Token: $8C

Format: **RESTORE** [Zeilennumer]

Zweck: Setzen oder Zurücksetzen des internen Zeigers für **READ** von **DATA**-Anweisungen.

Zeilennummer ist die neue Position für den Zeiger. Der Standardwert ist die erste Programmzeile.

Notiz: Das neue Zeigerziel **Zeilennummer** muss keine **DATA**-Anweisungen enthalten. Mit jeder **READ**-Anweisung wird der Zeiger automatisch zur nächsten **DATA**-Anweisung weitergeschaltet.

Beispiel: Verwendung von **RESTORE**:

```
100 REM RESTORE
110 DATA 3,1,4,1,5,9,2,6
120 DATA "MEGA65"
130 DATA 2,7,1,8,2,8,9,5
140 FOR I=1 TO 8:READ P:PRINT P;:NEXT
150 PRINT
160 RESTORE 130
170 FOR I=1 TO 8:READ P:PRINT P;:NEXT
180 PRINT
190 RESTORE 120
200 READ A$:PRINT A$

READY.
RUN
 3  1  4  1  5  9  2  6
 2  7  1  8  2  8  9  5
MEGA65

READY.
```

RESUME

Token: $D6

Format: **RESUME** [Zeilennummer | **NEXT**]

Zweck: Wird in einer **TRAP**-Routine verwendet, um die normale Programmausführung nach der Behandlung eines Fehlers wieder aufzunehmen.

Mit **RESUME** ohne Parameter wird versucht, die Anweisung, die den Fehler verursacht hat, erneut auszuführen. Die Routine **TRAP** sollte das Problem, bei dem der Fehler aufgetreten ist, untersucht und korrigiert haben.

Zeilennummer ist die Zeilennummer, bei der die Programmausführung fortgesetzt werden soll.

NEXT nimmt die Ausführung nach der Anweisung, die den Fehler verursacht hat, wieder auf. Dies kann die nächste Anweisung in derselben Zeile (getrennt durch einen Doppelpunkt :) oder die Anweisung in der nächsten Zeile sein.

Notiz: **RESUME** kann nicht im Direktmodus verwendet werden.

Beispiel: Verwendung von **RESUME**:

```
10 REM RESUME
20 TRAP 80
30 FOR I=80 TO 100
40 PRINT EXP(I)
50 NEXT
60 PRINT "GESTOPPT BEI I =";I
70 END
80 PRINT ERR$(ER): RESUME 60

READY.
RUN
 5.5406224E34
 1.5060973E35
 4.0939969E35
 1.1128638E36
 3.0250773E36
 8.2230125E36
 2.2352466E37
 6.0760302E37
 1.6516362E38
OVERFLOW
GESTOPPT BEI I = 89

READY.
```

RETURN

Token: $8E

Format: **RETURN**

Zweck: Springt im Programmablauf von einem Unterprogramm, das mit **GOSUB** oder einem mit **COLLISION** deklarierten Ereignishandler aufgerufen wurde, zurück an die Aufrufstelle.

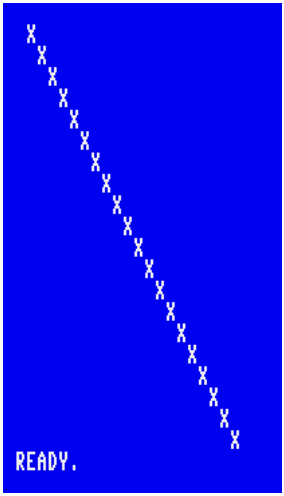
Die Ausführung wird mit der Anweisung fortgesetzt, die auf den Aufruf von **GOSUB** folgt.

Im Falle des Handlers **COLLISION** wird die Ausführung an der Stelle fortgesetzt, an der sie zum Aufruf des Handlers verlassen wurde.

Beispiel: Verwendung von **RETURN**:

```
100 REM RETURN
110 SCNCLR           :REM LOESCHE BILDSCHIRM
120 FOR I=1 TO 20    :REM DEFINIERE SCHLEIFE
130 GOSUB170:REM RUFE UNTERPROGRAMM AUF
140 NEXT I           :REM SCHLEIFENENDE
150 END              :REM PROGRAMMENDE
160 :
170 CURSOR ON,I,I,0  :REM AKTIVIERE UND POSITIONIERE DEN CURSOR
180 PRINT "X";       :REM GIB X AUS
190 SLEEP 0.5        :REM WARTE 0.5 SEKUNDEN
200 CURSOR OFF       :REM SCHALTE BLINKENDEN CURSOR AUS
210 RETURN           :REM ZURUECK ZUR AUFRUFSTELLE

READY.
```



RGRAPHIC

Token: $CC

Format: **RGRAPHIC**(Bildschirm, Parameter)

Zweck: Rückgabe des grafischen Bildschirmstatus und der grafischen Parameter.

Parameter	Beschreibung
0	Geöffnet (1), Geschlossen (0) oder Ungültig (>1)
1	Breite (0=320, 1=640)
2	Höhe (0=200, 1=400)
3	Tiefe (1-8 Bitplanes)
4	Verwendete Bitplanes (Bitmaske)
5	Verwendete Bank 4-Blöcke (Bitmaske)
6	Verwendete Bank 5-Blöcke (Bitmaske)
7	Drawscreen-Nr. (0-3) (Bildschirm, auf den gezeichnet wird)
8	Viewscreen-Nr. (0-3) (Bildschirm, der angezeigt wird)
9	Zeichenmodi (Bitmaske)
10	Mustertypen (Bitmaske)

Beispiel: Verwendung von **RGRAPHIC**:

```
100 REM RGRAPHIC
110 GRAPHIC CLR          : REM INITIALISIERE GRAFIK
120 SCREEN DEF 0,1,0,4   : REM GRAFIKBILDSCHIRM  0:640 X 200 X 4
130 SCREEN OPEN 0        : REM OEFFNE GRAFIKBILDSCHIRM
140 SCREEN SET 0,0       : REM DRAWSCREEN = VIEWSCREEN = 0
150 SCNCLR 0             : REM LOESCHE BILDSCHIRM
160 PEN 0,1              : REM WAEHLE FARBE AUS
170 LINE 0,0,639,199     : REM ZEICHNE LINIE
180 FOR I=0 TO 10
190 A(I)=RGRAPHIC(0,I)
200 NEXT I
210 SCREEN CLOSE 0
220 FOR I=0 TO 6
230 PRINT I;A(I)         : REM PRINT INFO
240 NEXT I

READY.
RUN
 0  1
 1  1
 2  0
 3  4
 4  15
 5  15
 6  15

READY.
```

RIGHT$

Token: $C9

Format: **RIGHT$**(String**,** Länge**)**

Zweck: Gibt einen String zurück, der die letzten **Länge** Zeichen von **String** enthält. Wenn die Länge von **String** gleich oder kleiner als **Länge** ist, ist der Ergebnisstring identisch mit dem Argument **String**.

String ist ein String-Ausdruck.

Länge ist ein numerischer Ausdruck (0-255).

Notiz: Leere Strings und Längen von Null sind zulässige Werte.

Beispiel: Verwendung von **RIGHT$**:

```
100 REM RIGHT$
110 PRINT RIGHT$("MEGA65",2)

READY.
RUN
65

READY.
```

RMOUSE

Token: $FE $3F

Format: **RMOUSE** x-Variable**,** y-Variable**,** Maustasten-Variable

Zweck: Liest die Mausposition und den Status der Maustasten.

x-Variable ist eine numerische Variable, in der die x-Position gespeichert wird.

y-Variable ist eine numerische Variable, in der die y-Position gespeichert wird.

Mausknopf-Variable ist eine numerische Variable, die den Status der Maustasten erhält.

Die linke Maustaste setzt Bit 7, während die rechte Maustaste Bit 0 setzt:

Wert	Status
0	Keine Maustaste gedrückt
1	Rechte Maustaste gedrückt
128	Linke Maustaste gedrückt
129	Beide Maustasten gedrückt

RMOUSE setzt alle Parameter-Variablen auf -1, wenn die Maus nicht angeschlossen oder deaktiviert ist.

Notiz: Aktivierte Mäuse an beiden Anschlüssen fassen die Ergebnisse zusammen.

Beispiel: Verwendung von **RMOUSE**:

```
100 REM RMOUSE
110 MOUSE ON
120 SCNCLR
130 PRINT "DRUECKE BEIDE MAUSTASTEN, UM DAS PROGRAMM ZU BEENDEN."
140 RMOUSE X,Y,B
150 M$=STR$(X)+" "+STR$(Y)+" "+STR$(B)+"                          "
160 CURSOR 10,10
170 PRINT "MAUSSTATUS: ";M$
180 IF B > 128 THEN GOTO 200
190 GOTO 140
200 MOUSE OFF

READY.
```

```
DRUECKE BEIDE MAUSTASTEN, UM DAS PROGRAMM ZU BEENDEN.









         MAUSSTATUS:  204  138  0
```

RND

Token: $BB

Format: **RND**(Typ)

Zweck: Gibt eine Pseudo-Zufallszahl zurück.

Dies wird als "Pseudo"-Zufallszahl bezeichnet, da die Zahlen nicht wirklich zufällig sind. Sie werden von einer anderen Zahl abgeleitet, die als "seed" (dt. "Saat") bezeichnet wird und reproduzierbare Sequenzen erzeugt. **Typ** legt fest, welcher "seed"verwendet wird:

- **Typ = 0** verwende die Systemuhr.
- **Typ < 0** verwendet den Wert von **Typ** als "seed".
- **Typ > 0** leitet eine neue Zufallszahl aus der vorherigen ab.

Notiz: Gesetzte Zufallszahlenfolgen ergeben für identische "seeds" die gleiche Folge.

Bestmögliche Zufallszahlen lassen sich erreichen, indem man zuerst den Zufallsgenerator mit **RND** und einem negativen Argument initialisiert und anschließend den Zufallsgenerator mit **RND(1)** nochmals initialisiert.

Beispiel: Verwendung von **RND**:

```
10 REM RND
20 DEF FNDI(X) = INT(RND(0)*6)+1 : REM WUERFEL-FUNKTION
30 FOR I=1 TO 10                 : REM WUERFLE 10 MAL
40 PRINT I;FNDI(0)               : REM GIB DIE GEWUERFELTE ZAHL AUS
50 NEXT

READY.
RUN
 1  2
 2  1
 3  2
 4  2
 5  3
 6  3
 7  4
 8  4
 9  6
 10  5

READY.
```

RPALETTE

Token: $CE $0D

Format: **RPALETTE**(Bildschirm**,** Index**,** RGB**)**

Zweck: Gibt den Rot-, Grün- oder Blauwert eines Palettenfarbindexes zurück.

Bildschirm Bildschirmnummer (0-3)

Index Palettenfarbindex

RGB (0: Rot, 1: Grün, 2: Blau)

Beispiel: Verwendung von **RPALETTE**:

```
10 REM RPALETTE
20 SCREEN 320,200,4    : REM DEFINIERE UND OEFFNE GRAFIKSCHIRM
30 R = RPALETTE(0,3,0) : REM LESE ROT
40 G = RPALETTE(0,3,1) : REM LESE GRUEN
50 B = RPALETTE(0,3,2) : REM LESE BLAU
60 SCREEN CLOSE        : REM SCHLIESSE GRAFIKSCHIRM
70 PRINT "PALETTENINDEX 3 RGB =";R;G;B

READY.
RUN
PALETTENINDEX 3 RGB = 0  15  15

READY.
```

RPEN

Token: $D0

Format: **RPEN**(Stift)

Zweck: Gibt den Farbindex von Stift **Stift** zurück.

Stift Stiftnummer (0-2), wobei:

- 0 Zeichenstift
- 1 Löschstift
- 2 Konturenstift

Beispiel: Verwendung von **RPEN**:

```
100 REM RPEN
110 GRAPHIC CLR          : REM INITIALISIERE GRAFIK
120 SCREEN DEF 0,1,0,4   : REM GRAFIKSCHIRM 0:640 X 200 X 4
130 SCREEN OPEN 0        : REM OEFFNE GRAFIKSCHIRM 0
140 SCREEN SET 0,0       : REM DRAWSCREEN = VIEWSCREEN = 0
150 SCNCLR 0             : REM LOESCHE GRAFIKSCHIRM 0
160 PEN 0,1              : REM WAEHLE FARBE AUS
170 X = RPEN(0)
180 Y = RPEN(1)
190 C = RPEN(2)
200 SCREEN CLOSE 0
210 PRINT "FARBE DES ZEICHENSTIFTS = ";X

READY.
RUN
FARBE DES ZEICHENSTIFTS =  1

READY.
```

RPLAY

Token: $FE $0F

Format: **RPLAY**(Stimme)

Zweck: **RPLAY** gibt einen Wert (0 oder 1) zurück, um anzuzeigen, ob eine Melodie auf dem angegebenen Stimmkanal gespielt wird (1) oder nicht (0).

Stimme ist der zu überprüfende Stimmkanal (1-6).

Beispiel: Verwendung von **RPLAY**:

```
10 REM RPLAY
20 PLAY "O4ICDEFGAB05CR","O2QCGEGC01GCR"
30 IF RPLAY(1) OR RPLAY(2) THEN GOTO 30 : REM WARTE AUF DAS ENDE DES SONGS

READY.
```

RREG

Token: $FE $09

Format: **RREG** [{a-Register**,** x-Register**,** y-Register**,** z-Register**,** s-Register}]

Zweck: Liest die Werte, die nach einem **SYS**-Aufruf in den CPU-Registern stehen, in die angegebenen Variablen.

a-Register erhält den Wert des Akkumulators.

x-Register erhält den Wert des X-Registers.

y-Register erhält den Wert des Y-Registers.

z-Register erhält den Wert des Z-Registers.

s-Register erhält den Wert des Statusregisters.

Notiz: Die Registerwerte werden nach einem **SYS**-Aufruf im Systemspeicher abgelegt. Auf diese Weise kann **RREG** sie abfragen.

Beispiel: Verwendung von **RREG**:

```
10 REM RREG
20 BANK 128
30 BLOAD "ML PROG",8192
40 SYS 8192
50 RREG A,X,Y,Z,S
60 PRINT "REGISTER:";A;X;Y;Z;S

READY.
```

RSPCOLOR

Token: $CE $07

Format: **RSPCOLOR**(Index)

Zweck: Gibt Farben eines mehrfarbigen Sprites zurück.

Index Farbindex des mehrfarbigen Sprites:

- **1** liefert Farbe Nummer 1.
- **2** liefert Farbe Nummer 2.

Notiz: Weitere Informationen finden Sie unter **SPRITE** und **SPRCOLOR**.

Beispiel: Verwendung von **RSPCOLOR**:

```
10 REM RSPCOLOR
20 SPRITE 1,1          : REM AKTIVIERE SPRITE 1
30 C1% = RSPCOLOR(1)   : REM LESE FARBE NUMMER 1
40 C2% = RSPCOLOR(2)   : REM LESE FARBE NUMMER 2
50 PRINT C1%,C2%       : REM GIB DIE FARBWERTE AUS

READY.
RUN
 1         2

READY.
```

RSPEED

Token: $CE $0E

Format: **RSPEED**(Dummy)

Zweck: Gibt die aktuelle CPU-Geschwindigkeit in MHz zurück.

Dummy ist ein numerisches Wert, der ignoriert wird.

Notiz: **RSPEED** gibt nicht den korrekten Wert zurück, wenn zuvor **POKE 0,65** verwendet wurde, um die höchste Geschwindigkeit (40 MHz) zu aktivieren.

Weitere Informationen finden Sie unter dem Befehl **SPEED**.

Beispiel: Verwendung von **RSPEED**:

```
10 REM RSPEED
20 X=RSPEED(0)  : REM LESE DIE AKTUELLE SYSTEM-GESCHWINDIGKEIT
30 IF X=1  THEN PRINT "1 MHZ"   : GOTO 60
40 IF X=3  THEN PRINT "3.5 MHZ" : GOTO 60
50 IF X=40 THEN PRINT "40 MHZ"
60 END

READY.
SPEED 3

READY.
RUN
3.5 MHZ

READY.
```

RSPPOS

Token: $CE $05

Format: **RSPPOS**(Sprite, Parameter)

Zweck: Gibt die Position und Geschwindigkeit eines Sprites zurück.

Sprite ist die Nummer des Sprites (0-7).

Parameter ist der Sprite-Parameter zum Abrufen:

- **0** X-Koordinate der aktuellen Position
- **1** Y-Koordinate der aktuellen Position
- **2** Geschwindigkeit

Notiz: Weitere Informationen finden Sie unter **MOVSPR** und **SPRITE**.

Beispiel: Verwendung von **RSPPOS**:

```
10 REM RSPPOS
20 SPRITE 1,1          : REM AKTIVIERE SPRITE 1
30 XP = RSPPOS(1,0)    : REM LESE DIE X-KOORDINATE VON SPRITE 1
40 SP = RSPPOS(1,2)    : REM LESE DIE GESCHWINDIGKEIT VON SPRITE 1
50 PRINT XP,SP

READY.
RUN
 0          0

READY.
```

RSPRITE

Token: $CE $06

Format: **RSPRITE**(Sprite**,** Parameter**)**

Zweck: Gibt den angegebenen **Parameter** eines Sprites zurück.

Sprite ist die Nummer des Sprites (0-7).

Parameter ist der Sprite-Parameter zum Abrufen folgender Werte:

- **0** Sprite ist an (1) oder aus (0).
- **1** Vordergrundfarbe (0-15)
- **2** Hintergrundpriorität (0 oder 1)
- **3** Sprite ist in x-Richtung erweitert (1) oder nicht (0).
- **4** Sprite ist in y-Richtung erweitert (1) oder nicht (0).
- **5** Sprite ist mehrfarbig (1) oder nicht (0).

Notiz: Weitere Informationen finden Sie unter **MOVSPR** und **SPRITE**.

Beispiel: Verwendung von **RSPRITE**:

```
10 REM RSPRITE
20 SPRITE 1,1          : REM AKTIVIERE SPRITE 1
30 EN = RSPRITE(1,0)   : REM SPRITE 1 AKTIVIERT?
40 FG = RSPRITE(1,1)   : REM SPRITE 1 VORDERGRUND-FARBENINDEX
50 BP = RSPRITE(1,2)   : REM SPRITE 1 HINTERGRUNDPRIORITAET
60 XE = RSPRITE(1,3)   : REM SPRITE 1 X-ERWEITERT?
70 YE = RSPRITE(1,4)   : REM SPRITE 1 Y-ERWEITERT?
80 MC = RSPRITE(1,5)   : REM SPRITE 1 MEHRFARBIG?

READY.
```

RUN

Token: $8A

Format: **RUN** [Zeilennummer]
RUN Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: Startet die Ausführung eines BASIC-Programms.

Wenn ein Dateiname angegeben wird, wird die Programmdatei in den Speicher geladen und ausgeführt, andernfalls wird das Programm, das sich gerade im Speicher befindet, gestartet.

Zeilennummer ist eine existierende Zeilennummer des aktuell im Speicher befindlichen Programms, von der die Ausführung gestartet werden soll.

Dateiname ist entweder ein String in Anführungszeichen, zum Beispiel **"PROG"** oder ein String-Ausdruck in Klammern, zum Beispiel **(PR$)**. Der Dateityp muss **PRG** sein.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

RUN setzt zunächst alle internen Zeiger auf ihre Standardwerte zurück. Es sind also beim Programmstart keine Variablen, Arrays oder Strings definiert. Der Laufzeitstapel wurde ebenfalls zurückgesetzt und die Tabelle der geöffneten Dateien wurde gelöscht.

Notiz: Um die Programmausführung zu starten oder fortzusetzen, ohne alles zurückzusetzen, verwenden Sie stattdessen den **GOTO**-Befehl.

Beispiel: Verwendung von **RUN**:

```
RUN "FLIGHTSIM" : REM LAEDT UND STARTET DAS PROGRAMM "FLIGHTSIM" VON DISKETTE
RUN 1000        : REM STARTET DAS PROGRAMM IM SPEICHER AB ZEILE 1000
RUN             : REM STARTET DAS PROGRAMM IM SPEICHER
```

RWINDOW

Token: $CE $09

Format: **RWINDOW**(Parameter)

Zweck: Gibt Informationen über das aktuelle Textfenster zurück.

Parameter ist der Bildschirm-Parameter zum Abrufen folgender Werte:

- **0** Breite des aktuellen Textfensters
- **1** Höhe des aktuellen Textfensters.
- **2** Anzahl der Spalten auf dem Bildschirm (40 oder 80).

Notiz: Ältere Versionen von **RWINDOW** im BASIC 10 berichteten hiervon abweichend über die letztmögliche Spaltenposition (= Breite - 1) und die letztmögliche Zeilenposition (= Höhe - 1) für die Argumente 0 und 1.

Weitere Informationen finden Sie im **WINDOW**-Befehl.

Beispiel: Verwendung von **RWINDOW**:

```
10 REM RWINDOW
20 W = RWINDOW(2)           :REM LESE AKTUELLE BILDSCSHIRMBREITE
30 IF W=80 THEN BEGIN       :REM IST DER 80 SPALTENMODUS AKTIV?
40   PRINT CHR$(27)+"X";    :REM JA, SCHALTE EIN DEN 40 SPALTENMODUS
50 BEND

READY.
```

SAVE

Token: $94

Format: **SAVE** Dateiname [**,** Gerät]
← Dateiname [**,** Gerät]

Zweck: Speichert ein BASIC-Programm in eine Datei des Typs **PRG**.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Die maximale Länge des Dateinamens beträgt 16 Zeichen, wobei das optionale Zeichen **@** zum Speichern und Ersetzen und die Laufwerksdefinition in dem Dateinamen nicht mitgezählt werden. Wenn die ersten beiden Zeichen des Dateinamens "**@:**" sind, wird dies als "Speichern und Ersetzen"-Vorgang interpretiert. Es wird nicht empfohlen, diese Option auf den Laufwerken 1541 und 1571 zu verwenden, da sie einen "Speichern und Ersetzen"-Fehler in ihrem DOS enthalten. Dem Dateinamen kann die Definition der Laufwerksnummer **0:** oder **1:** vorangestellt werden, was aber nur bei Diskettenstationen mit zwei Laufwerken relevant ist.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **SAVE** ist in BASIC 65 implementiert, um die Abwärtskompatibilität zu BASIC 2 (Commodore 64) zu gewährleisten. Es sollte stattdessen **DSAVE** verwendet werden.

Der alternative Befehl mit dem Abkürzungssymbol ← (liegt links neben der **1**-Taste) kann nur im Direktmodus verwendet werden.

Beispiel: Verwendung von **SAVE**:

```
SAVE "ABENTEUER"
SAVE "ZORK-I",8
SAVE "1:LABYRINTH",9
```

SAVEIFF

Token: $FE $44

Format: **SAVEIFF** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: Speichert ein Bild aus dem Speicher in eine Diskettendatei im **IFF**-Format. Das IFF ("Interchange File Format", dt. etwa "Austausch-Dateiformat") wird von vielen verschiedenen Anwendungen und Betriebssystemen unterstützt. **SAVEIFF** speichert das Bild, die Palette und die Auflösungsparameter.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Die maximale Länge des Dateinamens beträgt 16 Zeichen, wobei das optionale Zeichen **@** zum Speichern und Ersetzen und die Laufwerksdefinition in dem Dateinamen nicht mitgezählt werden. Wenn das erste Zeichen des Dateinamens ein At-Zeichen **@** ist, wird es als "Speichern und Ersetzen"-Vorgang interpretiert. Es wird nicht empfohlen, diese Option auf den Laufwerken 1541 und 1571 zu verwenden, da sie einen "Speichern und Ersetzen"-Fehler in ihrem DOS enthalten.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Mit **SAVEIFF** gespeicherte Dateien können mit **LOADIFF** geladen werden.

Beispiel: Verwendung von **SAVEIFF**:

```
10 REM SAVEIFF
20 SCREEN 320,200,2          :REM OEFFNE BILDSCSCHIRM 0:320 X 200 X 2
30 PEN 1                     :REM WAEHLE ZEICHENSTIFT 1 (WEISS)
40 LINE 25,25,295,175        :REM ZEICHNE LINIE
50 SAVEIFF "SAVEIFF-BSP",U8  :REM SPEICHERE AKTUELLE ANSICHT IN DATEI
60 SCREEN CLOSE              :REM SCHLIESSE SCHIRM UND SETZE PALLETTE ZURUECK

READY.
```

SCNCLR

Token: $E8

Format: **SCNCLR** [Farbe]

Zweck: Löscht ein Textfenster oder einen Bildschirm.

SCNCLR ohne Argumente löscht das aktuelle Textfenster. Das Standardfenster nimmt den gesamten Bildschirm ein.

SCNCLR Farbe löscht den Grafikbildschirm, indem er ihn mit der angegebenen Farbe füllt.

Beispiel: Verwendung von **SCNCLR**:

```
100 REM SCNCLR
110 GRAPHIC CLR           :REM INITIALISIERE GRAFIK
120 SCREEN DEF 1,0,0,2    :REM DEFINIERE GRAFIKSCHIRM 1: 320 X 200 X 2
130 SCREEN OPEN 1         :REM OEFFNE GRAFIKSCHIRM 1
140 SCREEN SET 1,1        :REM DRAWSCREEN = VIEWSCREEN = 1
150 SCREEN CLR 0          :REM LOESCHE GRAFIKBILDSCHIRM
160 PALETTE 1,1,15,15,15  :REM DEFINIERE FARBE 1 ALS WEISS
170 PEN 0,1               :REM ZEICHENSTIFT
180 LINE 25,25,295,175    :REM ZEICHNE LINIE
190 SLEEP 10              :REM WARTE 10 SEKUNDEN
200 SCREEN CLOSE 1        :REM SCHLIESSE SCHIRM UND SETZE PALETTE ZURUECK

READY.
```

SCRATCH

Token: $F2

Format: **SCRATCH** Dateiname [**,D** Laufwerk] [**,U** Gerät] [**,R**]

Zweck: **SCRATCH** (dt. *"kratzen"*) wird verwendet, um eine Datei auf der Diskette zu löschen.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

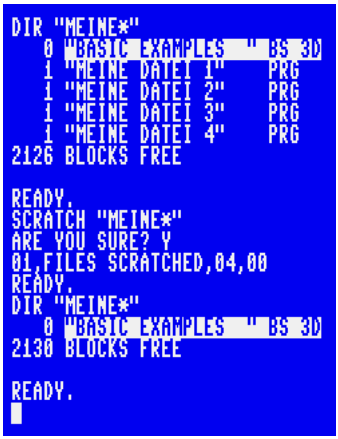
R Wiederherstellen einer zuvor gelöschten Datei. Dies funktioniert nur, wenn zwischen dem Löschen und der Wiederherstellung keine Schreibvorgänge stattgefunden haben, die den Inhalt der Diskette verändert haben könnten.

Notiz: **SCRATCH** arbeitet ähnlich wie **DELETE Dateiname** und **ERASE**.

Nach Ausführung gibt **SCRATCH** automatisch die Systemvariable **DS$** aus, die Erfolg und Anzahl der gelöschten Dateien zeigt. Die vorletzte Zahl, die im Falle eines Ein-/Ausgabefehlers normalerweise die Spurnummer angibt, gibt stattdessen die Anzahl der erfolgreich gelöschten Dateien an.

Im Direktmodus wird bei **SCRATCH** vor der Befehlsausführung zur Sicherheit mit `ARE YOU SURE?` nachgefragt, ob Sie wirklich löschen wollen? Bestätigen Sie diese mit `Y` (für "Yes", dt. "Ja"). Alle anderen Buchstaben brechen den Vorgang ab.

Beispiel: Verwendung von **SCRATCH**:



SCREEN

Token: $FE $2E

Format: **SCREEN** [Nummer,] Breite, Höhe, Farbtiefe
SCREEN CLR Farbe
SCREEN DEF Nummer, Breiten-Flag, Höhen-Flag, Farbtiefe
SCREEN SET Drawscreen, Viewscreen
SCREEN OPEN [Nummer]
SCREEN CLOSE [Nummer]

Zweck: Bei der Vorbereitung des Bildschirms für das Zeichnen von Grafiken gibt es folgende zwei Varianten: Eine vereinfachte Variante und eine detaillierte Variante.

Vereinfachte Variante:

Die erste Variante von **SCREEN** (mit Pixeleinheiten für Breite und Höhe) ist der einfachste Weg, einen Grafikbildschirm zu starten, und die bevorzugte Methode, wenn nur ein einziger Bildschirm benötigt wird (das heißt, ein zweiter Bildschirm wird nicht für die doppelte Pufferung benötigt). Dieses Programm übernimmt alle vorbereitenden Arbeiten für Sie und ruft selbstständig Befehle wie **GRAPHIC CLR**, **SCREEN CLR**, **SCREEN DEF**, **SCREEN OPEN** und **SCREEN SET** auf. Es benötigt die folgenden Parameter:

SCREEN [Nummer,] Breite, Höhe, Farbtiefe

- **Nummer** ist die Nummer des Bildschirms (0-3). Die Angabe ist optional. Falls keine Nummer angegeben wird, wird standardmäßig die für diese Variante empfohlenene Nummer 0 verwendet.
- **Breite** 320 oder 640 (Standardwert: 320)
- **Höhe** 200 oder 400 (Standardwert: 200)
- **Farbtiefe** 1 bis 8 (Standardwert: 8), Farben = 2^Farbtiefe

Der Argumentparser ist fehlertolerant und verwendet Standardwerte für Breite (320) und Höhe (200), wenn das geparste Argument nicht gültig ist.

Diese Variante von **SCREEN** beginnt mit einer vordefinierten Palette, setzt den Hintergrund auf Schwarz und den Stift auf Weiß, so dass das Zeichnen sofort mit den Standardwerten beginnen kann.

Der Wert von **Farbe** muss im Bereich von 0 bis 15 liegen. In der Farbtabelle unter **BACKGROUND** auf Seite 19 finden Sie die Werte für **Farbe** und die entsprechenden Farben.

Wenn Sie mit Ihrem Grafikbildschirm fertig sind, rufen Sie **SCREEN CLOSE** [Nummer] auf, um zum Textbildschirm zurückzukehren.

Detaillierte Variante:

Die andere Variante von **SCREEN** führt spezielle Aktionen aus, die für fortgeschrittene Grafikprogramme verwendet werden, die mehrere Bildschirme öffnen oder "double buffering" (dt. "doppelte Pufferung") erfordern. Wenn Sie sich für die vereinfachten Variante entschieden haben, benötigen Sie mit Ausnahme von **SCREEN CLOSE** keine der folgenden Befehlsvarianten.

SCREEN CLR Farbe (oder **SCNCLR** Farbe)
Löscht den aktiven Grafikbildschirm, indem er mit der Farbe **Farbe** gefüllt wird.

SCREEN DEF Nummer**,** Breiten-Flag**,** Höhen-Flag**,** Farbtiefe
Legt die Auflösungsparameter für den gewählten Bildschirm fest. Das Breiten-Flag und das Höhen-Flag geben an, ob eine hohe (1) oder eine niedrige Auflösung (0) gewählt wird.

- **Nummer** ist die Nummer des Bildschirms (0-3).
- **Breiten-Flag** 0-1 (0:320 Pixel, 1:640 Pixel)
- **Höhen-Flag** 0-1 (0:200 Pixel, 1:400 Pixel)
- **Farbtiefe** 1-8 (2 - 256 Farben)

Beachten Sie, dass es sich bei den Werten für Breite und Höhe um **Flags** (0 oder 1) und **und nicht um Pixel-Einheiten** handelt.

SCREEN SET Drawscreen**,** Viewscreen
Legt die Bildschirmnummern (0-3) für den sogenannten Drawscreen und Viewscreen (dt. in etwa "Zeichnen-Bildschirm"und "Ansichts-Bildschirm") fest. Das heißt, während ein Bildschirm angezeigt wird (Viewscreen), können Sie auf einem anderen Bildschirm zeichnen lassen (Drawscreen) und später zwischen den beiden Bildschirmen wechseln. Dieses Technik wird auch als "double buffering" (dt. "Doppelpufferung") bezeichnet.

SCREEN OPEN Nummer
Weist Ressourcen zu und initialisiert den Grafikkontext für den ausgewählten Bildschirm **Nummer** (0-3).

SCREEN CLOSE [Nummer]
Schließt Bildschirm **Nummer** (0-3) und gibt die dafür gebundenen Ressourcen frei. Wenn kein Wert angegeben ist, wird standardmäßig Bildschirm 0 geschlossen. Beachten Sie auch, dass beim Schließen ei-

nes Bildschirms automatisch der Befehl **PALETTE RESTORE** durchgeführt wird.

Beispiel: Verwendung von **SCREEN**:

```
100 REM SCREEN
110 REM *** VEREINFACHTE VARIANTE ***
120 SCREEN 320,200,2    : REM BILDSCHIRM NUMMER 0: 320 X 200 X 2
130 PEN 1               : REM ZEICHENSTIFTFARBE 1 (WEISS)
140 LINE 25,25,295,175  : REM ZEICHNE LINIE
150 GETKEY A$           : REM WARTE AUF TASTENDRUCK
160 SCREEN CLOSE        : REM SCHLIESSE BILDSCHIRM 0 UND SETZE PALETTE ZURUECK

READY.
```

```
100 REM SCREEN
110 REM *** DETAILLIERTE VARIANTE ***
120 GRAPHIC CLR          : REM INITIALISIERE GRAFIK
130 SCREEN DEF 1,0,0,2   : REM BILDSCHIRM NUMMER 1: 320 X 200 X 2
140 SCREEN OPEN 1        : REM OEFFNE BILDSCHIRM 1
150 SCREEN SET 1,1       : REM VERWENDE SCHIRM 1 FUER ZEICHNEN UND ANSICHT
160 SCREEN CLR 0         : REM LOESCHE BILDSCHIRM
170 PALETTE 1,1,15,15,15 : REM DEFINIERE FARBE 1 ALS WEISS
180 PEN 0,1              : REM ZEICHENSTIFT
190 LINE 25,25,295,175   : REM ZEICHNE LINIE
200 SLEEP 10             : REM WARTE 10 SEKUNDEN
210 SCREEN CLOSE 1       : REM SCHLIESSE BILDSCHIRM 1 UND SETZE PALETTE ZURUECK

READY.
```

SET

Token: $FE $2D

Format: **SET DEF** Gerät
SET DISK Alt **TO** Neu
SET VERIFY <**ON** | **OFF**>
SET BIT Adresse**,** Bit

Zweck: **SET DEF** definiert die Standardeinheit für den Diskettenzugriff neu. Nach dem Einschalten des MEGA65 ist dieser Wert auf 8 gesetzt. Befehle, die nicht ausdrücklich eine Einheit angeben, verwenden diese Standardeinheit.

SET DISK wird verwendet, um die Gerätenummer eines Laufwerks vorübergehend zu ändern.

SET VERIFY aktiviert oder deaktiviert im DOS den sogenannten *"verify-after-write-*Modus" (dt. in etwa *"Nach dem Schreiben überprüfen"*-Modus) für 3,5 Zoll-Diskettenlaufwerke. Dies bedeutet, dass Dateien nach dem Schreiben auf die Diskette automatisch auf korrekte Übertragung geprüft werden.

SET BIT setzt das angegebene Bit in der angegebenen Adresse (siehe dazu **BIT** auf Seite 24).

Notiz: Diese Einstellungen sind bis zu einem Reset oder dem Ausschalten des MEGA65 gültig.

Beispiel: Verwendung von **SET**:

```
DIR             : REM ZEIGE DATEIVERZEICHNIS VON GERAET 8
SET DEF 11      : REM SETZE GERAET 11 ALS STANDARD
DIR             : REM ZEIGE DATEIVERZEICHNIS VON GERAET 11
DLOAD "*"       : REM LADE DIE ERSTE DATEI VON GERAET 11
SET DISK 8 TO 9 : REM AENDERE GERAETENUMMER DES DISK-LAUFWERK VON 8 AUF 9
DIR U9          : REM ZEIGE DATEIVERZEICHNIS VON GERAET 9 (VORHER 8)
SET VERIFY ON   : REM AKTIVIERE "VERIFY-AFTER-WRITE"-MODUS
```

SGN

Token: $B4

Format: **SGN(**numerischer Ausdruck**)**

Zweck: Ermittelt das Vorzeichen des **numerischen Ausdruck**s und gibt es als Zahl zurück:

- **-1** der Ausdruck ist negativ (< 0)
- **-0** der Ausdruck ist Null (= 0)
- **1** der Ausdruck ist positiv (> 0)

Beispiel: Verwendung von **SGN**:

```
10 REM SGN
20 INPUT "X = ";X                : REM EINGABE VON X
30 ON SGN(X)+2 GOTO 100,200,300 : REM ZIELE FUER NEGATIV,NULL,POSITIV
40 :
100 REM NEGATIV
110 PRINT "X IST NEGATIV"
120 END
200 REM NULL
210 PRINT "X IST NULL"
220 END
300 REM PLUS
310 PRINT "X IST POSITIV"
320 END

READY.
```

SIN

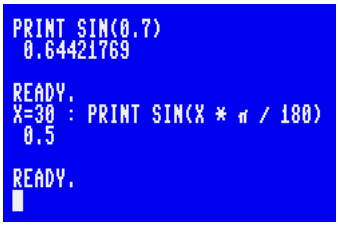
Token: $BF

Format: **SIN**(numerischer Ausdruck)

Zweck: Gibt den Sinus des **numerischen Ausdruck**s zurück. Das Argument wird im **Bogenmaß** erwartet. Das Ergebnis liegt im Bereich (-1.0 bis +1.0)

Notiz: Ein Argument in **Grad** kann ins benötigte **Bogenmaß** umgerechnet werden, indem es mit $\pi/180$ multipliziert wird.

Beispiel: Verwendung von **SIN**:



SLEEP

Token: $FE $0B

Format: **SLEEP** Sekunden

Zweck: **SLEEP** (dt. "schlafe") hält die Ausführung des Programms für die angegebene Dauer (in Sekunden) an. Das Argument ist eine positive Fließkommazahl. Die Genauigkeit beträgt 1 Mikrosekunde.

Notiz: Durch Drücken der RUN STOP-Taste wird der Ruhezustand unterbrochen.

Beispiel: Verwendung von **SLEEP**:

```
10 REM SLEEP
20 SLEEP 10     : REM WARTE 10 SEKUNDEN
40 SLEEP 0.0005 : REM WARTE 500 MIKROSEKUNDEN
50 SLEEP 0.01   : REM WARTE 10 MILLISEKUNDEN
60 SLEEP DD     : REM VERWENDET DEN WERT DER VARIABLE DD ALS PAUSENWERT
70 SLEEP 600    : REM WARTE 10 MINUTEN

READY.
```

SOUND

Token: $DA

Format: **SOUND** Stimme**,** Frequenz**,** Dauer [{**,** Richtung**,** Minimum**,** Sweep**,** Wellenform **,** Impulsweite}]

Zweck: Spielt einen Soundeffekt ab.

Stimme Stimmennummer (1-6).

Frequenz Frequenz (0-65535).

Dauer ein Wert für die Dauer des Effekts (0-32767). Die tatsächliche Dauer hängt von der Bild-Darstellungsnorm ab. Bei PAL (wie in Deutschland üblich) entspricht ein Wert von 50 einer Sekunde, bei NTSC entspricht ein Wert von 60 einer Sekunde.

Richtung Richtung (0:nach oben, 1:nach unten, 2:oszillieren).

Minimum Mindestfrequenz (0-65535).

Sweep Sweep-Bereich (0-65535).

Wellenform Wellenform (0:Dreieck, 1:Sägezahn, 2:Rechteck, 3:Rauschen).

Impulsweite Impulsweite (0-5095).

Remarks: **SOUND** startet die Wiedergabe des Soundeffekts und fährt sofort, während der Soundeffekt abgespielt wird, mit der Ausführung der nächsten BASIC-Anweisung fort. Dies ermöglicht die gleichzeitige Anzeige von Grafiken oder Text und das Abspielen von Sounds.

Beispiel: Verwendung von **SOUND**:

```
10 REM SOUND
15 REM SPIELE RECHTECKWELLE AUF STIMME 1 FUER 1 SEKUNDE AB
20 SOUND 1, 7382, 50
25 REM SPIELE RECHTECKWELLE AUF STIMME 2 FUER 1 MINUTE AB
30 SOUND 2,  800, 50*60
35 REM SAEGEZAHNSCHWINGUNG AUF STIMME 3 ABSPIELEN
40 SOUND 3, 4000, 120, 2, 2000, 400, 1

READY.
```

SPC

Token: $A6

Format: **SPC**(Anzahl)

Zweck: Überspringt **Anzahl** Spalten bei der Ausgabe.
Dies entspricht einem **Anzahl**-maligem Drücken der → -Taste.

Notiz: Der Name dieser Funktion ist von **SPACES** (dt. "Leerzeichen") abgeleitet, was irreführend ist. Die Funktion gibt **Cursorbewegungen nach rechts** aus, keine Leerzeichen. Die damit übersprungenen Zeichen werden nicht geändert.

Beispiel: Verwendung von **SPC**:

```
10 REM SPC
20 FOR I=8 TO 12
30 PRINT SPC(-(I(10));I :REM WAHR = -1, FALSCH = 0
40 NEXT I

READY.
RUN
  8
  9
 10
 11
 12

READY.
```

SPEED

Token: $CE $0E

Format: **SPEED** [Geschwindigkeit]

Zweck: Setzt die CPU-Geschwindkeit auf 1 MHz, 3.5 MHz oder 40 MHz.

Geschwindigkeit ist die CPU-Geschwindigkeit, wobei:

- **1** die CPU-Geschwindigkeit auf 1 MHz setzt.
- **3** die CPU-Geschwindigkeit auf 3,5 MHz setzt.
- Jeder andere Wert als **1** und **3** setzt die CPU-Geschwindigkeit auf 40 MHz.

Notiz: Es ist zwar möglich, **SPEED** mit einer beliebigen reellen Zahl aufzurufen, allerdings werden hierbei der Dezimalpunkt und alle Nachkommastellen ignoriert.

SPEED ist ein Synonym für **FAST**.

SPEED hat keinen Effekt, falls vorher **POKE 0,65** verwendet wurde, um die CPU-Geschwindigkeit auf 40 MHz zu setzen.

Beispiel: Verwendung von **SPEED**:

```
10 REM SPEED
20 SPEED      : REM SETZE DIE GESCHWINDIGKEIT AUF 40 MHZ (MAXIMUM)
30 SPEED 1    : REM SETZE DIE GESCHWINDIGKEIT AUF 1 MHZ
40 SPEED 3    : REM SETZE DIE GESCHWINDIGKEIT AUF 3.5 MHZ
50 SPEED 3.5  : REM SETZE DIE GESCHWINDIGKEIT AUF 3.5 MHZ

READY.
```

SPRCOLOR

Token: $FE $08

Format: **SPRCOLOR** [{Farbe1**,** Farbe2}]

Zweck: Legt die Farben der mehrfarbigen Sprites fest.

Der **SPRITE**-Befehl, der die Attribute eines Sprites setzt, setzt nur die Vordergrundfarbe. Für die Einstellung der zusätzlichen zwei Farben von mehrfarbigen Sprites verwenden Sie stattdessen **SPRCOLOR**.

Notiz: Die mit **SPRCOLOR** angegebenen Farben wirken sich auf alle Sprites aus. Siehe den Befehl **SPRITE** für weitere Informationen.

Beispiel: Verwendung von **SPRCOLOR**:

```
10 REM SPRCOLOR
20 SPRITE 1,1,2,,,,1  : REM SCHALTE SPRITE 1 EIN (VORDERGRUNDFARBE = 2)
30 SPRCOLOR 4,5       : REM FARBE1 = 4, FARBE2 = 5

READY.
```

SPRITE

Token: $FE $07

Format: **SPRITE CLR**
SPRITE LOAD Dateiname [**,D** Laufwerk] [**,U** Gerät]
SPRITE SAVE Dateiname [**,D** Laufwerk] [**,U** Gerät]
SPRITE Nummer [{**,** Schalter**,** Farbe**,** Priorität**,** x-Expansion**,** y-Expansion**,** Modus}]

Zweck: **SPRITE CLR** löscht alle Sprite-Daten und setzt alle Zeiger und Attribute auf ihre Standardwerte.

SPRITE LOAD lädt Sprite-Daten aus der Datei mit dem Namen **Dateiname** in den Sprite-Speicher.

SPRITE SAVE speichert Sprite-Daten aus dem Sprite-Speicher in die Datei mit dem Namen **Dateiname**.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Die letzte Variante des **SPRITE**-Befehls schaltet ein Sprite ein oder aus und setzt seine Attribute:

Nummer ist die Spritenummer

Schalter 1: An, 0: Aus

Farbe ist die Farbe des Sprite-Vordergrunds

Priorität Sprite (1)- oder Bildschirm (0)-Priorität

x-Expansion 1: Expansion des Sprites in x-Richtung

y-Expansion 1: Expansion des Sprites in y-Richtung

Modus 1: mehrfarbiges (multi-colour) Sprite

Notiz: Um zusätzliche Farben für mehrfarbige Sprites (Modus = 1) festzulegen, muss der Befehl **SPRCOLOR** verwendet werden.

Beispiel: Verwendung von **SPRITE**:

```
10 REM SPRITE
20 CLR:SCNCLR:SPRITE CLR
30 SPRITE LOAD "DEMOSPRITES1"
40 FOR I=0 TO 7: C=I: IF C=6 THEN C=8
50 MOVSPR I, 60+30*I,0 TO 60+30*I,65+20*I, 3:SPRITE I,1,C,,1,1:NEXT: SLEEP 3
60 FOR I=0 TO 7: SPRITE I,,,,0,0 :NEXT: SLEEP 3: SPRITE CLR
70 FOR I=0 TO 7: MOVSPR I,45*I#5 :NEXT: FOR I=0 TO 7: SPRITE I,1: NEXT
80 FOR I=0 TO 7:X=60+30*I:Y=65+20*I:DO
90 LOOP UNTIL (X=RSPPOS(I,.)) AND (Y=RSPPOS(I,1)):MOVSPR I,.#.: NEXT

READY.
```

SPRSAV

Token: $FE $16

Format: **SPRSAV** Quelle**,** Ziel

Zweck: Kopiert Spritedaten.

Quelle ist eine Spritenummer oder String-Variable.

Ziel ist eine Spritenummer oder String-Variable.

Notiz: Quelle und Ziel können entweder eine Spritenummer oder eine String-Variable sein, jedoch können nicht beide gleichzeitig eine String-Variable sein. Für solche Fälle kann eine einfache String-Zuweisung verwendet werden.

SPRSAV kann nur mit der Grundform von Sprites (C64-kompatibel) verwendet werden. Diese Sprites haben eine Größe von 64 Byte, und ihr Speicherbedarf beträgt 67 Byte.

Erweiterte Sprites und Sprites mit variabler Höhe können nicht als Argumente für **SPRSAV** verwendet werden.

Beispiel: Verwendung von **SPRSAV**:

```
10 REM SPRSAV
20 BLOAD "SPRITEDATA",P1600    : REM LADE DATEN FUER SPRITE NUMMER 1
30 SPRITE 1,1                  : REM AKTIVIERE SPRITE 1
40 SPRSAV 1,2                  : REM KOPIERE DATEN SPRITE 1 NACH SPRITE 2
50 SPRITE 2,1                  : REM AKTIVIERE SPRITE 2
60 SPRSAV 1,A$                 : REM SPEICHERE DATEN SPRITE 1 IN STRING A$

READY.
```

SQR

Token: $BA

Format: **SQR**(numerischer Ausdruck)

Zweck: Gibt die Quadratwurzel des numerischen Ausdrucks zurück.

Notiz: Das Argument darf nicht negativ sein.

Beispiel: Verwendung von **SQR**:

```
10 REM SQR
20 PRINT SQR(2)

READY.
RUN
 1.4142136

READY.
```

ST

Format: **ST**

Zweck: **ST** enthält den Status der letzten Ein-/Ausgabe-Operation.

Falls **ST** gleich Null ist, lag kein Fehler vor, andernfalls wird **ST** auf einen geräteabhängigen Fehlercode gesetzt.

Notiz: **ST** ist eine reservierte Systemvariable.

Beispiel: Verwendung von **ST**:

```
10 REM ST
20 MX=100:DIM T$(MX)          : REM DATEN-ARRAY
30 DOPEN#1,"DATEN"            : REM OEFFNE DATEI
40 IF DS THEN PRINT"KONNTE NICHT OEFFNEN" : STOP
50 LINE INPUT#1,T$(N):N=N+1   : REM LESE EINEN EINTRAG
60 IF N>MX THEN PRINT "ZU VIELE DATEN" : GOTO 80
70 IF ST=0 THEN 50: REM ST = 64 FALLS DATEIENDE
80 DCLOSE#1
90 PRINT "GELESEN: ";N;" EINTRAEGE"

READY.
```

STEP

Token: $A9

Notiz: **STEP** ist ein Schlüsselwort, das nur in Kombination mit **FOR** verwendet wird.

Näheres siehe bei **FOR** auf Seite 110.

STOP

Token: $90

Format: **STOP**

Zweck: Stoppt die Ausführung des BASIC-Programms. Es wird eine Meldung mit der Zeilennummer angezeigt, in der das Programm angehalten wurde. Die Eingabeaufforderung **READY.** erscheint und der Computer geht in den Direktmodus über und wartet auf Tastatureingaben.

Notiz: Alle Variablendefinitionen sind nach **STOP** noch gültig. Sie können eingesehen oder geändert werden, und das Programm kann mit **CONT** fortgesetzt werden. Jegliche Bearbeitung des Programmquelltextes macht jedoch eine weitere Fortsetzung unmöglich.

Beispiel: Verwendung von **STOP**:

```
10 REM STOP
20 INPUT V
30 IF V < 0 THEN STOP : REM NEGATIVE ZAHLEN STOPPEN DAS PROGRAMM
40 PRINT SQR(V)       : REM GIB DIE QUADRATWURZEL VON V AUS

READY.
```

STR$

Token: $C4

Format: **STR$**(numerischer Ausdruck)

Zweck: Gibt einen String zurück, der den formatierten Wert des Arguments enthält, so als ob er mit **PRINT** in den String ausgegeben worden wäre.

Notiz: Mit **STR$** ist es zum Beispiel möglich, eine Zahl in einen String umzuwandeln.

Beispiel: Verwendung von **STR$**:

```
10 REM STR$
20 A$ = "DER WERT VON PI IST" + STR$(π)
30 PRINT A$

READY.
RUN
DER WERT VON PI IST 3.1415927

READY.
```

SYS

Token: $9E

Format: **SYS** Adresse [,[a-Register] [,[x-Register] [,[y-Register] [,[z-Register] [,Statusregister]]]]]

Zweck: **SYS** ruft ein Maschinensprache-Unterprogramm auf. Dabei kann es sich um eine Systemroutine oder um ein anderes Programm handeln, das zuvor in den Arbeitsspeicher geladen oder mittels **POKE** erstellt worden ist.

Die CPU-Register werden mit den Argumenten geladen (falls sie angegeben sind), dann wird ein Unterprogrammaufruf (**JSR Adresse**) durchgeführt. **JSR** ist ein Assembler-Befehl, der für die Abkürzung für **J**ump to **S**ub**R**outine (dt. "Springe zum Unterprogramm") steht. Die aufgerufene Routine sollte mit einer **RTS**-Anweisung beendet werden. **RTS** ist ein weiterer Assembler-Befehl, der die Abkürzung für **R**e**t**urn from **S**ubroutine (dt. "Kehre vom Unterprogramm zurück") darstellt. Nach der Rückkehr aus dem mit **SYS** aufgerufenen Assembler-Programm werden die Registerinhalte gespeichert und die Ausführung des BASIC-Programms wird fortgesetzt.

Adresse ist die Startadresse des Unterprogramms.

Ist der Adresswert 16 Bit ($0000 - $FFFF), wird der aktuell gültige Bankwert (siehe **BANK**) untersucht. Ein Bankwert von 128 lässt das aktuelle Mapping bestehen. D.h.: RAM ist nur im Adressbereich ($0000 - $1FFF) verfügbar, während BASIC-ROM, KERNAL und I/O den Rest belegen. Kurze Maschinensprachprogramme können den Adressbereich ($1800 - $1FFF) verwenden, der nur von BASIC verwendet wird, wenn ein Grafikbildschirm geöffnet ist.

Ist die Adresse höher als $FFFF, wird sie als lineare 24-Bit-Adresse interpretiert und der Wert von **BANK** wird ignoriert.

Die anfängliche Zuordnung ist in der folgenden Tabelle dargestellt:

Bereich	Inhalt
0000 - 1FFF	Bank 0 mit direkten Seiten-, Stack-, Vektoren- und Interface-Routinen
2000 - BFFF	Ausgewählte RAM-Bank: Adressbits 16-23
C000 - CFFF	Kernal-ROM
D000 - DFFF	Ein-/Ausgabe (I/O)
E000 - EFFF	Editor-ROM
F000 - FFFF	Kernal-ROM und Sprungtabelle (jump table)

Die RAM-Bänke 0, 1, 4 und 5 können beim MEGA65 mit dem Befehl **SYS** verwendet werden. Das Attic-RAM kann für diesen Zweck nicht verwendet werden, da die 24-Bit-Adresse des Befehls **SYS** auf die unteren 16 MB des Adressbereichs beschränkt ist.

a-Register ist der Wert, der beim Aufruf an den Akkumulator (A-Register) übergeben wird.

x-Register ist der Wert, der beim Aufruf an das X-Register übergeben wird.

y-Register ist der Wert, der beim Aufruf an das Y-Register übergeben wird.

z-Register ist der Wert, der beim Aufruf an das Z-Register übergeben wird.

Statusregister ist der Wert, der beim Aufruf an das Statusregister übergeben wird.

Notiz: Die Registerwerte werden nach einem **SYS**-Aufruf im Systemspeicher abgelegt. Auf diese Weise kann **RREG** sie abfragen.

SYS bildet die Adressen $2000 - $FFFF auf die mit **BANK** eingestellte Bank ab, so dass Sie Unterprogramme in jeder Bank aufrufen können. Die Adressen $0000 - $1FFF werden nicht abgebildet und greifen immer auf das RAM in Bank 0 zu. Das bedeutet, dass Sie Unterprogramme an den Adressen $0000 - $1FFF nur von **BANK 0** und **BANK 128** aufrufen können.

Der Befehl **SYS** auf dem MEGA65 ist völlig anders als der bekannte Befehl **SYS** auf dem C64.

Es ist nicht möglich, das BASIC-ROM und ein BASIC-Programm im selben Mapping zu haben, da sie denselben Adressbereich belegen.

Die korrekte Verwendung von **SYS** (d.h. ohne das System zu beeinflussen) erfordert einige technische Kenntnisse, die den Rahmen des Benutzerhandbuchs sprengen würden. Wenn Sie jedoch mehr darüber erfahren möchten, finden Sie viele weitere Informationen und Beispiele im (englischsprachigen) "MEGA65 Developer's Guide".

Beispiel: Verwendung von **SYS**:

```
10 REM SYS
20 BANK 128
30 BLOAD "MSPR-PRG",8192
40 SYS 8192
50 RREG A,X,Y,Z,S
60 PRINT "REGISTER:";A;X;Y;Z;S

READY.
```

```
10 REM DEMO FUER SYS: RAHMENFARBE AENDERN
20 BANK 0
30 POKE $4000,$EE,$20,$D0,$60 :REM INC $D020:RTS
40 SYS $4000                  :REM RUFE ROUTINE BEI $4000 / BANK $00 AUF
50 GETKEY A$:IF A$ <> "Q" THEN 40

READY.
```

TAB

Token: $A3

Format: **TAB**(Spalte)

Zweck: Positioniert den Cursor an der Spaltennummer **Spalte**.
Dies geschieht nur, wenn die Zielspalte rechts von der aktuellen Cursor-Spalte liegt, andernfalls wird der Cursor nicht bewegt. Die Spaltenzählung beginnt am linken Rand mit der Spalte 0.

Notiz: Diese Funktion ist nicht zu verwechseln mit der TAB-Taste, die den Cursor zur nächsten Tabulatorposition bewegt.

Beispiel: Verwendung von **TAB**:

```
10 REM TAB
20 FOR I=1 TO 5
30 READ A$
40 PRINT "* " A$ TAB(10) " *"
50 NEXT I
60 END
70 DATA EINS,ZWEI,DREI,VIER,FUENF

READY.
RUN
* EINS     *
* ZWEI     *
* DREI     *
* VIER     *
* FUENF    *

READY.
```

TAN

Token: $C0

Format: **TAN**(numerischer Ausdruck)

Zweck: Gibt den Tangens des Arguments zurück. Das Argument wird in der Einheit Bogenmaß erwartet. Das Ergebnis liegt im Bereich (-1.0 bis +1.0)

Notiz: Ein Argument in Grad-Einheiten kann durch Multiplikation mit $\pi/180$ in Bogenmaß umgewandelt werden.

Beispiel: Verwendung von **TAN**:

```
10 REM TAN
20 PRINT TAN(0.7)
30 X=45:PRINT TAN(X * π / 180)

READY.
RUN
 0.84228838
 0

READY.
```

TEMPO

Token: $FE $05

Format: **TEMPO** Geschwindigkeit

Zweck: Legt die Wiedergabegeschwindigkeit für **PLAY** fest.

Geschwindigkeit ist ein Wert im Bereich 1-255.

Die Dauer (in Sekunden) einer ganzen Note wird mit

$$Dauer = 24/Geschwindigkeit$$

berechnet.

Beispiel: Verwendung von **TEMPO**:

```
10 REM TEMPO
20 VOL 8
30 FOR T = 24 TO 18 STEP -2
40 TEMPO T
50 PLAY "T0M3O4QGAGFED","T2O4M5P0H.DP5GB","T5O3IGAGAGAABABAB"
60 IF RPLAY(1) THEN GOTO 60
70 NEXT T
80 PLAY "T0O5QCO4GEH.C","T2O5IEFEDEDCEGO6P8CP0R","T5O3ICDCDEFEDCO4C"

READY.
```

THEN

Token: $A7

Notiz: **THEN** ist ein Schlüsselwort, das nur in Kombination mit **IF** verwendet wird.

Näheres siehe bei **IF** auf Seite 130.

TI

Format: **TI**

Zweck: **TI** ist ein hochpräziser Timer mit einer Auflösung von 1 Mikrosekunde.

Er wird mit **CLR TI** gestartet oder zurückgesetzt und kann wie jede andere Variable in Ausdrücken angesprochen werden.

Notiz: **TI** ist eine reservierte Systemvariable. Der Wert in **TI** ist die Anzahl der Sekunden (mit 6 Dezimalstellen) seit der letzten Löschung oder dem letzten Start.

Beispiel: Verwendung von **TI**:

```
10 REM TI
20 CLR TI                  : REM STARTE TIMER
30 FOR I%=1 TO 10000:NEXT  : REM MACH IRGENDWAS
40 ET = TI                 : REM SPEICHERE BENOETIGTE ZEIT IN VARIABLE ET
50 PRINT "AUSFUEHRUNGSZEIT:";ET;"SEKUNDEN"

READY.
RUN
AUSFUEHRUNGSZEIT:  0.13630764 SEKUNDEN

READY.
```

TI$

Format: **TI$**

Zweck: **TI$** speichert die Zeitinformationen der RTC (Real-Time Clock, dt. "Echtzeituhr") in Textform, wobei folgendes Format verwendet wird: "hh:mm:ss". Die Systemvariable wird bei jeder Verwendung aktualisiert.

TI$ ist eine schreibgeschützte Variable, die die Register der RTC ausliest und die Werte in ein String formatiert.

Notiz: **TI$** ist eine reservierte Systemvariable.

Es ist möglich, über **PEEK** direkt auf die RTC-Register zuzugreifen. Die Startadresse der Register liegt bei $FFD7110.

```
10 REM ****** LESE RTC ****** ALLE WERTE SIND BCD-CODIERT
20 RT = $FFD7110             :REM ADRESSE DER ECHTZEITUHR (RTC)
30 FOR I=0 TO 5              :REM SS,MM,HH,DD,MO,YY
40 T(I)=PEEK(RT+I)           :REM LESE REGISTER
50 NEXT I                    :REM VERWENDE NUR DIE LETZTEN ZWEI ZIFFERN
60 T(2) = T(2) AND 127       :REM ENTFERNE 24 STUNDEN-FLAG
70 T(5) = T(5) + $2000       :REM ADDIERE 2000 ZUM JAHR HINZU
80 FOR I=2 TO 0 STEP -1      :REM ZEIT INFO
90 PRINT USING ">##";HEX$(T(I));
100 NEXT I

READY.
RUN
10 57 46
READY.
```

Beispiel: Verwendung von **TI$**:

```
PRINT DT$;TI$
06-DEC-2021 10:59:11

READY.
```

TO

Token: $A4

Format: Schlüsselwort **TO**

Zweck: **TO** ist ein sekundäres Schlüsselwort, das in Kombination mit primären Schlüsselwörtern wie **BACKUP**, **BSAVE**, **CHANGE**, **CONCAT**, **COPY**, **FOR**, **GO**, **RENAME** und **SET DISK** verwendet wird.

Notiz: **TO** kann nicht allein verwendet werden.

Beispiel: Verwendung von **TO**:

```
10 REM TO
20 GO TO 70                 : REM FUER GOTO 70
30 GOTO 70                  : REM KUERZER UND SCHNELLER
40 FOR I=1 TO 10            : REM TO IST TEIL DER SCHLEIFE
50 PRINT I:NEXT             : REM SCHLEIFENENDE
60 COPY "CODE" TO "BACKUP" : REM KOPIERT EINE DATEI
70 REM ENDE

READY.
```

TRAP

Token: $D7

Format: **TRAP** [Zeilennummer]

Zweck: **TRAP** mit einer gültigen Zeilennummer aktiviert die BASIC-Fehlerbehandlungsroutine ab **Zeilennummer**. Wenn ein Programm einen Fehlerbehandlungsroutine aktiviert hat, ändert sich das Laufzeitverhalten. Normalerweise wird BASIC das Programm beenden und eine Fehlermeldung anzeigen.

Wenn jedoch eine BASIC-Fehlerbehandlungsroutine aktiviert wurde, speichert BASIC stattdessen den Ausführungszeiger und die Zeilennummer, legt die Fehlernummer in der Systemvariablen **ER** ab und springt zu der bei **TRAP** angegebenen **Zeilennummer**. Die **TRAP**-Routine kann **ER** untersuchen und den Fehler verarbeiten. Auf dieser Grundlage kann die Fehlerbehandlungsroutine dann entscheiden, ob sie die Ausführung stoppt (entsprechend **STOP**) oder fortsetzt (entsprechend **RESUME**).

TRAP ohne Argument deaktiviert die Fehlerbehandlungsroutine und Fehler werden nun wieder von den normalen Systemroutinen behandelt.

Beispiel: Verwendung von **TRAP**:

```
10 REM TRAP
20 TRAP 80
30 FOR I=1 TO 100
40 PRINT EXP(I)
50 NEXT
60 PRINT "GESTOPPT BEI I =";I
70 END
80 PRINT ERR$(ER): RESUME 60

READY.
```

TROFF

Token: $D9

Format: **TROFF**

Zweck: Deaktiviert den Trace-Modus (wird durch **TRON** aktiviert).

Beispiel: Verwendung von **TROFF**:

```
10 REM TROFF
20 TRON              :REM AKTIVIERE TRACE-MODUS
30 FOR I=85 TO 100
40 PRINT I;EXP(I)
50 NEXT
60 TROFF             :REM DEAKTIVIERE TRACE-MODUS

READY.
RUN
[10][20][30][40] 85  8.2230125E36
[50][40] 86  2.2352466E37
[50][40] 87  6.0760302E37
[50][40] 88  1.6516362E38
[50][40] 89
?OVERFLOW ERROR IN 40
READY.
```

TRON

Token: $D8

Format: **TRON**

Zweck: Aktiviert den Trace-Modus (wird durch **TROFF** deaktiviert).

Wenn Sie das BASIC 65-Programm ausführen, werden die jeweiligen Zeilennummern in Klammern angezeigt, bevor eine Aktion für diese Zeile erfolgt.

Beispiel: Verwendung von **TRON**:

```
10 REM TRON
20 TRON              :REM AKTIVIERE TRACE-MODUS
30 FOR I=85 TO 100
40 PRINT I;EXP(I)
50 NEXT
60 TROFF             :REM DEAKTIVIERE TRACE-MODUS

READY.
RUN
[10][20][20][30][40] 85  8.2230125E36
[50][40] 86  2.2352466E37
[50][40] 87  6.0760302E37
[50][40] 88  1.6516362E38
[50][40] 89
?OVERFLOW ERROR IN 40
READY.
```

TYPE

Token: $FE $27

Format: **TYPE** Dateiname [**,D** Laufwerk] [**,U** Gerät]

Zweck: Gibt den Inhalt einer Datei aus, die PETSCII-kodierten Text enthält.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: **TYPE** kann nicht zum Ausgeben von BASIC-Programmen verwendet werden. Verwenden Sie stattdessen für Programmausgaben den **LIST**-Befehl.

TYPE kann nur **SEQ**- oder **USR**-Dateien verarbeiten, die Datensätze mit PETSCII-Text enthalten, die durch das CR-Zeichen begrenzt sind.

Das CR-Zeichen ist auch als Wagenrücklauf bekannt und kann durch Verwendung von **CHR$(13)** erzeugt werden.

Beispiel: Verwendung von **TYPE**:

```
TYPE "LIESMICH"
TYPE "LIESMICH ZUERST",U9
```

UNLOCK

Token: $FE $4F

Format: **UNLOCK** Dateiname/Muster [**,D** Laufwerk] [**,U** Gerät]

Zweck: **UNLOCK** dient zum Entsperren von Dateien. Die angegebene Datei oder eine Reihe von Dateien, die dem Muster entsprechen, wird/werden entsperrt und ist/sind nicht mehr geschützt. Die Datei kann/die Dateien können anschließend mit den Befehlen **DELETE**, **ERASE** oder **SCRATCH** gelöscht werden.

Der Befehl **LOCK** aktiviert die Sperre.

Dateiname ist entweder ein String in Anführungszeichen, z.B. **"Daten"** oder ein Stringausdruck in Klammern gesetzt, z.B. **(FI$)**.

Laufwerk (drive) ist die Laufwerksnummer bei Diskettenstationen mit zwei Laufwerken. Die standardmäßige Laufwerksnummer ist **0** und kann bei Diskettenstationen mit nur einem Laufwerk, wie der 1541, 1571 oder 1581, weggelassen werden.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Das Entsperren einer Datei, die bereits entsperrt ist, hat keine Auswirkungen.

Im Direktmodus wird die Anzahl der entsperrten Dateien ausgegeben. Die vorletzte Zahl der Meldung enthält die Anzahl der entsperrten Dateien.

Beispiel: Verwendung von **UNLOCK**:

```
UNLOCK "SNOOPY",U9 : REM ENTSPERRE DIE DATEI "SNOOPY" AUF GERAET 9
04,FILES UNLOCKED,01,00
READY.

UNLOCK "BS*"       : REM ENTSPERRE ALLE DATEIEN, DIE MIT "BS" BEGINNEN
04,FILES UNLOCKED,03,00
READY.
```

UNTIL

Token: $FC

Notiz: **UNTIL** ist ein Schlüsselwort, das nur in Kombination mit **DO** verwendet wird.

Näheres siehe bei **DO** auf Seite 76.

USING

Token: $FB

Format: **PRINT**[**#** Kanal**,**] **USING** Format**;** Argument

Zweck: Gibt unter Berücksichtigung von **Format** das **Argument** aus.

Das Argument kann entweder ein String oder ein numerischer Wert sein. Das Format der Ausgabe richtet sich nach dem String **Format**.

Kanal ist die Kanalnummer, die bei einem vorherigen Aufruf von Befehlen wie **APPEND**, **DOPEN** oder **OPEN** verwendet worden ist. Wenn kein Kanal angegeben wird, erfolgt die Ausgabe auf dem Bildschirm.

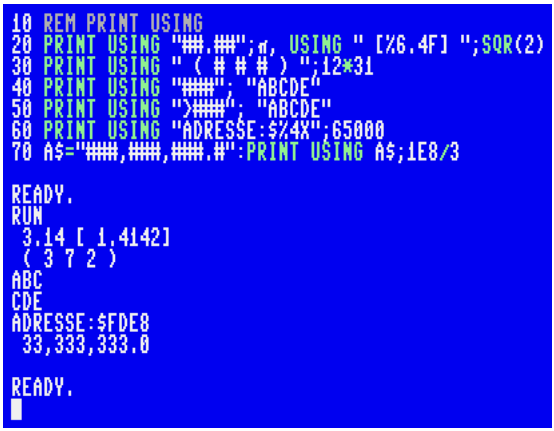
Format ist eine String-Variable oder String-Konstante, die die Regeln für die Formatierung definiert. Bei Verwendung einer Zahl als **Argument** kann die Formatierung entweder im CBM-Stil mit einem Muster wie **###.##** oder im C-Stil mit einem <Breite.Genauigkeit>-Spezifikator wie **%3D %7.2F %4X** erfolgen.

Argument ist die zu formatierende Zahl. Wenn das Argument nicht in das Format passt, zum Beispiel wenn versucht wird, eine vierstellige Variable in eine Reihe von drei Rautezeichen auszugeben (**###**), werden stattdessen Sterne (*****) verwendet.

Notiz: Der Formatstring wird nur für ein Argument verwendet. Es ist allerdings möglich, weitere **USING Format;Argument**-Sequenzen anzuhängen.

Argument kann aus darstellbare Zeichen und Steuercodes bestehen. Darstellbare Zeichen werden an der Cursorposition ausgegeben, während Steuercodes ausgeführt werden. Die Anzahl der **#**-Zeichen bestimmt die Breite der Ausgabe. Ist das erste Zeichen des Formatstrings ein Gleichheitszeichen **=**, wird der Argumentstring zentriert. Ist das erste Zeichen des Formatstrings ein **>**-Zeichen, wird der Argumentstring rechtsbündig ausgerichtet.

Beispiel: Verwendung von **USING**:



USR

Token: $B7

Format: **USR**(numerischer Ausdruck)

Zweck: Ruft eine Assembler-Routine auf, deren Speicheradresse bei $02F8 - $02F9 gespeichert ist. Das Ergebnis des **numerischen Ausdrucks** wird in den Fließkomma-Akkumulator 1 geschrieben.

Nach der Ausführung der Assembler-Routine gibt BASIC den Inhalt des Fließkomma-Akkumulators 1 zurück.

Notiz: Die Bänke 0-127 ermöglichen den Zugriff auf RAM- oder ROM-Bänke. Bänke über 127 werden für den Zugriff auf Ein-/Ausgabe und die zugrunde liegende Systemhardware wie VIC (Grafik), SID (Ton), FDC (Diskettencontroller) usw. verwendet.

Wenn Sie mehr darüber erfahren möchten, finden Sie viele weitere Informationen und Beispiele im (englischsprachigen) "MEGA65 Developer's Guide".

Beispiel: Verwendung von **USR**:

```
10 REM USR
20 UX = DEC("7F00")             : REM ADRESSE DER NUTZERROUTINE
30 BANK 128                     : REM WAEHLE SYSTEM-BANK AUS
40 BLOAD "ML-PROG",P(UX)        : REM LADE NUTZERROUTINE
50 POKE (DEC("2F8")),UX AND 255 : REM ZIELADRESSE DER NUTZERROUTINE (LOW)
60 POKE (DEC("2F9")),UX / 256   : REM ZIELADRESSE DER NUTZERROUTINE (HIGH)
70 PRINT USR(π)                 : REM GIB ERGEBNIS FUER ARGUMENT PI AUS

READY.
```

VAL

Token: $C5

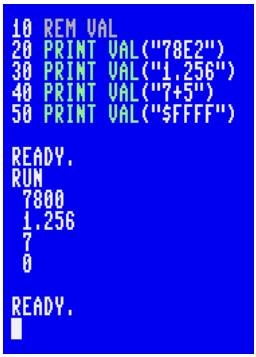
Format: **VAL**(String-Ausdruck)

Zweck: Konvertiert einen String in einen Fließkommawert.

Diese Funktion verhält sich genauso wie das Lesen aus einem String.

Notiz: Ein String, der eine ungültige Zahl enthält, führt nicht zu einem Fehler, sondern liefert stattdessen 0 als Ergebnis.

Beispiel: Verwendung von **VAL**:



VERIFY

Token: $95

Format: **VERIFY** Dateiname [**,** Gerät [**,** Binärflag]]

Zweck: **VERIFY** ohne **Binärflag** vergleicht ein BASIC-Programm im Speicher mit einer Datei auf Diskette des Typs **PRG**. Es macht das Gleiche wie **DVERIFY**, allerdings unterscheidet sich die Syntax.

VERIFY mit **Binärflag** vergleicht einen Speicherbereich mit einer Datei auf Diskette des Typs **PRG**. Es macht das Gleiche wie **BVERIFY**, allerdings unterscheidet sich die Syntax.

Dateiname is entweder ein String in Anführungszeichen, zum Beispiel **"PROGRAMM"**, oder ein String-Ausdruck.

Gerät (unit) ist die Gerätenummer auf dem IEC-Bus. Normalerweise im Bereich von 8 bis 11 für die Diskettenlaufwerke. Wenn eine Variable verwendet wird, muss sie in Klammern gesetzt werden. Standardmäßig ist das Gerät auf **8** eingestellt.

Notiz: Der Befehl **VERIFY** kann nur auf Gleichheit prüfen. Er gibt keine Informationen über die Anzahl oder Position der unterschiedlich bewerteten Bytes. **VERIFY** beendet sich entweder mit der Meldung **OK** oder mit **VERIFY ERROR**.

VERIFY ist in BASIC 65 veraltet. Es ist nur noch aus Gründen der Abwärtskompatibilität vorhanden. Es wird empfohlen, stattdessen **DVERIFY** und **BVERIFY** zu verwenden.

Beispiel: Verwendung von **VERIFY**:

```
VERIFY "ABENTEUER"
VERIFY "ZORK-I",9
VERIFY "1:LABYRINTH",10
```

VIEWPORT

Token: $FE $31

Format: **VIEWPORT CLR**
VIEWPORT DEF x, y, Breite, Höhe

Zweck: **VIEWPORT DEF** definiert einen sogenannten Clipping-Bereich (dt. etwa "Ausschnittsbereich") mit dem Ursprung (obere linke Position) auf **x, y** und der **Breite** und **Höhe**. Alle folgenden Grafikbefehle sind auf den Bereich **VIEWPORT** beschränkt.

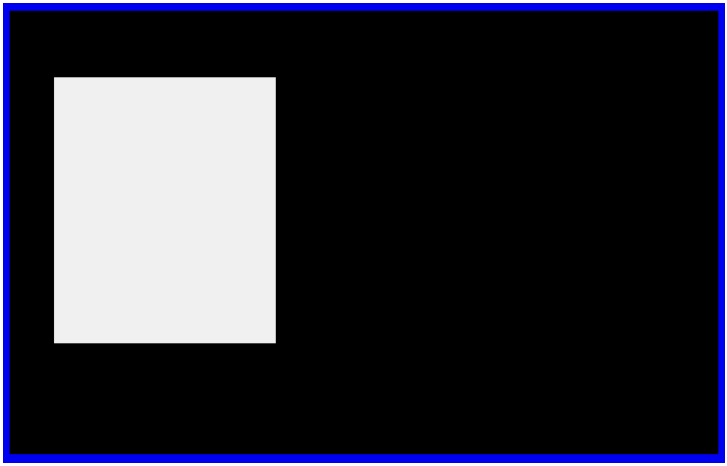
VIEWPORT CLR füllt den Clipping-Bereich mit der Farbe des Zeichenstifts.

Notiz: Der Clipping-Bereich kann mit folgendem Befehl auf Vollbild zurückgesetzt werden: **VIEWPORT DEF 0, 0, Breite, Höhe** unter Verwendung der gleichen Werte für **Breite** und **Höhe** wie im **SCREEN**-Befehl.

Beispiel: Verwendung von **VIEWPORT**:

```
10 REM VIEWPORT
20 SCREEN 320,200,2
30 VIEWPORT DEF 20,30,100,120 : REM BEREICH 20->119, 30->149
40 PEN 1                      : REM WAEHLER FARBE 1
50 VIEWPORT CLR               : REM FUELLE BEREICH MIT STIFTFARBE
60 GETKEY A$                  : REM WARTE AUF TASTENDRUCK
70 SCREEN CLOSE

READY.
```



VOL

Token: $DB

Format: **VOL** Lautstärke

Zweck: Stellt die Lautstärke für die Tonausgabe mit **SOUND** oder **PLAY** ein.

Lautstärke 0 (aus) bis 15 (maximale Laustärke)

Notiz: Diese Lautstärkeeinstellung wirkt sich auf alle Stimmen aus.

Beispiel: Verwendung von **VOL**:

```
10 REM VOL
20 TEMPO 22
30 FOR V = 2 TO 8 STEP 2
40 VOL V
50 PLAY "T0M3O4QGAGFED","T2O4M5P0H.DP5GB","T5O3IGAGAGAABABAB"
60 IF RPLAY(1) THEN GOTO 60
70 NEXT V
80 PLAY "T005QCO4GEH.C","T2O5IEFEDEDCEGO6P9CP0R","T5O3ICDCDEFEDCO4C"

READY.
```

WAIT

Token: $92

Format: **WAIT** Adresse**,** AND-Maske [**,** XOR-Maske]

Zweck: Hält das BASIC-Programm an, bis ein angefordertes Bitmuster von der angegebenen Adresse gelesen wird.

Adresse ist die Adresse in der aktuellen Speicherbank, die gelesen wird.

AND-Maske ist die anzuwendende **AND**-Maske.

XOR-Maske ist die anzuwendende **XOR**-Maske.

WAIT liest den Byte-Wert von **Adresse** und wendet die Masken an: **Ergebnis = PEEK(Adresse) AND AND-Maske XOR XOR-Maske**.

Die Pause endet, wenn das Ergebnis ungleich Null ist, andernfalls wird das Lesen wiederholt. Dies kann den Computer auf unbestimmte Zeit aufhalten, wenn die Bedingung nie erfüllt wird.

Notiz: **WAIT** wird üblicherweise verwendet, um Hardware-Register oder Systemvariablen zu untersuchen und auf ein Ereignis zu warten, zum Beispiel ein Joystick-Ereignis, ein Maus-Ereignis, einen Tastendruck oder eine bestimmte Rasterlinie, die auf dem Bildschirm gezeichnet werden soll.

Beispiel: Verwendung von **WAIT**:

```
10 REM WAIT
20 BANK 128
30 WAIT 211,1 : REM WARTE BIS DIE SHIFT-TASTE GEDRUECKT WIRD

READY.
```

WHILE

Token: $ED

Notiz: **WHILE** ist ein Schlüsselwort, das nur in Kombination mit **DO** verwendet wird.

Näheres siehe bei **DO** auf Seite 76.

WINDOW

Token: $FE $1A

Format: **WINDOW** Links**,** Oben**,** Rechts**,** Unten [**,** Lösche]

Zweck: Legt das Textbildschirmfenster fest.

Links linke Spalte

Oben obere Zeile

Rechts rechte Spalte

Unten untere Zeile

Lösche Flag zum Textbildschirm löschen

Notiz: Die Zeilenwerte reichen von 0 bis 24. Die Spaltenwerte reichen von 0 bis entweder 39 oder 79. Dies hängt vom Bildschirmmodus ab.

Es kann nur ein Fenster auf dem Bildschirm angezeigt werden. Durch zweimaliges Drücken der CLR HOME-Taste oder durch die Anweisung **PRINT CHR$(19)CHR$(19)** wird das Fenster auf die Standardeinstellung (Vollbild) zurückgesetzt.

Beispiel: Verwendung von **WINDOW**:

```
10 REM WINDOW
20 WINDOW 0,1,79,24   : REM BILDSCHIRM OHNE DIE OBERSTE ZEILE
30 WINDOW 0,0,79,24,1 : REM LOESCHE VOLLBILD-TEXTBILDSCHIRM
40 WINDOW 0,12,79,24  : REM UNTERE HAELFTE DES BILDSCHIRMS
50 WINDOW 20,5,59,15  : REM KLEINES ZENTRIERTES FENSTER

READY.
```

XOR

Token: $E9

Format: Operand **XOR** Operand

Zweck: Der boolesche Operator **XOR** führt eine bitweise logische Exklusiv-ODER-Verknüpfung an zwei 16-Bit-Werten durch. Integer-Operanden werden so verwendet, wie sie sind. Reelle Operanden werden in eine vorzeichenbehaftete 16 Bit-Ganzzahl umgewandelt (mit Verlust der Genauigkeit). Logische Operanden werden in 16 Bit-Ganzzahlen umgewandelt mit $FFFF (dezimal -1) für TRUE ("wahr") und $0000 (dezimal 0) für FALSE ("falsch").

Ausdruck	Ergebnis
0 XOR 0	0
0 XOR 1	1
1 XOR 0	1
1 XOR 1	0

Notiz: Das Ergebnis ist vom Typ Integer (ganze Zahl). Wenn das Ergebnis in einem logischen Kontext verwendet wird, wird der Wert 0 als FALSE betrachtet, und alle anderen Werte ungleich Null werden als TRUE betrachtet.

Beispiel: Verwendung von **XOR**:

```
10 REM XOR
20 FOR I = 0 TO 8: PRINT I XOR 5;: NEXT I

READY.
RUN
 5  4  7  6  1  0  3  2  13
READY.
```

Schlüsselwörter und Token - Teil 1

*	AC	COLOR	E7	FAST	FE25
+	AA	CONCAT	FE13	FGOSUB	FE48
-	AB	CONT	9A	FGOTO	FE47
/	AD	COPY	F4	FILTER	FE03
<	B3	COS	BE	FIND	FE2B
=	B2	CURSOR	FE41	FN	A5
>	B1	CUT	E4	FONT	FE46
ABS	B6	DATA	83	FOR	81
AND	AF	DCLEAR	FE15	FOREGROUND	FE39
APPEND	FE0E	DCLOSE	FE0F	FORMAT	FE37
ASC	C6	DEC	D1	FRE	B8
ATN	C1	DEF	96	FREAD#	FE1C
AUTO	DC	DELETE	F7	FWRITE#	FE1E
BACKGROUND	FE3B	DIM	86	GCOPY	FE32
BACKUP	F6	DIR	EE		
BANK	FE02	DISK	FE40	GET	A1
BEGIN	FE18	DLOAD	F0	GO	CB
BEND	FE19	DMA	FE1F	GOSUB	8D
BLOAD	FE11	DMODE	FE35	GOTO	89
BOOT	FE1B	DO	EB	GRAPHIC	DE
BORDER	FE3C	DOPEN	FE0D	HEADER	F1
BOX	E1	DPAT	FE36	HELP	EA
BSAVE	FE10	DSAVE	EF	HEX$	D2
BUMP	CE03	DVERIFY	FE14	HIGHLIGHT	FE3D
BVERIFY	FE28	ECTORY	FE29	IF	8B
CATALOG	FE0C	EDIT	FE45	INPUT	85
CHANGE	FE2C	EDMA	FE21	INPUT#	84
CHAR	E0	ELLIPSE	FE30	INSTR	D4
CHR$	C7	ELSE	D5	INT	B5
CIRCLE	E2	END	80	JOY	CF
CLOSE	A0	ENVELOPE	FE0A	KEY	F9
CLR	9C	ERASE	FE2A	LEFT$	C8
CMD	9D	ERR$	D3	LEN	C3
COLLECT	F3	EXIT	ED	LET	88
COLLISION	FE17	EXP	BD	LINE	E5

Schlüsselwörter und Token – Teil 2

LIST	9B	PRINT#	98	SLEEP	FE0B
LOAD	93			SOUND	DA
LOADIFF	FE43	RCOLOR	CD	SPC(	A6
LOG	BC	RCURSOR	FE42	SPEED	FE26
LOG10	CE08	READ	87	SPRCOLOR	FE08
LOOP	EC	RECORD	FE12	SPRDEF	FE1D
LPEN	CE04	REM	8F	SPRITE	FE07
MEM	FE23	RENAME	F5	SPRSAV	FE16
MERGE	E6	RENUMBER	F8	SQR	BA
MID$	CA	RESTORE	8C	STEP	A9
MOD	CE0B	RESUME	D6	STOP	90
MONITOR	FA	RETURN	8E	STR$	C4
MOUSE	FE3E	RGRAPHIC	CC	SYS	9E
MOVSPR	FE06	RIGHT$	C9	TAB(	A3
NEW	A2	RMOUSE	FE3F	TAN	C0
NEXT	82	RND	BB	TEMPO	FE05
NOT	A8	RPALETTE	CE0D	THEN	A7
OFF	FE24	RPEN	D0	TO	A4
ON	91	RPLAY	CE0F	TRAP	D7
OPEN	9F	RREG	FE09	TROFF	D9
OR	B0	RSPCOLOR	CE07	TRON	D8
PAINT	DF	RSPEED	CE0E	TYPE	FE27
PALETTE	FE34	RSPPOS	CE05	UNTIL	FC
PASTE	E3	RSPRITE	CE06	USING	FB
PEEK	C2	RUN	8A	USR	B7
PEN	FE33	RWINDOW	CE09	VAL	C5
PIXEL	CE0C	SAVE	94	VERIFY	95
PLAY	FE04	SAVEIFF	FE44	VIEWPORT	FE31
POINTER	CE0A	SCNCLR	E8	VOL	DB
POKE	97	SCRATCH	F2	WAIT	92
POLYGON	FE2F	SCREEN	FE2E	WHILE	FD
POS	B9	SET	FE2D	WINDOW	FE1A
POT	CE02	SGN	B4	XOR	E9
PRINT	99	SIN	BF	^	AE

Token und Schlüsselwörter - Teil 1

80 END	A3 TAB(	C6 ASC
81 FOR	A4 TO	C7 CHR$
82 NEXT	A5 FN	C8 LEFT$
83 DATA	A6 SPC(	C9 RIGHT$
84 INPUT#	A7 THEN	CA MID$
85 INPUT	A8 NOT	CB GO
86 DIM	A9 STEP	CC RGRAPHIC
87 READ	AA +	CD RCOLOR
88 LET	AB -	CF JOY
89 GOTO	AC *	D0 RPEN
8A RUN	AD /	D1 DEC
8B IF	AE ^	D2 HEX$
8C RESTORE	AF AND	D3 ERR$
8D GOSUB	B0 OR	D4 INSTR
8E RETURN	B1 >	D5 ELSE
8F REM	B2 =	D6 RESUME
90 STOP	B3 <	D7 TRAP
91 ON	B4 SGN	D8 TRON
92 WAIT	B5 INT	D9 TROFF
93 LOAD	B6 ABS	DA SOUND
94 SAVE	B7 USR	DB VOL
95 VERIFY	B8 FRE	DC AUTO
96 DEF	B9 POS	DD IMPORT
97 POKE	BA SQR	DE GRAPHIC
98 PRINT#	BB RND	DF PAINT
99 PRINT	BC LOG	E0 CHAR
9A CONT	BD EXP	E1 BOX
9B LIST	BE COS	E2 CIRCLE
9C CLR	BF SIN	E3 PASTE
9D CMD	C0 TAN	E4 CUT
9E SYS	C1 ATN	E5 LINE
9F OPEN	C2 PEEK	E6 MERGE
A0 CLOSE	C3 LEN	E7 COLOR
A1 GET	C4 STR$	E8 SCNCLR
A2 NEW	C5 VAL	E9 XOR

Token und Schlüsselwörter - Teil 2

EA HELP	FE02 BANK	FE26 SPEED
EB DO	FE03 FILTER	FE27 TYPE
EC LOOP	FE04 PLAY	FE28 BVERIFY
ED EXIT	FE05 TEMPO	FE29 ECTORY
EE DIR	FE06 MOVSPR	FE2A ERASE
EF DSAVE	FE07 SPRITE	FE2B FIND
F0 DLOAD	FE08 SPRCOLOR	FE2C CHANGE
F1 HEADER	FE09 RREG	FE2D SET
F2 SCRATCH	FE0A ENVELOPE	FE2E SCREEN
F3 COLLECT	FE0B SLEEP	FE2F POLYGON
F4 COPY	FE0C CATALOG	FE30 ELLIPSE
F5 RENAME	FE0D DOPEN	FE31 VIEWPORT
F6 BACKUP	FE0E APPEND	FE32 GCOPY
F7 DELETE	FE0F DCLOSE	FE33 PEN
F8 RENUMBER	FE10 BSAVE	FE34 PALETTE
F9 KEY	FE11 BLOAD	FE35 DMODE
FA MONITOR	FE12 RECORD	FE36 DPAT
FB USING	FE13 CONCAT	FE37 FORMAT
FC UNTIL	FE14 DVERIFY	FE39 FOREGROUND
FD WHILE	FE15 DCLEAR	FE3B BACKGROUND
CE02 POT	FE16 SPRSAV	FE3C BORDER
CE03 BUMP	FE17 COLLISION	FE3D HIGHLIGHT
CE04 LPEN	FE18 BEGIN	FE3E MOUSE
CE05 RSPPOS	FE19 BEND	FE3F RMOUSE
CE06 RSPRITE	FE1A WINDOW	FE40 DISK
CE07 RSPCOLOR	FE1B BOOT	FE41 CURSOR
CE08 LOG10	FE1C FREAD#	FE42 RCURSOR
CE09 RWINDOW	FE1D SPRDEF	FE43 LOADIFF
CE0A POINTER	FE1E FWRITE#	FE44 SAVEIFF
CE0B MOD	FE1F DMA	FE45 EDIT
CE0C PIXEL	FE21 EDMA	FE46 FONT
CE0D RPALETTE	FE23 MEM	FE47 FGOTO
CE0E RSPEED	FE24 OFF	FE48 FGOSUB
CE0F RPLAY	FE25 FAST	FE49 MOUNT

BASIC 65-Fehlermeldungen

BAD DISK (36) — **"schlechte Diskette"**
Es wurde versucht, eine noch nicht formatierte Diskette mit dem verkürzten FORMAT- bzw. HEADER-Befehl zu löschen, eine schadhafte Diskette zu formatieren oder ein BOOT SYS-Befehl ist fehlgeschlagen, weil die Diskette nicht gelesen werden konnte.

BAD SUBSCRIPT (18) — **"falscher Array-Index"**
Das Programm hat versucht, auf ein Element eines Arrays zu verweisen, das außerhalb des durch eine DIM-Anweisung, eine fehlende DIM-Anweisung oder einen falsch eingegebenen Funktionsnamen angegebenen Bereichs liegt.

BEND NOT FOUND (37) — **"BEND nicht gefunden"**
Für eine BEGIN-Anweisung wurde keine dazugehörige BEND-Anweisung gefunden.

BREAK (30) — **"Unterbrechung"**
Das Programm wurde entweder der STOP-Taste oder durch eine STOP-Anweisung angehalten.

CAN'T CONTINUE (26) — **"keine Fortsetzung möglich"**
Der Befehl CONT funktioniert nicht, wenn das Programm nicht ausgeführt wurde, ein Fehler aufgetreten ist oder nach der Unterbrechung eine Zeile bearbeitet wurde.

CAN'T RESUME (31) — **"RESUME ohne TRAP"**
Es wurde eine RESUME-Anweisung gefunden, ohne dass eine TRAP-Anweisung aktiv ist, oder es ist ein Fehler im Trap-Handler selbst aufgetreten.

DEVICE NOT PRESENT (5) — **"Gerät nicht verfügbar"**
Das gewünschte Ein-/Ausgabe-Gerät ist nicht verfügbar.

DIRECT MODE ONLY (34) — **"nur Direktmodus erlaubt"**
Eine Anweisung, die nur im Direktmodus verwendet werden kann, wurde in einem Programm aufgerufen.

DIVISION BY ZERO (20)	**"Division durch Null"** Eine Division durch Null ist unzulässig.
EDIT MODE (42)	**"Textmodus-Fehler"** Eine Anweisung, die nur im Programmiermodus (Standardmodus oder EDIT OFF) funktioniert, wurde im Textmodus (EDIT ON) ausgeführt.
FILE DATA (24)	**"ungültige Dateidaten"** Es wurde der falsche Datentyp aus einer Datei gelesen.
FILE NOT FOUND (4)	**"Datei nicht gefunden"** Auf dem angegebenen Laufwerk existiert keine Datei mit diesem Namen.
FILE NOT OPEN (3)	**"Datei nicht geöffnet"** Die in einer E/A-Anweisung angegebene Dateinummer muss vor der Verwendung geöffnet werden.
FILE OPEN (2)	**"Datei ist bereits geöffnet"** Es wurde versucht, eine Datei mit der Nummer einer bereits geöffneten Datei zu öffnen.
FILE READ (41)	**"Lesefehler aus Datei"** Es gab ein Problem beim Lesen von Daten aus einer Datei. Ähnlich wie bei LOAD ERROR.
FORMULA TOO COMPLEX (25)	**"zu komplexer Ausdruck"** Ein Ausdruck ist für BASIC 65 zu kompliziert, um ihn auf einmal zu verarbeiten. Zerlegen Sie ihn in kleinere Teile oder verwenden Sie weniger Klammern.
ILLEGAL DEVICE NUMBER (9)	**"unerlaubte Geräteadresse"** Es wird entweder versucht eine Ein-/Ausgabe-Operation mit einem unerlaubten Gerät oder Einheit auszuführen (z.B. SAVE zum Bildschirm) oder es wird eine Geräteadresse oberhalb von 15 verwendet.
ILLEGAL DIRECT (21)	**"unerlaubter Eingabemodus"** Der Befehl darf nicht im Direktmodus, sondern nur in einem Programm verwendet werden.

ILLEGAL QUANTITY (14)	**"unerlaubter Wert"** Ein numerisches Argument einer Funktion oder ein numerischer Parameter für eine Anweisung liegt außerhalb des zulässigen Bereiches.
LINE NUMBER TOO LARGE (38)	**"Zeilennummer zu groß"** Eine Zeilennummer darf in BASIC 65 nicht größer als 65535 sein.
LOAD (29)	**"Ladefehler"** Beim Laden eines Programms von Diskette ist ein Lesefehler aufgetreten.
LOOP NOT FOUND (32)	**"DO ohne LOOP"** Das Programm ist auf eine DO-Anweisung gestoßen und kann die entsprechende LOOP-Schleife nicht finden.
LOOP WITHOUT DO (33)	**"LOOP ohne DO"** Es wurde eine LOOP-Schleife ohne eine aktive DO-Anweisung gefunden.
MISSING FILE NAME (8)	**"Dateiname fehlt"** Die in einer E/A-Anweisung angegebene Dateinummer muss vor der Verwendung geöffnet werden.
NEXT WITHOUT FOR (10)	**"NEXT ohne FOR"** Entweder sind Schleifen falsch verschachtelt oder es gibt einen Variablennamen in einer NEXT-Anweisung, der nicht mit einem in FOR übereinstimmt.
NOT INPUT FILE (6)	**"keine Eingabedatei"** Es wurde versucht, Daten aus einer Datei zu lesen, die zum Schreiben geöffnet wurde.
NOT OUTPUT FILE (7)	**"keine Ausgabedatei"** Es wurde versucht, Daten in eine Datei zu schreiben, die zum Lesen geöffnet wurde.
OUT OF DATA (13)	**"nicht genug Daten"** Es wird versucht, mit einer READ-Anweisung mehr Daten zu lesen, als in DATA-Zeilen deklariert sind.

OUT OF MEMORY **"Speicherüberlauf"**
(16)
Entweder reicht der Arbeitsspeicher für das Programm oder die Variablen nicht aus, oder der Stapelspeicher ist wegen zuvieler aktiver DO-, FOR- oder GOSUB-Anweisungen übergelaufen.

OVERFLOW **"Überlauf"**
(15)
Das Ergebnis einer Berechnung ist größer als die größte zulässige Zahl (1.701411834E+38).

REDIMENSIONED ARRAY **"mehrfache Felddimensionierung"**
(19)
Ein Array darf in BASIC 65 nur ein Mal dimensioniert werden.

RESERVED NAME **"reservierte Systemvariable"**
(43)
Es wurde versucht, eine reservierte Systemvariable (z.B. TI oder TI$) zu deklarieren oder ihr einen Wert zuzuweisen.

RETURN WITHOUT GOSUB **"RETURN ohne GOSUB"**
(12)
Es wurde eine RETURN-Anweisung gefunden, obwohl keine GOSUB-Anweisung aktiv war.

SCREEN NOT OPEN **"Grafikbildschirm nicht geöffnet"**
(35)
Eine Grafikanweisung wurde verwendet, bevor ein Grafikbildschirm definiert und geöffnet worden ist.

STRING TOO LONG **"zu langer String"**
(23)
Es wurde versucht, einem String mehr als die möglichen 255 Zeichen zuzuweisen oder mehr als 160 Zeichen über die Tastatur einzugeben.

SYNTAX **"Syntaxfehler"**
(11)
Eine Anweisung kann von BASIC 65 nicht erkannt werden. Das kann an fehlenden oder zu vielen Klammern, Parametern, Begrenzungszeichen oder einem falsch geschriebenen Schlüsselwort liegen.

TOO MANY FILES **"zu viele Dateien"**
(1)
Es sind maximal 10 gleichzeitig geöffnete Dateien erlaubt.

TYPE MISMATCH (22) — **"fehlende Variablentyp-Übereinstimmung"**
Eine numerische Variable wurde anstelle einer erforderlichen String-Variable verwendet oder umgekehrt.

UNDEFINED FUNCTION (27) — **"nichtdefinierte Funktion"**
Es wurde versucht, eine benutzerdefinierte Funktion zu verwenden, die nie definiert wurde.

UNDEFINED STATEMENT (17) — **"nichtdefinierte Zeilennummer"**
Eine Zeilennummer, auf die verwiesen wird, existiert nicht.

UNIMPLEMENTED COMMAND (40) — **"nicht implementierter Befehl"**
Der angegebene Befehl ist nicht in BASIC 65 implementiert.

UNRESOLVED REFERENCE (39) — **"offener Verweis"**
Die Neunummerierung der Zeilennummern mit RENUMBER ist fehlgeschlagen, weil eine referenzierte Zeilennummer nicht existiert.

VERIFY (28) — **"Verifizierungsfehler"**
Das mit VERIFY überprüfte Programm auf der Diskette stimmt nicht mit dem Programm im Speicher überein.

BASIC 65-FEHLERCODES:

```
 1 TOO MANY FILES
 2 FILE OPEN
 3 FILE NOT OPEN
 4 FILE NOT FOUND
 5 DEVICE NOT PRESENT
 6 NOT INPUT FILE
 7 NOT OUTPUT FILE
 8 MISSING FILE NAME
 9 ILLEGAL DEVICE NUMBER
10 NEXT WITHOUT FOR
11 SYNTAX
12 RETURN WITHOUT GOSUB
13 OUT OF DATA
14 ILLEGAL QUANTITY
15 OVERFLOW
16 OUT OF MEMORY
17 UNDEFINED STATEMENT
18 BAD SUBSCRIPT
19 REDIMENSIONED ARRAY
20 DIVISION BY ZERO
21 ILLEGAL DIRECT
22 TYPE MISMATCH
23 STRING TOO LONG
24 FILE DATA
25 FORMULA TOO COMPLEX
26 CAN'T CONTINUE
27 UNDEFINED FUNCTION
28 VERIFY
29 LOAD
30 BREAK
31 CAN'T RESUME
32 LOOP NOT FOUND
33 LOOP WITHOUT DO
34 DIRECT MODE ONLY
35 SCREEN NOT OPEN
36 BAD DISK
37 BEND NOT FOUND
38 LINE NUMBER TOO LARGE
39 UNRESOLVED REFERENCE
40 UNIMPLEMENTED COMMAND
41 FILE READ
42 EDIT MODE
43 RESERVED NAME
```

KAPITEL 3

Spezielle Tastaturbefehle und -Sequenzen

- PETSCII-Codes und CHR$
- Kontrollcodes
- SHIFT-Codes
- Escape-Sequenzen

PETSCII-CODES UND CHR$

In BASIC 65 kann **PRINT CHR$(X)** verwendet werden, um das einem PETSCII-Code zugeordnete Zeichen auszugeben. Unten finden Sie eine vollständige, nach PETSCII-Code sortierte Zeichentabelle. Zum Beispiel, wenn Sie Index 65 aus der untenstehenden Tabelle verwenden, **PRINT CHR$(65)**, dann werden Sie den Buchstaben **A** drucken. Weitere Informationen zu **CHR$** finden Sie auf Seite 44.

Mit dem ASC-Befehl können Sie auch den umgekehrten Weg gehen. Zum Beispiel: **PRINT ASC("A")** gibt **65** aus, was mit dem Code in der Tabelle übereinstimmt.

HINWEIS: Die Funktionstastenwerte (F1-F14 + HELP) in dieser Tabelle sind nicht dazu gedacht über CHR$() gedruckt zu werden, sondern um die Eingabe von Funktionstasten in BASIC-Programmen über die Befehle GET / GETKEY zu auszuwerten.

0

1

2 UNTERSTREICHEN AN

3

4

5 WEISS

6

7 GLOCKE

8

9 TAB

10 ZEILENVORSCHUB LF

11 DEAKTIVIERE SHIFT + M

12 AKTIVIERE SHIFT + M

13 RETURN

14 KLEINSCHRIFT

15 BLINKEN AN

16 F9

17 ↓

18 RVS ON

19 CLR HOME

20 INST DEL

21 F10 / LETZTES WORT

22 F11

23 F12 / NÄCHSTES WORT

24 SETZE/LÖSCHE TAB

25 F13

26 F14 / BACK TAB

27 ESCAPE

28 ROT

29 →

30 GRÜN

31 BLAU

32 SPACE

33 !

34 "

35 #

36 $

37 %

38 &

39 '

40 (

41)

42 *

43 +

44 ,

45 -

46 .

47 /

48 0

49 1

50 2

51 3

52 4

53 5

54 6

55 7

56 8

57 9

58 :

59 ;

60 <

61 =

62 >

63 ?

64 @

65 A

66 B

67 C

68 D

69 E

70 F

71 G

72 H

73 I

74 J

75 K

76 L
77 M
78 N
79 O
80 P
81 Q
82 R
83 S
84 T
85 U
86 V
87 W
88 X
89 Y
90 Z
91 [
92 £
93]
94 ↑
95 ←
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126 π
127
128
129 ORANGE
130 UNTERSTREICHEN AUS
131
132 HELP (HILFE)
133 F1
134 F3
135 F5
136 F7
137 F2
138 F4
139 F6
140 F8
141 SHIFT RETURN
142 GROSSSCHRIFT
143 BLINKEN AUS
144 SCHWARZ
145 ↑
146 RVS OFF
147 SHIFT CLR HOME
148 SHIFT INST DEL
149 BRAUN
150 HELLROT
151 DUNKELGRAU
152 GRAU
153 HELLGRÜN
154 HELLBLAU
155 HELLGRAU
156 VIOLETT
157 ←
158 GELB
159 CYAN
160 SPACE
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191

HINWEIS: Die Codes von 192 bis 223 entsprechen den Werten 96 bis 127. Die Codes von 224 bis 254 entsprechen 160 bis 190 und Code 255 ist gleich 126.

KONTROLLCODES

Tastatursteuerung	Funktion
Farben	
CTRL + 1 bis 8	Wählt die Textfarbe aus der ersten Farbpalette. Weitere Informationen zu den verfügbaren Farben finden Sie unter dem BASIC-Befehl **BACKGROUND** auf Seite 19.
C= + 1 bis 8	Wählt die Textfarbe aus der zweiten Farbpalette.
CTRL + R	Invertiert die Textdarstellung.
CTRL + E	Setzt die Farbe des Cursors auf die Standardfarbe (weiß) zurück.
Tabulatoren	
CTRL + Z	Verschiebt den Cursor nach links auf die nächste Tabulatorposition. Wenn keine Tabulatorpositionen mehr vorhanden sind, bleibt der Cursor am Anfang der Zeile stehen.
CTRL + I	Verschiebt den Cursor nach rechts auf die nächste Tabulatorposition. Wenn keine Tabulatorpositionen mehr vorhanden sind, bleibt der Cursor am Ende der Zeile stehen.
CTRL + X	Setzt oder löscht die aktuelle Bildschirmspalte als Tabulatorposition. Verwenden Sie CTRL + Z und I, um zu allen mit X eingestellten Positionen hin und her zu springen.

Tastatursteuerung	Funktion
Cursorbewegung	
CTRL + Q	Bewegt den Cursor jeweils um eine Zeile nach unten. Entspricht der Taste ↓.
CTRL + J	Bewegt den Cursor um eine Position nach unten. Wenn Sie sich in einer langen BASIC-Codezeile befinden, die sich auf zwei Zeilen ausgedehnt hat, bewegt sich der Cursor zwei Zeilen nach unten, um in die nächste Zeile zu gelangen.
CTRL +]	Entspricht der Taste →.
CTRL + T	Verschiebt das Zeichen unmittelbar nach links und verschiebt alle Zeichen rechts davon um eine Position nach links. Dies entspricht der Taste INST DEL.
CTRL + M	Führt einen Zeilenumbruch aus, der der Taste RETURN entspricht.
Wortbewegung	
CTRL + U	Bewegt den Cursor zurück an den Anfang des vorherigen Wortes. Wenn keine Wörter zwischen der aktuellen Cursorposition und dem Anfang der Zeile liegen, bewegt sich der Cursor zur ersten Spalte der aktuellen Zeile.
CTRL + W	Setzt den Cursor an den Anfang des nächsten Wortes. Wenn sich zwischen dem Cursor und dem Ende der Zeile keine Wörter befinden, springt der Cursor in die erste Spalte der nächsten Zeile.

Tastatursteuerung	Funktion
Bildlauf (Scrollen)	
CTRL + P	BASIC-Listing um eine Zeile nach unten scrollen. Entspricht der Taste F9.
CTRL + V	BASIC-Listing um eine Zeile nach oben scrollen. Entspricht der Taste F11.
CTRL + S	Unterbindet das Scrollen. Entspricht der Taste NO SCROLL.
Formatierung	
CTRL + B	Aktiviert den Unterstreichungsmodus für Text. Sie können den Unterstreichungsmodus deaktivieren, indem Sie die ESC und dann O drücken.
CTRL + O	Aktiviert den blinkenden Textmodus. Sie können den Blinkmodus deaktivieren, indem Sie die ESC und dann O drücken.
Groß- und Kleinschreibung	
CTRL + N	Ändert den Textmodus von Groß- auf Kleinschreibung.
CTRL + K	Sperrt die Umschaltung zwischen Groß- und Kleinschrift, die normalerweise mit den Tasten und SHIFT vorgenommen wird.
CTRL + L	Erlaubt die Umschaltung zwischen Groß- und Kleinschrift, die normalerweise mit den Tasten und SHIFT vorgenommen wird.

Tastatursteuerung	Funktion
Sonstiges	
CTRL + G	Erzeugt einen Glockenton.
CTRL + [	Entspricht dem Drücken der ESC-Taste.
CTRL + *	Ruft den Matrix-Modus-Debugger auf.

SHIFT-CODES

Tastatursteuerung	Funktion
SHIFT + INST DEL	Fügt ein Zeichen an der aktuellen Cursorposition ein und verschiebt alle Zeichen um eine Position nach rechts.
SHIFT + HOME	Löscht den gesamten Bildschirm und bewegt den Cursor in die obere linke Ecke.

ESCAPE-SEQUENZEN

Um eine Escape-Sequenz auszuführen, drücken Sie die Taste ESC, lassen sie wieder los und drücken anschließend eine der folgenden Tasten:

Taste	Aktion
Editor-Verhalten	
ESC X	Löscht den Bildschirm und schaltet zwischen dem 40- und 80-Spalten-Modus um.
ESC @	Löscht einen Bereich des Bildschirms, beginnend mit der aktuellen Cursorposition bis zum Ende des Bildschirms.
ESC O	Hebt den Zitier-, Umkehr-, Unterstreichungs- und Blitzmodus auf.
Bildlauf (Scrollen)	
ESC V	Scrollt den gesamten Bildschirm eine Zeile nach oben.
ESC W	Scrollt den gesamten Bildschirm eine Zeile nach unten.
ESC L	Aktiviert den Bildlauf, wenn ↓ am unteren Rand des Bildschirms gedrückt wird.
ESC M	Deaktiviert den Bildlauf. Wenn Sie am unteren Rand des Bildschirms ↓ drücken, bewegt sich der Cursor an den oberen Rand des Bildschirms. Wenn Sie jedoch ↑ am oberen Rand des Bildschirms drücken, bleibt der Cursor in der ersten Zeile.

Taste	Aktion
Einfügen und Löschen	
ESC I	Fügt an der aktuellen Cursorposition eine Leerzeile ein und verschiebt alle nachfolgenden Zeilen um eine Position nach unten.
ESC D	Löscht die aktuelle Zeile und verschiebt die Zeilen unterhalb des Cursors um eine Position nach oben.
ESC P	Löscht alle Zeichen ab der Cursorposition bis zum Anfang der aktuellen Zeile.
ESC Q	Löscht alle Zeichen ab der Cursorposition bis zum Ende der aktuellen Zeile.
Cursorbewegung	
ESC J	Bewegt den Cursor an den Anfang der aktuellen Zeile.
ESC K	Bewegt den Cursor auf das letzte Zeichen ohne Leerzeichen in der aktuellen Zeile.
ESC ↑	Speichert die aktuelle Cursorposition. Verwenden Sie ESC ← (neben 1), um ihn zurück an die gespeicherte Position zu setzen. Beachten Sie, dass die hier verwendete ↑-Taste neben dem RESTORE liegt.
ESC ←	Bewegt die Cursorposition auf die Position zurück, die durch ein vorheriges Drücken von ESC + ↑ (neben der Taste RESTORE) gespeichert wurde. Beachten Sie, dass die hier verwendete ←-Taste neben der 1 liegt.

Taste	Aktion
Fenstersteuerung	
ESC T	Legt die linke obere Ecke des Fensterbereichs fest. Alle getippten Zeichen und Bildschirmaktivitäten werden auf diesen Bereich beschränkt. Siehe auch ESC B. Der Fenstermodus kann durch zweimaliges Drücken von CLR HOME deaktiviert werden.
ESC B	Legt die untere rechte Ecke des Fensterbereichs fest. Alle getippten Zeichen und Bildschirmaktivitäten werden auf diesen Bereich beschränkt. Siehe auch ESC T. Der Fenstermodus kann durch zweimaliges Drücken von CLR HOME deaktiviert werden.
Cursorverhalten	
ESC A	Aktiviert den automatischen Einfügemodus. Jede gedrückte Taste wird an der aktuellen Cursorposition eingefügt und verschiebt alle Zeichen auf der aktuellen Zeile nach dem Cursor um eine Position nach rechts.
ESC C	Deaktiviert den automatischen Einfügemodus und kehrt zum Überschreibmodus zurück.
ESC E	Setzt den Cursor in den nicht blinkenden Modus.
ESC F	Versetzt den Cursor in den normalen Blinkmodus.

Taste	Aktion
Glocke	
ESC G	Aktiviert die Glocke, die mit CTRL und G ausgelöst werden kann.
ESC H	Deaktiviert die Glocke, so dass das Drücken von CTRL und G keine Wirkung hat.
Farben	
ESC S	Schaltet den VIC-IV auf den Farbbereich 16-31 um. Auf diese Farben kann mit CTRL und den Tasten 1 bis 8 oder der Taste ◣ und den Tasten 1 bis 8 zugegriffen werden.
ESC U	Schaltet den VIC-IV auf den Farbbereich 0-15 um. Auf diese Farben kann mit CTRL und den Tasten 1 bis 8 oder der Taste ◣ und den Tasten 1 bis 8 zugegriffen werden.
Tabulatoren	
ESC Y	Legt die Standard-Tabulatorstopps (alle 8 Leerzeichen) für den gesamten Bildschirm fest.
ESC Z	Löscht alle Tabstopps. Jedes Tabulieren mit CTRL und I bewegt den Cursor an das Ende der Zeile.

Index

MEGA65-TEAM

Dr. Paul Gardner-Stephen
(highlander)
Gründer
Software und virtuelle Hardware
Pressesprecher und leitender Wissenschaftler

Detlef Hastik
(deft)
Mitbegründer
Vorsitzender
Marketing und Vertrieb

Martin Streit
(seriously)
Video- und Fotoproduktion
Steuer und Organisation
Soziale Medien

Anton Schneider-Michallek
(adtbm)
Verwaltung des Hardware-Pools
Soft-, Hard- und V-Hardware-Tester
Forum-Administrator

Falk Rehwagen
(bluewaysw)
Jenkins Build-Automatisierung
GEOS, Hardware-Qualitätsmanagement

Antti Lukats
(antti-brain)
Entwurf und Herstellung von
Host-Hardware

Dieter Penner
(doubleflash)
Überprüfung und Test der Host-Hardware
Dateihosting

Dr. Edilbert Kirk
(Bit Shifter)
MEGA65.ROM
Englisches Handbuch und Tools

Gábor Lénárt
(LGB)
Emulator

Mirko H.
(sy2002)
Zusätzliche Hardware/Plattformen

Farai Aschwanden
(Tayger)
Datenverwaltung und Tools
Finanzberatung

Gürçe Isıkyıldız
(gurce)
Tools und Erweiterungen
Sound

Oliver Graf
(lydon)
VHDL, englisches Handbuch und Tests

Daniel England
(Mew Pokémon)
Zusätzlicher Code und Tools

Roman Standzikowski
(FeralChild)
Open-ROM

Hernán Di Pietro
(indiocolifa)
Zusätzliche Emulation

Ablaufkontrolle
BEGIN
BEND
CONT
DEF FN
DO
ELSE
END
EXIT
FGOSUB
FGOTO
FN ()
FOR
GOSUB
GOTO
IF
LOOP
NEXT
ON
REM
RETURN
RREG
RUN
SLEEP
STEP
STOP
SYS
THEN
UNTIL
USR ()
WAIT
WHILE

Programmierung
AUTO [1]
CHANGE [1]
DELETE [1]
EDIT [1]
FIND [1]
HELP
HIGHLIGHT
LIST
NEW
RENUMBER [1]
TROFF
TRON

Math. Funktionen
ABS ()
ATN ()
COS ()
EXP ()
INT ()
LOG ()
LOG10 ()
MOD ()
RND ()
SGN ()
SIN ()
SQR ()
TAN ()

Speicher
BANK
CLR
DIM
DMA
EDMA
FRE ()
LET
PEEK ()
PEEKW ()
POINTER ()
POKE
POKEW

Math. Operatoren		
+	*	↑
-	/	

Umrechnen
ASC ()
CHR$ ()
DEC ()
HEX$ ()
STR$ ()
VAL ()

Daten
DATA
READ
RESTORE

Strings
+
ASC ()
CHR$ ()
INSTR ()
LEFT$ ()
LEN ()
MID$ ()
RIGHT$ ()

Logik-Operatoren [3]	
AND	OR
NOT	XOR

Relationale Operat.	
<	<=
=	<>
>	>=

Fehlerbehandlung
EL [2]
ER [2]
ERR$ ()
RESUME
TRAP

Zeit
DT$ [2]
TI [2]
TI$ [2]

Diskette	Abkürzung
APPEND	
BACKUP	
BLOAD	
BOOT	
BSAVE	
BVERIFY	
CATALOG	$ [1]
COLLECT	
CONCAT	
COPY	
DCLEAR	
DCLOSE	
DELETE	
DIR	$ [1]
DIRECTORY	$ [1]
DISK	@ [1]
DLOAD	
DOPEN	
DS [2]	
DS$ [2]	
DSAVE	
DVERIFY	
ERASE	
HEADER	
LIST	
LOAD	/ [1]
LOADIFF	
MERGE	
RECORD	
RENAME	
RUN	
SAVE	← [1]
SAVEIFF	
SCRATCH	
SET	
TYPE	
VERIFY	

Eingabe
GET
GETKEY
INPUT
JOY ()
LPEN ()
MOUSE
POT ()
RMOUSE

Ein- und Ausgabe
CLOSE
CMD
FREAD
FWRITE
GET#
INPUT#
LINE INPUT#
OPEN
PRINT#
PRINT# USING
ST [2]

System
FAST
GO64
KEY
MONITOR
RSPEED ()
SPEED

Sekundäres
OFF
TO

Grafik
BOX
CHAR
CIRCLE
DMODE
DPAT
ELLIPSE
GRAPHIC CLR
LINE
LOADIFF
PAINT
PALETTE
PEN
PIXEL ()
POLYGON
RGRAPHIC ()
RPALETTE ()
RPEN ()
SAVEIFF
SCNCLR
SCREEN
VIEWPORT

Bildschirm
BACKGROUND
BORDER
COLOR
CURSOR
FONT
FOREGROUND
PALETTE
POS ()
PRINT
PRINT USING
RCURSOR
RCOLOR ()
RPALETTE ()
RWINDOW ()
SCNCLR
SPC ()
TAB ()
WINDOW

Sprites
BUMP ()
COLLISION
MOVSPR
RSPCOLOR ()
RSPPOS ()
RSPRITE ()
SPRCOLOR
SPRITE
SPRSAV

Musik
ENVELOPE
FILTER
PLAY
RPLAY ()
SOUND
TEMPO
VOL

[1] Nur im Direktmodus

[2] Reservierte Variable

[3] Auch boolesche Operatoren

() Funktion