Basic 7.0 – das starke Basic des C 128

Wer auf dem C128 in Basic programmiert, wird angenehm überrascht sein. Mit den ausgezeichneten Befehlen kann der Anwender die Fähigkeiten des Computers voll ausnützen.

Betrachtet man die Entwicklung der Commodore-Computer, so standen sich bisher auf getrennten Seiten stets die Homecomputer und die professionellen Systeme gegenüber. Der C128 schließt nun zum erstenmal die dazwischen klaffende Lücke.

Der Commodore 128 Personal Computer ist ein Computer, der sowohl von seiner Leistungsfähigkeit als auch von seinem ergonomischen Design allen Anforderungen eines professionellen Anwenders genügt.

Basic ist effektiver im C128-Modus

Nicht nur das neue Styling rückt den C128 näher an die Profis heran, sondern vor allem zwei der drei möglichen Betriebsarten: der C128-Modus und der CP/M-Modus. Im C128-Modus kann man durch Basic effektiver programmieren, der Wortschatz des Basic 7.0 ist der bisher größte Wortschatz eines Commodore-Computers. Im CP/M-Modus steht die CP/M-Version 3.0 zur Verfügung – eine riesige Zahl von professioneller Software hierzu ist bereits auf dem Markt.

Die drei Betriebsarten werden durch zwei Prozessoren erzeugt: Ein MOS 8502-Prozessor ist für den C128-Modus verantwortlich. Außerdem simuliert er den MOS 6510, will man im C64-Modus arbeiten. Der zweite Prozessor ist ein Z80A von Zilog.

Der Grundwortschatz aller Basic-Versionen war das Basic 2.0 des VC20 und C64. Dieses Basic unterstützte allerdings nicht die hervorragenden Fähigkeiten des C64 – wie Grafik, Sprites, Musik. Ein Musikstück zu programmieren oder einen Sprite zu definieren, bedeutet mühsame Kleinarbeit mit PEEK- und POKE-Befehlen. Dies ist sehr zeitraubend und fehleranfällig.

Basic bietet mehr für Grafik, Sprites und Musik

Deshalb hat man für die nächste Generation von Homecomputern, C16, C116 und Plus4, entsprechende Befehle in den Basic-Wortschatz integriert. Zusätzlich sind neue Floppy-Befehle im Basic 3,5 des C16, C116 und Plus4 enthalten. Wie Tabelle 1 zeigt, enthält das Basic 7.0 alle Befehle der vorausgehenden Basic-Versionen. Zusätzlich sind weitere Musik- und Hilfsbefehle enthalten. Vor allem aber ist die Ergänzung von Sprite-Befehlen sehr wichtig. Während der C16, C116, und Plus4 keine Sprites verarbeiten kann, sondern statt dessen »Shapes« besitzt – dies sind kleine bewegliche Figuren, die mit Hilfe von Grafik-Befehlen gezeichnet werden – arbeitet der C128 wieder mit Sprites, die vom Basic kräftig unterstützt werden.

Der C128 ist mit externem Speicher auf 256 KB und 512 KB erweiterbar. Man spricht hierbei von einer RAM-Disk. Für die Arbeit mit der RAM-Disk stehen spezielle Befehle zur Verfügung.

Insgesamt unterscheidet sich der Wortschatz des Basic 7.0 vom Urgroßvater Basic 2.0 durch über 90 zusätzliche Befehle, Anweisungen, Funktionen und Variablen. Im folgenden sind alle diese kurz aufgeführt und erläutert. Bei einigen Befehlen ist die entsprechende Programmierung in Basic 2.0 gegenübergestellt, um die gravierenden Verbesserungen schwerpunktmäßig aufzuzeigen.

(Renate Benz-Heinbücher/zu).

Floppy-Befehle

APPEND:

Eine bereits auf Diskette existierende sequentielle Datei (SEQ und USR) kann hiermit wieder eröffnet und weiter beschrieben werden. Die neuen Daten werden dabei an das Dateiende angefügt. Beim C64 konnte man eine sequentielle Datei nach dem Schließen mit CLOSE nicht weiter beschreiben. Man muß hierzu erst eine Hilfsdatei eröffnen, auf die man alle Daten der alten Datei herüberholt und die neuen Daten dazuschreibt.

C128-MODUS	C64-MODUS	CP/M-Modus
Basic 7.0	Basic 2.0	CP/M V 3.0
128 KB RAM	64 KB RAM	128 KB RAM
48 KB ROM	20 KB ROM	
512 KB-Speicher extern möglich		512 KB-Speicher extern möglich
CPU MOS 8502	MOS 8502 simuliert MOS 6510	Z80A
Taktfrequenz wahlweise 1 MHz oder 2 MHz	Taktfrequenz 1 MHz	Taktfrequenz wahlweise 4 MHz
wahlweise 40 oder 80 Zeichen/ Zeile	40 Zeichen/Zeile	wahlweise 40 oder 80 Zeichen/Zeile
wahlweise 320x200 oder 640x200 Grafikpunkte	320x200 Grafikpunkte	wahlweise 320x200 oder 640x200 Grafikpunkte
16 Farben	16 Farben	8 Farben
8 Sprites	8 Sprites	

Tabelle 1. Die wichtigsten Daten der drei Betriebsarten auf einen Blick:

BACKUP:

Dupliziert eine Diskette auf einem Doppellaufwerk. Hierbei wird die Zieldiskette formatiert. War die Zieldiskette bereits beschrieben, so sind dann die alten Daten alle gelöscht! Verbindet die Befehle HEADER und COPY.

Um eine Diskette mit einem Einzellaufwerk zu kopieren, ist die Zieldis-

kette zu formatieren. Danach ist das zu kopierende Programm erst in den Arbeitsspeicher mit LOAD-"name", 8 einzulesen und mit SAVE"name",8 auf die Zieldiskette abzuspeichern. Oder man kann ein Kopierprogramm kaufen. Steht über ein Interface ein Doppellaufwerk für den C64 zur Verfügung, so kann man dann mit BACKUP die Quelldiskette kopieren. Ist aber nur ein Einzellaufwerk angeschlossen, so muß das zu kopierende Programm erst in den Arbeitsspeicher mit LOAD"name",8 eingelesen und mit SAVE"name",8 auf die Zieldiskette abgespeichert werden.

BLOAD:

Ein Binärfile (das heißt eine Datei mit binär codierten Werten wie Maschinensprache, Sprites usw.) wird von Diskette in den Speicher geladen. Dabei ist die Speicherbank anzugeben, in die es eingelesen werden soll. Beim C64 nicht möglich.

BOOT:

Ein Binärfile wird von Diskette geladen und bei seiner Startadresse gestartet. Fügt man dem Befehl BOOT keine Parameter an, so sieht das Betriebssystem in Spur 1, Sektor 0 der Diskette in der Floppy mit Gerätenummer 8 nach, was geladen und gestartet werden soll. Diese Information kann man mit Direktzugriffsbefehlen auf Diskette bringen. Beim C64 nicht möglich.

BSAVE:

Speichert einen Hauptspeicherbereich (Maschinensprache, Binärdaten) als Binärfälle auf Diskette. Beim C64 nicht möglich.

CATALOG, DIRECTORY:

Beide Befehle sind identisch und geben den Disketteninhalt auf dem Bildschirm wieder. Der Befehl löscht hierbei nicht das im Arbeitsspeicher befindliche Programm! Beim C64 ist der Disketten-Inhalt abzurufen mit: LOAD"$",8 und auf dem Bildschirm zu zeigen mit LIST. Dabei wird das im Arbeitsspeicher befindliche Programm gelöscht.

COLLECT:

Löscht alle offenen Dateien und gibt den entsprechenden Speicherplatz wieder frei.

CONCAT:

Hängt eine vom Anwender zu bestimmende sequentielle Datei an das Ende einer anderen sequentiellen Datei auf Diskette an.

COPY:

Kopiert eine angegebene Datei unter gleichem oder neuem Namen auf die Diskette im angegebenen Laufwerk einer Doppelfloppy. Arbeitet man innerhalb des gleichen Laufwerks oder mit einem Einzellaufwerk, so dupliziert dieser Befehl die angegebene Datei, doch ist ein neuer Name anzugeben.

Syntax:
COPY (Dquelle)quelldatei TO (Dziel)zieldatei (ON Ugeräteadr).

Beim C64 ist das Kopieren innerhalb einer Diskette ebenso möglich:

```
Syntax:  OPEN 1,8,15
         PRINT#1,"C:neu=alt"
         CLOSE1
```

DCLEAR:

Schließt alle offenen Floppy-Kanäle, nicht die Dateien. Geöffnete Dateien sind vorher mit DCLOSE zu schließen.

DCLOSE:

Schließt eine angegebene oder alle offenen Dateien auf der angeschlossenen Floppy.

Beim C64: CLOSE n.

DLOAD:

Lädt ein Basic-Programm von Diskette in den Arbeitsspeicher. Bei angeschlossenem Doppellaufwerk ist das Laufwerk anzugeben, auf das zugegriffen werden soll.

DOPEN:

Öffnet eine sequentielle oder relative Datei auf der Diskette für Datenaustausch.

DSAVE:

Speichert ein Basic-Programm vom Arbeitsspeicher auf Diskette. Bei angeschlossener Doppelfloppy ist das gewünschte Laufwerk anzugeben.

DS,DS$:

Diese beiden Systemvariablen liefern den Fehlerstatus der Floppy als Code (DS) und als komplette Statusmeldung (DS$) des zuletzt angesprochenen Floppy-Gerätes. Die Statusmeldung beinhaltet den Fehlercode, die Meldung in Worten und Spur und Sektor, bei denen der Fehler auftrat.

Syntax: PRINT DS, DS$.

Beim C64: Fehlerabfrage über Fehlerkanal:

```
10 OPEN 1,8,15
20 INPUT#1, A,B$,C,D
30 PRINT A,B$,C,D
40 CLOSE1
```

DVERIFY:

Vergleicht das Programm im Arbeitsspeicher mit einem Programm auf Diskette.

Beim C64: VERIFY"name",8.

HEADER:

Formatiert eine neue Diskette oder löscht das Inhaltsverzeichnis einer bereits bespielten Diskette.

```
Syntax:  HEADER"name",
         Dlw (,I id) (,Ugerät).
```

Beim C64: OPEN 1,8,15

```
         PRINT#1,"N:name,id"
         CLOSE1
```

RECORD:

Setzt den Schreib/Lese-Zeiger auf eine bestimmbare Position innerhalb einer Relativ-Datei auf Diskette.

RENAME:

Benennt eine Datei auf Diskette um.

```
Syntax:  RENAME alt TO neu
         (;Dlw) (,Ugerät).
```

SCRATCH:

Löscht die angegebene Datei aus dem Disketten-Inhaltsverzeichnis. Damit ist die Datei auf Diskette gelöscht.

```
Syntax:  SCRATCH name (,Dlw)
         (,Ugerät).
```

Die Befehle HEADER und SCRATCH haben, um versehentliches Löschen zu vermeiden, die Sicherungsrückfrage ARE YOU SHURE? eingebaut. Erst nach Bejahen dieser Frage mit Y oder Yes wird formatiert, beziehungsweise gelöscht.

Programmierhilfen

Befehle für RAM-Disk, Maschinensprache-Befehle (Farb-, Print-Befehle, Variablen, Funktionen etc.)

AUTO:

Automatisiert die Zeilennumerierung mit angegebener Schrittweite. Nach AUTO wird die folgende Zeilennummer angezeigt, wenn man am Ende einer Basic-Zeile RETURN drückt.

BANK:

Dieser Maschinensprachebefehl legt eine 64KB Speicherbank für PEEK-, POKE-, SYS- oder WAIT-Anweisungen fest. Syntax: Bank n (n zw. 0 u. 15).

BEGIN...BEND:

Stehen immer im Zusammenhang mit der IF...THEN-Anweisung. Im Gegensatz zu IF...THEN...ELSE (siehe später) schließen BEGIN... BEND, nach THEN stehend, mehrere Basic-Anweisungen ein, die sich auch über mehrere Zeilen erstrecken können.

Beispiel:

```
5 GETA$:IFA$=""THEN5
10 IF A$="J" THEN BEGIN: X=1
20 Y=2: Z=3
30 PRINT X*Y*Z:
   BEND: PRINT"=X*Y*Z"
40 PRINT"WEITER MIT SPACE"
50 .....
```

Die Tastatur wird mit GETA$ abgefragt. Ist der Buchstabe J gedrückt worden, so trifft das Programm auf BEGIN und arbeitet alle danach folgenden Anweisungen

bis BEND ab, fährt nach BEND im Programm fort. Wird irgendeine andere Taste gedrückt, so ist A$="J" logisch falsch und das Programm wird in Zeile 20 fortgesetzt. Trifft es auf BEND, so kann es diese Anweisung nicht verstehen, da es die Anweisung BEGIN übersprungen hatte. Das BEND ist dann eine Aufforderung, in der nächsten Programmzeile fortzufahren. Das heißt, die Anweisung PRINT"=X*Y*Z" wird nur ausgeführt, wenn A$="J", also die Taste J gedrückt wurde.

CHAR:

Schreibt einen String (Zeichenkette) an anzugebender Position auf den Bildschirm. Diese Anweisung arbeitet im Text- und Grafik-Modus.

COLOR:

Legt für die Farbquellen

0 Hintergrund (40 Z-Darstellung)
1 graf. Vordergrund (bei HIRES)
2 Zusatzfarbe 1 bei Multicolor-Grafik
3 Zusatzfarbe 2 bei Mulitcolor-Grafik
4 Rand
5 Textfarbe
6 Hintergrund (80 Z-Darstellung)

eine der möglichen 16 Farben fest.

Farbcodes: 1 schwarz, 2 weiß, 3 rot, 4 grün, 5 violett, 6 dunkelgrün, 7 blau, 8 gelb, 9 hellbraun, 10 braun, 11 rosa, 12 dunkelgrau, 13 grau, 14 hellgrün, 15 hellblau, 16 hellgrau.

Beispiel:

COLOR0, 2:COLOR1, 3:COLOR4,1. Der Hintergrund erscheint weiß, der Vordergrund (Text) rot und der Rahmen schwarz. Beim C64 wird die Hintergrund- und Rahmenfarbe mit POKE-Befehlen festgelegt: POKE53281,2:POKE53280,1. Die Farbe des Textes wird in der PRINT-Anweisung definiert.

DELETE:

Löscht den durch die Zeilennummern zu bestimmenden Bereich des Programms im Arbeitsspeicher.

DO...(UNTIL)..(EXIT)..LOOP..(UNTIL)..bzw.
DO...(WHILE)..(EXIT)..LOOP..(WHILE):

Die Anweisungen DO...LOOP definieren eine Programmschleife, die unendlich oft durchlaufen wird. Wie FOR...NEXT-Schleifen können auch diese geschachtelt sein. Um eine solche DO...LOOP-Schleife zu verlassen, gibt es 3 Möglichkeiten:

1. EXIT: Durch diese Anweisung springt das Programm an der entsprechenden Stelle, an der sich das EXIT befindet, aus der Schleife und setzt das Programm hinter der nach LOOP folgenden Anweisung fort.
2. UNTIL: Das Programm durchläuft die Schleife so lange, bis (englisch:until) der nach UNTIL folgende Ausdruck logisch wahr wird.
3. WHILE: Das Programm durchläuft die Schleife solange der nach WHILE folgende Ausdruck wahr ist (das heißt, bis er falsch wird).

FAST:

Schaltet den Prozessor MOS 8502 von der Taktfrequenz 1 MHz auf die doppelt so schnelle Arbeitsweise im 2 MHz-Takt um. Beim 2 MHz-Betrieb ist die Bildschirm-Anzeige bei der 40 Zeichen-Darstellung abgeschaltet. Der für die 40 Zeichen-Darstellung verantwortliche VIC-Chip ist der gleiche wie im C64. Er kann die schnelle Taktfrequenz nicht mithalten. Für die 80 Zeichen-Darstellung ist ein anderer VIC-Chip zuständig, der schneller arbeitet. Deshalb bleibt die Bildschirm-Anzeige dabei erhalten.

FETCH:

Holt, das heißt bringt eine anzugebende Byte-Menge von der RAM-Disk (Speichererweiterungsbank) in den Arbeitsspeicher.

Beispiel:

FETCH 1000,52000,7,2000 überträgt 1000 Byte aus Speicherbank 7 ab Adresse 2000 in den BASIC-Arbeitsspeicher ab Adresse 52000.

GETKEY:

Vergleichbar mit GET (GET liest Tastatur, ohne zu warten). Diese Anweisung wartet so lange, bis eine Taste gedrückt wird. Diese nur im Programm-Modus anwendbare Anweisung überträgt den ASCII-Code der gedrückten Taste in die angegebene Variable.

G064:

Schaltet den C128 vom C128-Modus in den C64-Modus um. Im Direkt- und Programm-Modus einsetzbar!

HELP:

Bringt – wie die HELP-Taste – nach einer Fehlermeldung die fehlerhafte Programmzeile auf dem Bildschirm und zeigt den fehlerhaften Teil unterlegt (40Z9 beziehungsweise unterstrichen (80Z) an.

IF...THEN...ELSE:

Verzweigt in verschiedene Programmteile, je nach dem logischen Wahrheitsgehalt eines dem IF folgenden Ausdrucks. Ist der nach IF folgende Ausdruck wahr, so arbeitet das Programm nach THEN weiter. Ist dieser Ausdruck falsch, so springt das Programm, wenn kein ELSE vorhanden ist, in die nächste Zeile. Ist aber das ELSE vorhanden, so werden die danach folgenden Anweisungen ausgeführt. THEN und ELSE müsssen in der gleichen Zeile stehen. Nur eine BEGIN-Anweisung (siehe weiter vorn) kann sich über mehrere Zeilen erstrecken.

KEY:

Dieser Befehl bezieht sich nur auf die Funktionstasten. Er fragt die Belegung der Funktionstasten ab oder belegt sie neu. Beispiel:

a) momentane Belegung wird mit KEY ohne Parameter angezeigt.
b) KEY1,"COLOR": F1 schreibt das Wort COLOR auf den Bildschirm. Man gibt dann die gewünschten Parameter ein und erhält nach RETURN die entsprechende Farbgebung.

MONITOR:

Ruft den Maschinensprache-Monitor auf.

PRINT USING:

Formatiert die Ausgabe auf Bildschirm oder Drucker, wobei das Druckformat als Steuerzeichen in einem String abzulegen ist.

Syntax:

PRINT (#filenr.) USING v$; liste (;)
wobei mit v$ das Druckformat definiert wird (Tabelle im Handbuch).

Beispiel:

PRINT USING ".";"23" liefert das Ergebnis 23.00.

PUDEF:

Definiert bis zu vier unterschiedliche Steuerzeichen neu, die in der PRINT-USING-Anweisung bereits festgelegt sind.

RENUMBER:

Numeriert das im Arbeitsspeicher stehende BASIC-Programm ganz oder abschnittweise neu. Anzugeben ist hierbei die Zeilennummer, mit der die neue Numerierung beginnen soll (Default ist 10), die Schrittweite der neuen Numerierung (Default ist 10) und die Zeilennummer im Programm, ab der neu zu numerieren ist (Default ist 10).

RESUME:

Am Ende eines Unterprogramms zur Fehlerbehandlung (siehe TRAP) setzt diese Anweisung das Programm mit der nach RESUME folgenden Zeilennummer fort.

Beispiel:

```
 1000   TRAP 20000
  .
  .
20000   IF ER=74 THEN
            RESUME 30000
  .
```

```
30000  PRINT"FLOPPY ÜBER-
       PRÜFEN"
```

Bei dem Fehler DRIVE NOT READY (Fehlercode 74) erscheint für den Anwender die Aufforderung auf dem Bildschirm, daß er die Floppy überprüfen soll. Zum Beispiel kann es sein, daß sich keine Diskette im Laufwerk befindet, aber ein Zugriff stattfinden sollte.

RREG:

Liest die Prozessorregister A,X,Y und SR in die vom Anwender angegebenen Variablen (am Ende eines mit SYS aufgerufenen Unterprogramms in Maschinensprache).

RUN:

Ergänzt Anwendung des RUN-Befehls:

RUN"programmname" lädt und startet das spezifizierte Programm.

SCNCLR:

Löscht den vom Anwender anzugebenden Bildschirm. Syntax: SCNCLR (n), wobei n=0,...5 dem Grafik-Modus entspricht.

SLEEP:

Hält die Programmausführung für eine bestimmbare Zeit an. Ersetzt eine Warteschleife. Syntax: SLEEP n, mit n zw. 1 und 65535.

SLOW:

Schaltet den Prozessor von der 2 MHz-Taktfrequenz wieder auf 1 MHz um. Der Rechner ist dann nur noch halb so schnell, aber die Bildschirmanzeige bei der 40-Zeichen-Darstellung ist wieder aktiviert.

STASH:

Eine weitere Anweisung für die Arbeit mit der RAM-Disk. Sie arbeitet in umgekehrter Richtung wie FETCH: eine vom Anwender anzugebende Byte-Menge geht vom Basic-Arbeitsspeicher in die RAM-Disk.

Beispiel:

STASH 1000,520000,7,2000 überträgt 1000 Byte ab Adresse 52000 aus dem Basic-Arbeitsspeicher in die Speicherbank 7 ab Adresse 2000.

SWAP:

Überträgt eine anzugebende Menge von Bytes aus dem Arbeitsspeicher auf die RAM-Disk und gleichzeitig wird eine gleich große Anzahl von Bytes aus dieser Stelle der RAM-Disk an die frei gewordene Adresse im Arbeitsspeicher übertragen. Dies entspricht einem Austausch von Bytes.

TRAP:

Bei einem Fehler findet normalerweise eine Programmunterbrechung statt. Die Anweisung TRAP führt jedoch dazu, daß das Programm bei einem Fehler nicht unterbrochen wird, sondern zu der nach TRAP folgenden Zeilennummer verzweigt. Dies unterdrückt eine Fehlermeldung und ermöglicht eine Fehlerbehandlung in einem Unterprogramm. Syntax: TRAP n, wobei die n die Zeilennummer ist, mit der die Fehlerbehandlungsroutine beginnt. Mit RESUME wird dann das Hauptprogramm fortgesetzt.

TRON/TROFF:

Schaltet die Programmablaufverfolgung ein oder aus (trace on beziehungsweise trace off). Dies bedeutet, daß beim Austesten eines Basic-Programmes die aktuelle Zeilennummer schrittweise angezeigt wird.

WINDOW:

Definiert ein Bildschirmfenster durch Angabe der Koordinaten seiner linken oberen und rechten unteren Ecke oder löscht das Fenster. Syntax: WINDOW xlo, ylo, xru, yru (,lö) wobei lö=0 oder 1, je ob löschen (1) oder nicht (0), Default ist 0.

Systemvariablen

EL,ER:

EL liefert die Nummer der Programmzeile, in der ein Fehler aufgetreten ist.

Beispiel:

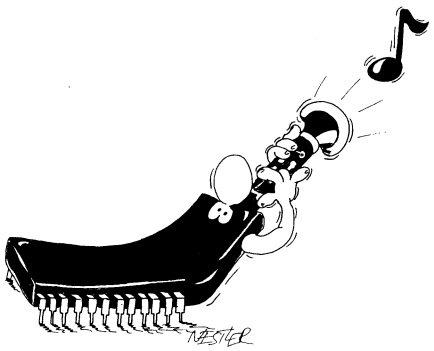
100 IF EL=100 THEN 1000.

ER liefert den Code des zuletzt vom Interpreter diagnostizierten Fehlers.

Beispiel:

100 IF ER=30 THEN 1000.

Wenn die Stop-Taste gedrückt wird, verzweigt das Programm nach Zeile 1000. 30 ist der Fehlercode für einen BREAK.



Funktionen

DEC, ERR$, HEX$, INSTR, JOY, PEN, POINTER, POT, RCLR, RGR, RWINDOW, XOR:

Für Programmierung in Maschinensprache wichtig: DEC gibt den Dezimalwert eines Zahlenwertes in hexadezimaler Schreibweise an.

Beispiel: PRINT DEC("FFFF") liefert das Ergebnis 65535.

HEX$ gibt umgekehrt die hexadezimale Darstellung eines Dezimalwertes n an. (n zwischen 0 und 65535).

Beispiel:

PRINT HEX$(1456) liefert 5B0.

ERR$ gibt die Fehlermeldung in Worten wieder, entsprechend dem Fehlercode n (n zwischen 1 und 127).

Beispiel:

PRINT ERR$(ER).

INSTR gibt die Position eines vom Anwender anzugebenden Teilstrings in einem String an (string=Zeichenkette).

Beispiel:

PRINT INSTR("COMMODORE", "DO")

liefert das Ergebnis 6.

JOY gibt einen Wert für die Position des Joysticks und den Zustand des Feuerknopfes an. Die den Werten entsprechenden Zustände sind dem Handbuch zu entnehmen.

Beispiel:

PRINT JOY(2) liefert 135: Joystick 2 steht bei gedrücktem Feuerknopf in der Stellung nach links.

PEN gibt die Bildschirmkoordinaten des Lichtstiftes an.

POINTER (für Maschinensprache wichtig) gibt die Speicheradresse des ersten Bytes eines Variablenzeigers in der Speicherbank 1 an. In dieser Speicherbank sind vom Interpreter die Variablen Strings und Felder abgelegt.

POT gibt die Position eines der 4 anschließbaren Drehregler an.

RCLR gibt den Farbcode (1 – 16) für die vom Anwender angegebene Farbquelle an. (siehe COLOR weiter vorn).

RGR gibt den momentanen Grafik-Modus an (0 – 5, siehe GRAPHIC).

RWINDOW gibt die Koordinaten des momentanen Bildschirmfensters an.

XOR liefert das Ergebnis einer Exklusiv-Oder-Verknüpfung zweier ganzer, vorzeichenloser 16-Bit-Werte. Beispiel: PRINT XOR (124,12); XOR(0,12) liefert als Ergebnis 112 12.

Grafik-Befehle:

BOX:

Zeichnet ein Rechteck an angegebener Stelle auf dem Bildschirm und kann dieses auch ausmalen.

Syntax:

BOX (farbquelle), x1, y(, (x2,y2)) (,(winkel)) (,(ausmal)), wobei Farbquelle = 0 Hintergrund bedeutet,
1 Vordergrund,
2 Zusatzfarbe 1,
3 Zusatzfarbe 2.
x1, y1 Koordinaten der linken oberen Ecke, x2,y2 Koordinaten der rechten unteren Ecke. x-Werte liegen zwischen 0 und 320 im HIRES-Modus, zwischen 0 und 160 im Multicolor-Modus, y-Werte liegen zwischen 0 und 199. Der Ursprung liegt in der linken oberen Bildschirmecke. Der Winkel (Grad) gibt die Drehung des Rechtecks um seinen Mittelpunkt an. Default = 0. Der grafische Cursor (auch Pixel-Cursor genannt) befindet sich am Ende des Zeichnens in der rechten unteren Ecke x2,y2.

CIRCLE:

Zeichnet einen Kreis, eine Ellipse oder ein beliebiges Polygon mit dem Mittelpunkt x,y auf dem Bildschirm. Anzugeben ist die Farbquelle, die Mittelpunktskoordinaten, Radius der Hauptachse, Radius der Nebenachse, Winkelangabe für Drehung der Figur, Segmentwinkel.

DRAW:

Verbindet die angegebenen Koordinaten durch eine Linie.

Syntax:

DRAW (farbquelle),(x1,y1) TO x2,y2. Hierbei sind x1,y1 die Koordinaten des Anfangspunktes und x2,y2 die Koordinaten des Endpunktes der Linie. Nur die Länge einer Programmzeile beschränkt die Anzahl der Koordinaten, die durch TO miteinander verbunden werden. Stimmen die Koordinaten x1,y1 und x2,y2 überein, so wird ein Punkt gezeichnet.

GRAPHIC:

Ruft einen der 6 verfügbaren Grafik-Modi auf und belegt den Grafik-Speicher (bit-map) beziehungsweise gibt diesen wieder frei.

Syntax: GRAPHIC modus (,(lösch), textz).

modus:

0 = Text mit 40 Zeichen (default)
1 = hochauflösende Grafik (320x200 Punkte)
2 = hochauflösende Grafik, geteilter Bildschirm
3 = Mehrfarbengrafik (160x200 Punkte)
4 = Mehrfarbengrafik, geteilter Bildschirm
5 = Text mit 80 Zeichen

lösch: 0 BS nicht löschen, 1 BS löschen.

textz: 0–24, legt für den Modus 2 oder 4 fest, in welcher Zeile der Text beginnen soll (Default ist 19). GRAPHIC CLR gibt reservierten Speicher wieder frei.

LOCATE:

Setzt den nicht sichtbaren Pixel-Cursor an gewünschte Stelle auf den Bildschirm.

PAINT:

Füllt eine geschlossene Fläche mit der angegebenen Farbe aus. Koordinaten innerhalb der auszumalenden Fläche sind anzugeben.

SCALE:

Mit dieser Anweisung kann man die Skalierung der Bildschirmkoordinaten verändern. Er schaltet ebenso die Skalierung ein oder aus. Die Grenzen für die neue Skalierung: der maximale x-Wert muß im HIRES-Modus zwischen 320 und 1023, im Mehrfarben-Modus zwischen 160 und 1023 liegen, der maximale y-Wert zwischen 200 und 1023.

Syntax:

SCALE (n)(,xmax, ymax) wobei n = 0 die Skalierung aus- und n = 1 die Skalierung einschaltet.

WIDTH:

Legt die Strichstärke für alle Zeichenanweisungen (wie BOX, CIRCLE...) fest. Einfache oder doppelte Strichstärke ist möglich. Syntax: WIDTH n, mit n = 1 oder n = 2.

Funktionen

RDOT:

Gibt die momentane Bildschirm-Position des Pixel-Cursors an oder den entsprechenden Farbquellcode.

Syntax:

RDOT(n)m n = 0: x-Position, n = 1: y-Position, n = 2: Farbquellcode.

Sprite-Befehle

Sprites sind selbst entworfene Figuren, die auf dem Bildschirm bewegt werden können. Sie bestehen aus einer 24x21 Punktmatrix, beziehungsweise aus einer 12x21 Punktmatrix im Mehrfarbenmodus. Der C128 kann wie der C64 acht solcher Sprites gleichzeitig auf dem Bildschirm bewegen.

Im C 128-Modus kann man auf 3 Arten Sprites erzeugen:

1. Mit POKE-Anweisungen wie beim C64. Hierbei wird das Bitmuster der Sprite-Matrix in spezielle Speicherplätze gePOKEt.
2. Mit dem SPRDEF-Befehl.
3. Mit den SSHAPE-, SPRSAV- und SPRITE-Anweisungen.

Zu 2) Der SPRDEF-Befehl ist der schnellste und einfachste Weg, Sprites zu erzeugen. Er schaltet den Sprite-Editor ein. Mit Hilfe dieses Editors kann ein Sprite direkt in einer auf dem Bildschirm erscheinenden Punktematrix definiert werden. Geht man aus dem Editor wieder heraus, so können Sie die Sprite-Werte mit BSAVE auf Diskette speichern oder den Sprite mit SPRITE aktivieren und mit MOVSPR bewegen.

Zu 3) a. Man zeichnet ein beliebiges Bild mit den Befehlen DRAW-, CIRCLE, BOX und PAINT. Die Größe richtet sich nach der 24x21 beziehungsweise 12x21 Punktematrix.

b. Die SSHAPE-Anweisung speichert dieses Bild in einer Zeichenkettenvariablen.

c. Dann überträgt die Anweisung SPRSAV diesen Entwurf in die Sprite-Grafik.

d. Die Anweisung SPRITE aktiviert diesen SPRITE, weist ihm eine Farbe zu, vergrößert ihn.

e. Mit MOVSPR kann man ihn bewegen.

Die Sprite-Befehle genau unter die Lupe genommen

COLLISION:

Aktiviert eine Programmunterbrechung für Sprite-Kollision. Kollidieren zwei Sprites oder ein Sprite mit der Bildschirmanzeige, so kann man mit dieser Anweisung in ein Unterprogramm verzweigen. In diesem Unterprogramm kann die Kollision zum Beispiel akustisch verarbeitet sein. Mit dieser Anweisung ist die Programmunterbrechung aber auch zu inaktivieren.

Syntax:

COLLISION typ (,zeilennummer)
wobei
typ = 1: Sprite-Sprite-Kollision,
typ = 2 Sprite-Anzeige-Kollision,
typ = 3 Lichtstift-Aktivierung zeilennummer muß eine vorhandene Programmzeilennummer sein, mit der bei Unterbrechung das Programm weiter fortfährt.

GSHAPE:

Bringt den Wert eines Strings als binäre Bildinformation auf den Grafik-Bildschirm.

MOVSPR:

Bewegt ein Sprite mit einer angegebenen Geschwindigkeit und in gegebener Richtung. Ebenso setzt dieser Befehl einen Sprite an die gewünschte Position auf dem Bildschirm.

SPRCOLOR:

Definiert im Mehrfarben-Modus die Zusatzfarben.
SPRDEF:
Ruft den Sprite-Editor auf. Es erscheint ein Raster von 24x21 Punkten. In diesem Raster erzeugt der Anwender seinen Sprite indem er jeweils einen Punkt setzt oder nicht. Die Farbe des Sprites ist mit den Farbtasten der Tastatur zu bestimmen, die Ausdehnung in x- und / oder y-Richtung ist ebenfalls ganz einfach mit den Tasten x und y zu erreichen und vieles mehr. Mit BSAVE kann man darauf diesen Sprite oder diese Sprites auf Diskette speichern und mit BLOAD wieder einladen.
SPRITE:
Legt die folgenden Eigenschaften für einen Sprite fest: Farbe, Priorität, Dehnung, Modus.
Beispiel: SPRITE3,1 aktiviert (zweite 1) das Sprite Nummer 3, das heißt Sprite 3 ist dann auf dem Bildschirm zu sehen.
SPRSAV:
Überträgt den Sprite-Entwurf (mit Hilfe von Zeichenanweisungen), das heißt die String-Variable an die Sprite-Grafik. Aber auch umgekehrt speichert er ein bestimmtes Sprite als Punkteraster in eine Stringvariable.
SSHAPE:
Speichert den Inhalt eines angegebenen Bereichs auf dem Grafikbildschirm (zum Beispiel Entwurf eines Sprites) als binäre Bildinformation in eine Zeichenkettenvariable.
Beispiel:
1000 SSHAPE A$,11,10,34,31
1010 SPRSAV A$,1.
Zeile 1000 speichert das Shape (Rechteck mit den Koordinaten 11,10 für linke obere Ecke und 34,31 für rechte untere Ecke) als binär codierte Information im String A$. Zeile 1010 speichert den Inhalt von A$ in das Sprite 1.

Funktionen

BUMP:
Gibt die Nummer des Sprites (1–8), das seit der letzten BUMP-Abfrage entweder mit einem anderen Sprite (n=1) oder mit der Bildschirmanzeige (n=2) kollidiert ist.
Syntax: BUMP(n).
RSPCOLOR:
Gibt den Farbcode der Zusatzfarbe für Sprites.
RSPPOS:
Gibt für gewünschtes Sprite Bildschirmposition oder momentane Geschwindigkeit an.
Syntax:
v=RSPPOS(n,m), wobei n=1,..,8 das zu untersuchende Sprite definiert und m=0 aktuelle x-Koordinate, m=1 y-Koordinate, m=2 Geschwindigkeit liefert.
RSPRITE:
Gibt für gewünschten Sprite dessen zum Beispiel mit SPRITE definierte Eigenschaften an, wie Farbe, Dehnung etc.

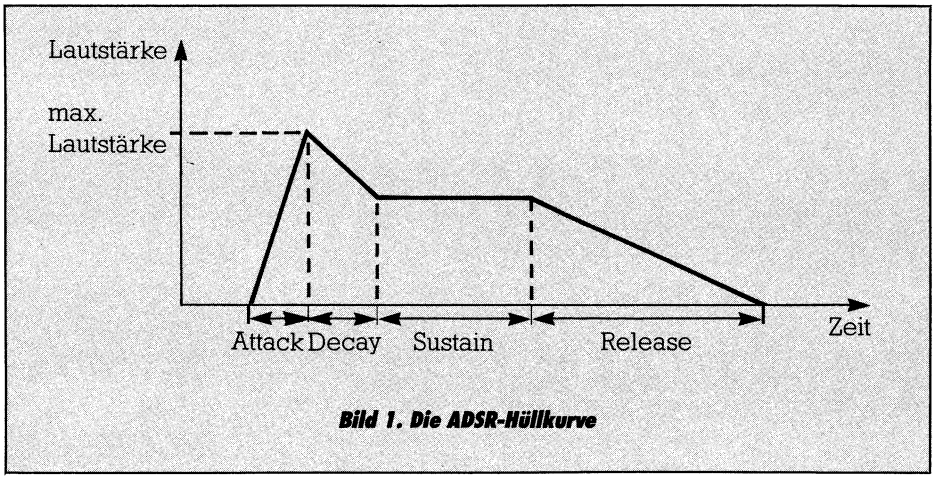


Bild 1. Die ADSR-Hüllkurve

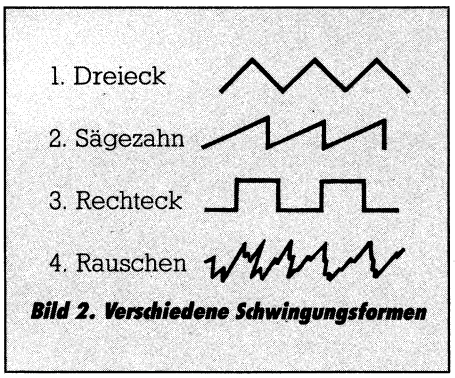


Bild 2. Verschiedene Schwingungsformen

Musik-Befehle

Verschiedene Parameter beeinflussen die Klangfarbe eines Tones: die ADSR-Parameter, die Wellenform, der Filter. Der Klangbaustein des C128 (SID-Chip) erlaubt eine Programmierung dieser Parameter. Hierfür stehen eine Reihe von Basic-Vokabeln zur Verfügung.
ENVELOPE:
Mit ENVELOPE ändert man die Klangfarbe eines Tones. Dazu benötigt der Befehl Werte, die den Lautstärkeverlauf des Tones wiedergeben, die sogenannten ADSR-Parameter (Attack, Decay, Sustain, Release). Diese vier Werte bestimmen die Anschlagzeit (Attack), die Abschwellzeit (Decay), die Haltezeit (Sustain) und die Ausklingzeit (Release) des Tones (siehe Bild 1).
Diesen Lautstärkenverlauf eines Tones bezeichnet man als Hüllkurve.
Die Klangfarbe eines Tones ist aber auch über die Wellenform beeinflußbar. Der C128 kann vier verschiedene Tonwellen erzeugen (Bild 2).
Bei ENVELOPE ist eine dieser vier Wellenformen anzugeben.
Der Befehl ENVELOPE kann durch Angabe der Impulsbreite für die Rechteckwelle die Klangfarbe nochmals verändern.
Mit ENVELOPE erstellt man also eigene Hüllkurven und Klangformen (zehn sind möglich). Der C128 hat aber bereits die Klangfarben für zehn verschiedene Musikinstrumente vorprogrammiert, das heißt die oben aufgezählten Parameter für diese Instrumente sind bereits mit entsprechenden Werten einprogrammiert.
Diese Instrumente sind:
Klavier
Akkordeon
Zirkusorgel (Calliope)
Trommel
Flöte
Gitarre
Cembalo
Orgel
Trompete
Xylophon
Mit ENVELOPE können Sie diese vordefinierten Parameter jedoch abändern.

FILTER:
Der SID-Chip simuliert drei Filtertypen, die die Wellenform hervorheben oder eliminieren:
1. Tiefpaß-Filter
Er läßt niedrige Frequenzen, das heißt die Frequenzen unterhalb der angegebenen Grenzfrequenz durch und sperrt die hohen, oberhalb der Grenzfrequenz liegenden Frequenzen (siehe Bild 3). (Ton dumpf).
2. Hochpaß-Filter
Gerade umgekehrt läßt dieser die hohen Frequenzen durch, während er die unter der Grenzfrequenz liegenden Frequenzen sperrt (siehe Bild 4). (Ton scharf).
3. Bandpaß-Filter

Nur Frequenzen innerhalb des angegebenen Frequenzbereiches kommen durch (siehe Bild 5) : (Ton hohl). Der entsprechende Filter ist mit der FILTER-Anweisung ein- oder auszuschalten. Alle drei Filtermöglichkeiten kann man auch kombinieren.

PLAY:

Mit diesem Befehl können Sie Noten spielen, die in einem String definiert sind. In diesem String sind außer den Noten aber auch Steuerzeichen enthalten, die Anweisungen für die Tonerzeugung oder für das Abspielen enthalten.

Syntax:

PLAY A$ oder PLAY "CDEFGAB", letzteres spielt die Tonleiter, die Note B wird im Deutschen als H bezeichnet. Die folgenden Zeichen kann man – als Steuerzeichen – in den String einbetten, um Änderungen vorzunehmen. Diese Steuerzeichen muß man nicht benutzen, sie nutzen nur die Fähigkeiten des Synthesizers besser aus.

On: bestimmt eine von sieben Oktaven (n = 0,..,6),

Tn: legt eine der folgenden zehn Klanghüllkurven fest:
- 0 – Klavier
- 1 – Akkordeon
- 2 – Zirkusorgel
- 3 – Trompete
- 4 – Flöte
- 5 – Gitarre
- 6 – Cembalo
- 7 – Orgel
- 8 – Trompete
- 9 – Xylophon

Un: bestimmt die Lautstärke (n = 0,...,9)

Vn: bestimmt eine von drei möglichen Stimmen

Xn: schaltet das mit der FILTER-Anweisung gewählte Filter ein (n = 1) oder aus (n = 0)

N: eine der Noten A B C D E F G

#N: Note N einen Halbton höher

$N: Note N einen Halbton tiefer

W: Note wird als ganze Note gespielt

H: Note wird als halbe Note gespielt

Q: Note wird als viertel Note gespielt

I: Note wird als achtel Note gespielt

S: Note wird als sechzehntel Note gespielt

.N: Note wird als punktierte Note (die Hälfte ihres Wertes) gespielt

R: setzt Pause

M: wartet, bis die momentan gespielte Musik zu Ende ist.

SOUND:

Ton- und Klangeffekte sind mit SOUND einfach und schnell zu erzeugen. Die SOUND-Anweisung gibt diese dann durch den Lautsprecher des Monitors oder TVs wieder. SOUND benötigt Parameter wie Angabe der Stimme (beziehungsweise Ton), der Frequenz, Dauer des Tones, Richtung (das heißt ob zunehmende, abnehmende oder oszillierende Klangstufe), Maximalfrequenz (für anschwellende Klangeffekte), Stufe (bestimmt Zeitdauer für stufige Klangeffekte), Wellenform, Impulsbreite.

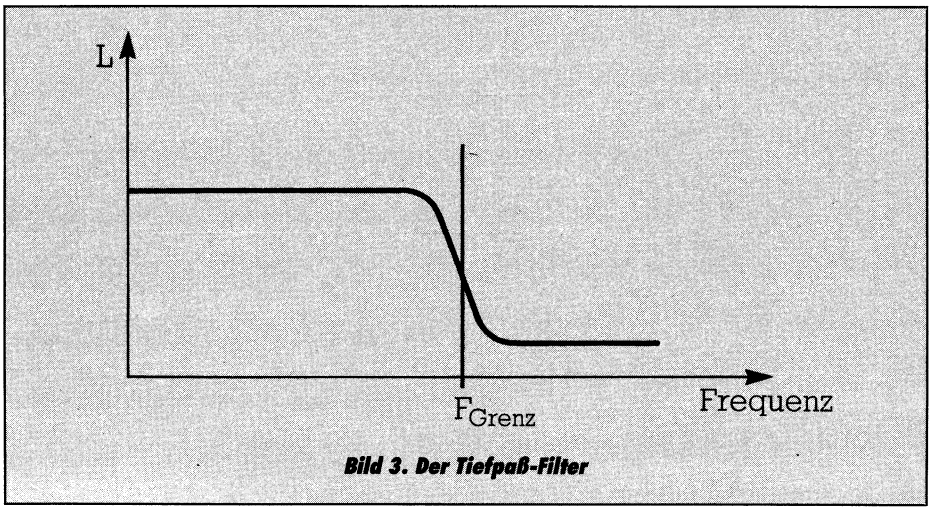


Bild 3. Der Tiefpaß-Filter

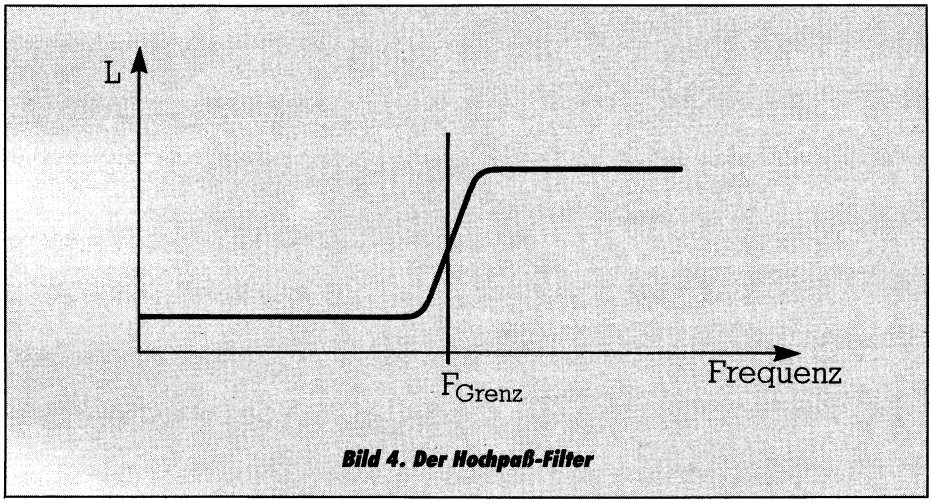


Bild 4. Der Hochpaß-Filter

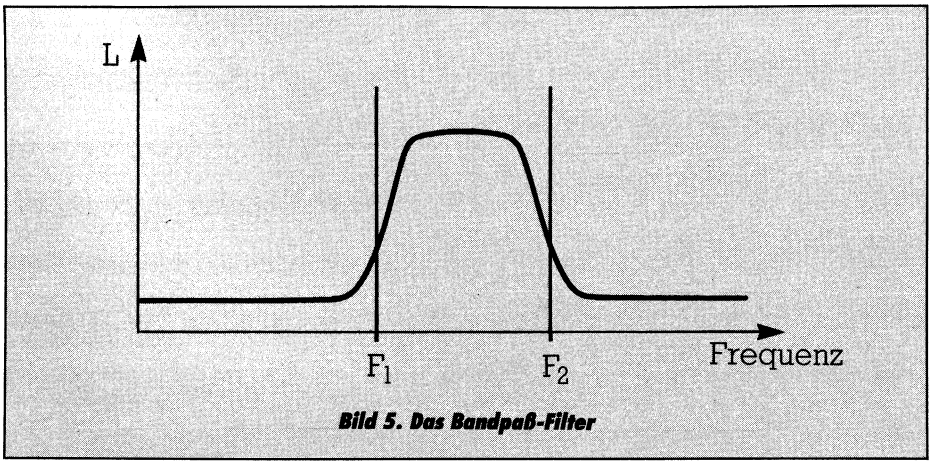


Bild 5. Das Bandpaß-Filter

TEMPO:

Definiert das Spieltempo für die Noten.

Syntax:

TEMPO n, wobei n = 0,...,255 die Dauer angibt, die Dauer entspricht 19.22/n Sekunden.

VOL:

Bestimmt die Lautstärke für die mit der SOUND- und der PLAY-Anweisung entwickelten Geräusche beziehungsweise eines Musikstückes. Syntax: VOL n, wobei n = 0,...,15.

(zu)