AC1-BASIC6 USB

V. 1.1

Ein neues „farbiges“ BASIC für den AC1

Stand: 5. Dez 2016

Inhalt

Eckdaten „AC1-BASIC6 * USB“

Speicheraufteilung:

2000h...5FFFh	16 kB Interpreter (ROM fähig, passend zu Modul 1)
6000h...62FFh	Arbeitszellen und Pufferbereiche
6300h...	BASIC-Programm
1900h...1FFFh	frei für MC-Routinen

Leistungsfähigkeit:

- Variablentypen: numerisch: 48-Bit-Fließkommaarithmetik (11 angezeigte Stellen)
  Wertebereich ...1.7014118346 E+38
  String: max. 255 Zeichen
- Variablennamen bis zu vier Zeichen signifikant
- Großer Befehlsumfang, insgesamt aktuell 142 Grund-Anweisungen
- monitorunabhängige USB-Schnittstelle (mit Unterverzeichnisunterstützung)
- einfache Programmierung farbiger Ausgaben (COLOR-Anweisung)
- Fenstertechnik (WINDOW-Anweisung)
- Umgang mit Datum & Zeit durch Auslesen der RTC
- schnelle MC-Routinen für Linie/Kreis in Blockgrafik
- mit Ausnahme der farbbezogenen Anweisungen auch auf AC1 ohne Farb-BWS lauffähig
- lauffähig im EPROM (Modul1-kompatibel)
- Schattenseiten:
  - fehlende Kompatibilität der Programme (GS-BASIC Import aber möglich)
  - etwas geringere Verarbeitungsgeschwindigkeit als GS-BASIC

Programmstart:

- wenn händisch in RAM geladen: J 2000 (kalt) bzw. J 2003 (warm)
- wenn als EPROM-Variante in Modul1: „b“ (kalt) bzw. „r“ (warm)
  (Hinweise beachten!!!)

- Ist ein COLOR-BWS verfügbar, so wird die Titelzeile wie im Bild farbig dargestellt. Für die Beantwortung von „Highest Memory“ gibt es folgende Möglichkeiten:
  - nur ENTER: übernimmt den voreingestellten Maximalwert (siehe Systemadressen)
  - Eingabe einer sinnvollen RAM-Obergrenze im Bereich von ca. 26000...65635, auch als Hexzahl (ca. &6590...&FFFF)
  - Alle übrigen (falschen) Eingaben führen zu neuer Abfrage.

BASIC verlassen: BYE

Im folgenden sollen nun die Neuerungen und wesentlichsten Änderungen beschrieben werden. Eine Übersicht aller Anweisungen und Funktionen ist der Anlage zu entnehmen.

USB-Schnittstelle

Voraussetzung ist ein VDIP, Basisadresse der USB-PIO: #FC. Alle USB-Kommandos sind eingebaut und funktionieren unabhängig von der Monitorversion (lauffähig ab Mon. 3.1).

LOAD

Laden Basicprogramm, Syntax: **`LOAD “name“`**
LOAD ermöglicht (Meldung „merging...“) auch ein Anhängen weiterer Programmteile, wenn bereits ein Programm im Speicher ist. Beachten:

- Nachzuladende Programme müssen höhere Zeilennummern besitzen.
- Bevor ein nachgeladenes Programm gelistet oder gestartet wird, ist unbedingt "0 REM" auszuführen, sonst keine ordnungsgemäße Funktion oder gar Absturz...
- Soll nicht angehängt werden (Normalfall), ist vor LOAD ein NEW sinnvoll.

SAVE

Sichern Basicprogramm, Syntax: **`SAVE “name“`**
Ist eine gleichnamige Datei bereits vorhanden, so erfolgt eine Rückfrage:
`FILE ALREADY EXIST'S OVERWRITE (Y)`
Nur mit „Y“ wird dann gespeichert (überschrieben), ansonsten abgebrochen.

RUN

Laden + Starten Basicprogramm, Syntax: **`RUN “name“`**

DIR

Anzeige USB-Inhaltsverzeichnis, Syntax: **`DIR`**
Dateimaske nicht möglich.

CD

Wechsel in ein Unterverzeichnis, Syntax:
`CD “name“`
`CD “..“` eine Ebene hoch
`CD` ohne Parameter: wechseln ins root-Verzeichnis
Bei erstmaliger Verwendung einer USB-Anweisung befindet man sich im root-Verzeichnis!

Datei-/Verzeichnisnamen können als Literal oder als Stringvariable angegeben werden. Der Name darf max. 8 Zeichen lang sein, längene Namen werden absgeschnitten. Nicht erlaubt sind Leerzeichen, Umlaute und Sonderzeichen wie /, =,?,* u.a. Ein solcher Versuch wird mit „USB Error“ quittiert und abgebrochen.

Die Speicherung der BASIC-Dateien erfolgt als tokenisiertes Programm mit der Endung „ABC“ (**A**c1**B**asic**C**olor). Die Endung ist nicht mit anzugeben. Das Erstellen/Löschen von Verzeichnissen und das Löschen von Dateien ist aktuell nicht vorgesehen; das muss bei Bedarf am PC erfolgen.

Ist zum Zeitpunkt eines der USB-Kommandos kein USB-Medium präsent, so erfolgt eine Fehlermeldung mit „ND“ (No Disk) und „USB ERROR“. In BASIC kann trotzdem (natürlich ohne USB-Zugriffe) gearbeitet werden. Nach Anstecken des USB-Mediums steht dieses dann beim nächsten Kommando auch zur Verfügung.

LOAD und SAVE kehren nach der Ausführung in den Direktmodus zurück. DIR und CD können auch innerhalb eines Programmes verwendet werden. Tritt dabei kein Fehler auf, wird das Programm mit der nächsten Anweisung fortgesetzt.

Für das Laden und Speichern von Binärdateien gibt es die gesonderten Anweisungen BLOAD und BSAVE, siehe dort!

Farbe

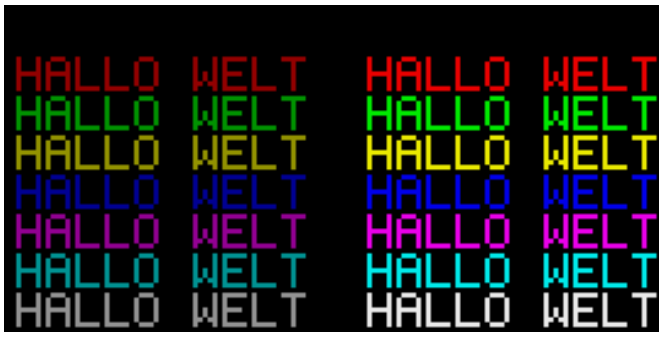
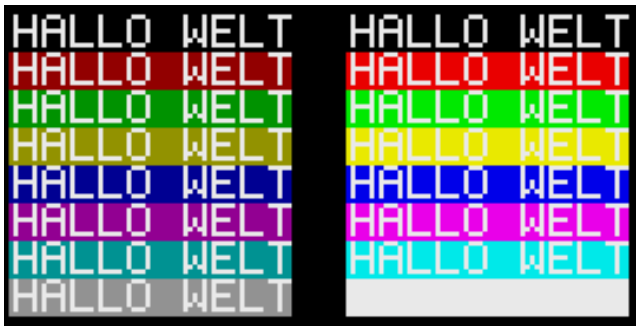
Der AC1 ermöglicht mit dem COLOR-BWS farbige Darstellungen. Das Prinzip besteht in einem zweiten BWS (auch auf 1000...17FF), welcher das Farbattribut für je ein Textzeichen enthält. Hardwaremäßig werden beide Speicher dann zur gewünschten Ausgabe verknüpft. Die CPU kann zu einem Zeitpunkt immer nur auf einen der beiden Bereiche zugreifen. Welcher das ist, entscheiden die Werte am Farbport (#F0).

Die Grundfarben für Vorder- und Hintergrund werden bei RESET des Monitors bzw. mit dem Monitorbefehl „!“ eingestellt.

Die Programmierung von Farbe erfordert nun das parallele Beschreiben des Farb-RAMs. Jedes einzelne Text-Zeichen lässt sich getrennt mit Vorder- und Hintergrundfarbe einstellen. Dafür ist ein Byte im Farb-RAM entsprechend zu setzen. Mit 4 Bit kann man neben schwarz jeweils 7 Farben (additive Farbmischung) in 2 Helligkeitsstufen erzeugen.

Bit	7	6	5	4	3	2	1	0
	Hintergrundfarbe				Vordergrundfarbe			
Farbe	intensiv	blau	grün	rot	intensiv	blau	grün	rot

Aus den o.a. Bitkombinationen ergeben sich für die einzelnen Farben Werte zwischen 0 ... 15:

	normal	**als Vordergrundfarbe, Hintergrund schwarz**	**intensiv**	
0	Schwarz		Schwarz	8
1	Rot		Rot	9
2	Grün		Grün	10
3	Gelb		Gelb	11
4	Blau		Blau	12
5	Magenta		Magenta	13
6	Cyan		Cyan	14
7	Weiß (Grau)		Weiß	15
		als Hintergrundfarbe/ Vordergrund Int.weiß		
0	Schwarz		Schwarz	8
1	Rot		Rot	9
2	Grün		Grün	10
3	Gelb		Gelb	11
4	Blau		Blau	12
5	Magenta		Magenta	13
6	Cyan		Cyan	13
7	Weiß (Grau)		Weiß	15

Diese BASIC-Version unterstützt nun bequem farbige Ausgaben. Dazu wurde die Anweisung COLOR (analog KC85) implementiert, die CLS-Anweisung entsprechend geändert und eine GETCL()-Funktion hinzugefügt.

COLOR	
Mit der Anweisung wird Vorder-/Hintergundfarbe der folgenden Ausgaben eingestellt.	
COLOR 1,2	stellt Vordergrundfarbe auf 1 und Hintergrund auf 2
COLOR 5	stellt nur Vordergrundfarbe auf 5, bisheriger Hintergrund bleibt
COLOR	es werden die Farb-Grundeinstellungen (Monitor-Reset bzw. !) benutzt
CLS	
CLS	löscht den Bildschirm in der RESET-(Hintergrund-)Farbe
CLS 5	löscht den Bildschirm in der gewählten (Hintergrund-)Farbe 5
GETCL()	
GETCL(&1000)	gibt das Farbbyte auf BS-Adresse (adr) zurück

Höhere Parameterwerte als 15 werden als Fehler mit „ILLEGAL FUNCTION“ quittiert. Anstelle der o.a. beispielhaften Festwerte können auch numerische Variablen verwendet werden.

Weiterhin können genutzt werden:

A=PEEK(&6006)	ermitteln aktuelles COLOR-Byte	
A=PEEK(&6007)	ermitteln aktuelle CLS-Hintergrund-Farbe	
OUT (&F0),A	Beschreiben des Farbports	siehe dazu Anleitung zum Farb-BWS!
A=INP (&F0)	Lesen des Farbports	

Alle Rückgabewerte von Farben entsprechen dem o.a. Schema. Um aus dem Kombi-Wert die Vorder- und Hintergrundfarbe zu extrahieren, kann man wie folgt vorgehen:

```
A=GETCL(&1000)     :REM holt das für Adresse #1000 aktuell geltende Farbbyte
VG=(A AND 15)      :REM Vordergrundfarbe
HG=(A AND 240)/16  :REM Hintergrundfarbe
```

Die mit COLOR gewählte Farbkombination gilt für alle nachfolgenden Ausgaben, bis eine neue COLOR-Anweisung ergeht.

- Ausgaben können mittels PRINT auf den BWS erfolgen. Dazu wird für die Zeitdauer der BASIC-Sitzung der RST10h-Aufruf umgeleitet und das quasi-parallele Beschreiben des Farb-RAMs an der entsprechenden Bildschirmposition eingefügt. Beim Verlassen wird wieder der „normale“ RST10-Aufruf hergestellt. Um inverse Farben zu erzeugen, können neben der Vertauschung der COLOR-Parameter auch die Steuerzeichen #11 (invers ein) und #10 (invers aus) benutzt werden. Man beachte aber den bekannten Nebeneffekt, dass die beiden Steuerzeichen je eine Ausgabeposition (Leerzeichen vor und nach dem Text) auf dem Bildschirm belegen.
- Auch die POKEs (DOKEs nicht!) in den Bereich des Bildschirms schreiben parallel zum ASCII-Byte das mit der letzten COLOR-Anweisung gesetzte Farbbyte.
- Blockgrafik ist ebenfalls in der vorher gewählten Farbe möglich. Es wird jedoch aktuell immer die per CLS [n] eingestellte Hintergrundfarbe verwendet. Ansonsten ergäben sich unschöne „Fehlfarben“ an den Begrenzungen der „Block-Pixel“. Die Vorder- und Hintergrundfarbe ist nur jeweils für ein Zeichen (Block 2x2 Pixel) getrennt setzbar. Daher ist es nicht möglich, mit SET/RESET sowie DRAW und CIRCLE innerhalb eines Blockes zwei oder mehr verschiedene Vordergrund-Farben zu setzen. Störende Effekte bei der Überlappung von mit verschiedenen Farben gezeichneten Pixeln/Linien/Kreisen lassen sich deshalb nicht ganz vermeiden.

Besonderheiten und Einschränkungen:

1. Die mit CLS n erzeugte Hintergrundfarbe der Fläche ist unabhängig von der mit COLOR eingestellten Hintergrundfarbe für Ausgaben von Zeichen.

2. Sind Ausgaben im Direktmodus mal nicht sichtbar, so wurde im Programm versehentlich für Vorder- und Hintergrund die gleiche Farbe eingestellt. Dann hilft ein „blind“ eingetipptes COLOR (ohne Parameter), um auf die RESET-Farben zurückzustellen.

3. Das RESET-Farbbyte auf Adresse #181F der Monitor-Arbeitszellen dient zur Entscheidungsfindung, ob ein COLOR-BWS installiert ist oder nicht: 00 => kein Farb-BWS. Leider gehen Monitor 8.x und 10/88 damit nicht konform, bei ihnen steht dort „AF“.
   - Diese Farbkombination „AF“ (HG: intensiv-grün, VG: intensiv weiß) kann also nicht als RESET-Farbbyte benutzt werden. Das wird aber kaum jemand stören, denn es tut den Augen weh :-)
   - Alle Farb-Anweisungen bleiben dann wie bei den anderen (nicht-Farb-)Monitoren wirkungslos.

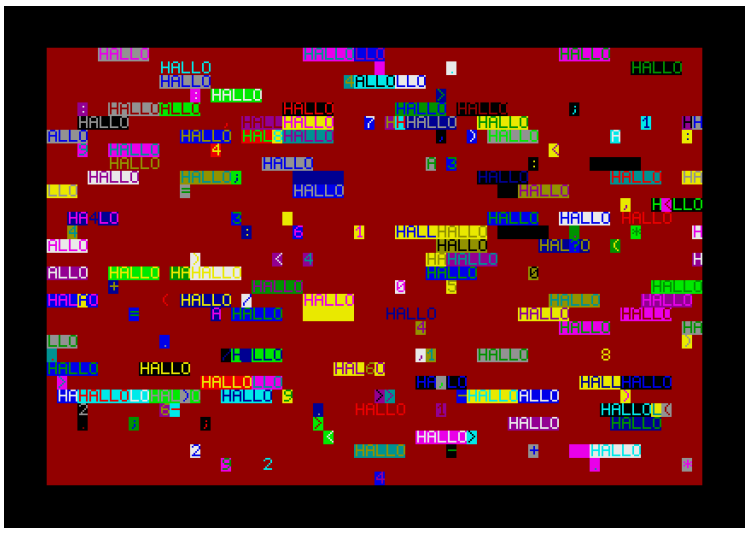
4. Hinweise für die Umsetzung von KC85-Basicprogrammen:
   - `PAPER h` `=> CLS h` Hintergrundfarbe einstellen
   - `INK v` `=> COLOR v` Vordergrundfarbe einstellen
   - `COLOR v,h` `=>` Vordergrund- und Hintergrundfarbe einstellen
   - `PRINT COLOR 0,4;A$` unter *AC1BASIC6* unmöglich; COLOR-Anweisung ist vor PRINT zu stellen und gilt bis zum nächsten COLOR v,h
   - Alle Farbwerte (mit Ausnahme von schwarz und weiß) sind anders als am KC85!

DEMO 1:

```
10 CLS
20 FOR I=0 TO 15
30  COLOR: PRINTI,"VG: ";
40  COLORI:PRINT"HALLO WELT";
50  COLOR: PRINT"     HG: ";
60  COLOR15,I:PRINT"HALLO WELT"
70 NEXT
80 PAUSE
```

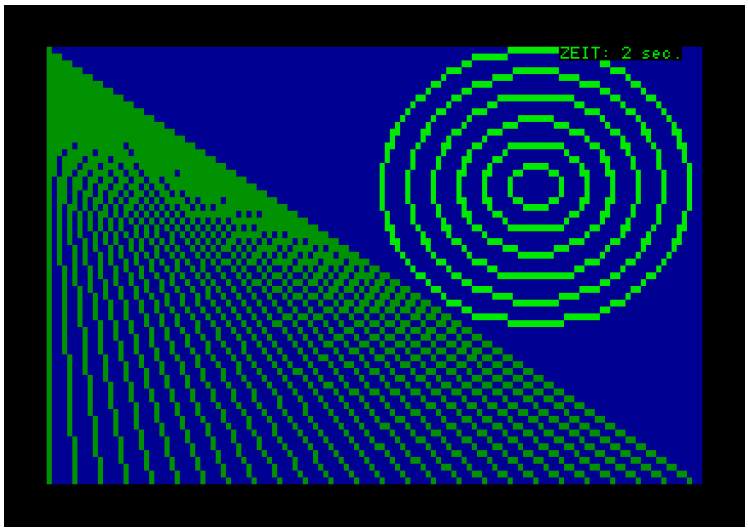
DEMO 2:

```
10 CLS1
15 COLOR
20 FOR I=1 TO 100
25  COLV=16*RND(1):COLH=16*RND(1)
30  X=64*RND(1):Y=30*RND(1)
35  Z=64+26*RND(1)
40  COLOR COLV,COLH
45  POKE 4096+X+Y*64,Z
50  CURSORX,Y+1:PRINT"HALLO";
55 NEXT
60 PAUSE:COLOR:CLS
```



DEMO 3:

```
10 GOSUB 90
15 CLS4
20 T1=FNTICK(0)
25 COLOR2
30 FOR I=0 TO 127 STEP5
35  DRAW0,0,I,63
40 NEXT
45 COLOR10
50 FOR I=0 TO 30 STEP5
55  CIRCLE95,20,I
60 NEXT
65 T2=FNTICK(0)
70 CURSOR50,0
75 PRINT"ZEIT:";
80 PRINT INT(.5+(T2-T1)) ;"sec."
85 PAUSE:CLS:END
90 REM-TICKS
95 DEF FNTICK(TI)
100 A$=TIME$(2)
105 TI=VAL(RIGHT$(A$,2))
110 TI=TI+60*VAL(MID$(A$,4,2))
115 TI=TI+3600*VAL(LEFT$(A$,2))
120 FNEND TI
125 RETURN
```



Demonstration der schnellen MC-Routinen DRAW und CIRCLE.

Um obiges Bild mit herkömmlichen SET- Anweisungen zu erzeugen, werden ca. 250 sec. benötigt.

Die benutzerdefinierte Funktion FNTICK verwendet eine RTC-Abfrage (TIME$) und dient zur Zeitmessung.

Fenster

Mit dieser BASIC-Variante kann nunmehr auch am AC1 (farbig) mit Fenstern gearbeitet werden:

WINDOW	
`WINDOW z1,z2,s1,s2`	definiert ein Fenster zwischen Zeile z1...z2 (z1 < z2 = 0...31) und Spalte s1...s2 s1 < s2 = 0...63)
`WINDOW`	hebt die letzte Fensterdefinition auf und stellt das maximale Fenster ein

- Alle nach einer WINDOW-Anweisung folgenden **PRINT-Ausgaben** erscheinen nur noch in diesem Fenster. Die z.B. vom KC85 her bekannte Möglichkeit, bei aktiviertem Fenster mittels „PRINT AT“ außerhalb des Fensters zu drucken, existiert bei *AC1-BASIC6* nicht!
- Eine CLS n - Anweisung löscht nur den Inhalt des aktiven Fensters und stellt bei Bedarf dessen Hintergrundfarbe ein.
- Würden Ausgaben die untere Zeile überschreiten, so wird der Fensterinhalt hochgerollt, inklusive des ggf. verwendeten Farb-RAMs. Als Hintergrundfarbe wird dabei die des letzten CLS n benutzt.
- Wird ein neues Fenster definiert, so bleiben alte Bildschirminhalte (sofern nicht vom neuen Fenster überlagert bzw. vorher gelöscht) erhalten.
- POKEs auf den Bildschirm und Zeichenelemente (SET/RESET/LINE/CIRCLE) bleiben (wie beim KC85) fensterunabhängig.
- Auftretende Fehlermeldungen und direktpositionierte Ausgaben maximieren automatisch das letzte Fenster.

Innerhalb eines Fensters sind folgende **Steuerzeichen** wirksam (Ausgabe z.B. per PRINT CHR$())

`01h`	Kursor in linke obere Ecke des Fensters („home“)
`02h`	Fenster ab Kursor löschen*)
`03h`	Zeile ab Kursor löschen*?
`06h`	Kursor an Zeilenanfang
`08h`	Kursor nach links
`09h`	Kursor nach rechts
`0Ah`	Kursor runter
`0Bh`	Kursor hoch
`0Ch`	„home“ + Löschen Fenster*)
`0Dh`	neue Zeile
`0Eh`	zwingend gefolgt von 4 Ziffern (0000...3163 =Zeile und Spalte) Kursor-Direktpositionierung (hebt gleichzeitig ein kleineres Fenster auf!)
`10h/11h`	normal und invers

*)
Alle Löschfunktionen verwenden die letzte mit CLS n definierte Farbe.

Blockgrafik (Codes 00...0Fh) ist nur per SET- oder POKE-Anweisung möglich, nicht mit PRINT CHR$(). Nur die Zeichen mit den Codes 12h...FFh werden als ASCII-Zeichen bzw. Pseudo-grafikzeichen ausgegeben. Ihr Aussehen ist abhängig vom installierten Zeichengenerator.

Besonderheit:

- Die Parameterreihenfolge bei CHR$(14) ist (wie in GS-BASIC auch) genau umgekehrt:
  - CURSOR spalte, zeile
  - CHR$(14);“zeilespalte“;
- Direktpositionierungen per CHR$(14) oder CURSOR x,y maximieren automatisch das aktuelle Fenster.

CURSOR 10,5:PRINT „HALLO WELT“ v---Steuerzeichen 0Eh (GSB-Import)
PRINT **CHR$(14)**;“0510HALLO WELT“ bzw. PRINT“_0510HALLO WELT“

```
10 REM DEMO-WIN
20 WINDOW:CLS4
30 PRINT"FENSTER1 - MAXIMAL"
40 FOR I=2 TO 15 STEP 3
50  WINDOW I,31-I,I,63-I:CLS I/2
60  PRINT"FENSTER";I
70 NEXT
80 FORI=1TO15
90  COLOR15-I,I
100  PRINT"VORDERGRUND";15-I;"HINTERGRUND:";I
110 NEXT
120 COLOR:PRINT:PRINTCHR$(17);"SYSTEMFARBE INVERS";CHR$(16)
130 COLOR15,4
140 PRINTCHR$(14);"0330POSITIONIERT: HALLO WELT"
150 WINDOW18,24,20,45:CLS
160 COLOR:PRINT"LETZTES NEUES FENSTER:"
170 PRINT"I J K L"
180 PRINT"ENDE MIT TASTE...":PAUSE:WINDOW:CLS
```

Zeit

TIME$

Ist eine RTC vorhanden (GIDE auf Basisadresse 80h), so kann damit Datum und Zeit bequem ausgelesen werden:

A$=TIME$(1) holt das Tagesdatum nach A$, Format: TT.MM.JJ
A$=TIME$(2) holt die aktuelle Uhrzeit nach A$, Format: HH:MM:SS
A$=TIME$(3) holt den aktuellen Wochentag nach A$, Format: kurz (z.B. „Do.“)

Anmerkungen:

- Das Jahr wird durch die RTC nur zweistellig zur Verfügung gestellt.
- Der Wochentag der RTC ist nicht an das Datum gekoppelt, er wird nur weiter geschaltet!
- Das Stellen der Uhr per AC1-BASIC6 ist nicht vorgesehen.
- Wie oben gezeigt, lässt sich die TIME$-Funktion auch zur (Lauf-)Zeitmessung benutzen

PAUSE

Die vom KC85 übernommene PAUSE-Anweisung ermöglicht das Anhalten des Programms für eine definierte Zeitspanne. Diese wird in 1/10 sek angegeben.

PAUSE 50 wartet 5 Sekunden und setzt dann Programm fort
PAUSE ohne Argument = „ewiges“ Warten

In beiden Fällen kann die Pause durch eine beliebige Taste beendet werden. Dann ist bei Bedarf verfügbar:

- Code der Abbruchtaste: in 1822h
- Restzeit: in 18D6h

Mit dieser Anweisung kann z.B. das Konstrukt „Warteschleife“ `100 IF INKEY$=““ THEN 100` bequem ersetzt werden.

Joystick

JOY

Die Abfrage erkennt einen „parallel zur Tastatur“ (Port 04) angeschlossenen Joystick und benutzt die Monitorfunktion auf 0EB4h. Mit der Funktion **JOY(0)** kann der Zustand des Joysticks abgefragt werden.. Das Argument 0 ist ein beliebiger Dummywert, da nichts an die Funktion übergeben wird. Rückgabewerte sind:

0 = nicht betätigt oder kein Joystick vorh.	4 = links
1 = hoch	8 = rechts
2 = runter	16 = „Feuer“

sowie mögliche Kombinationen daraus (z.B. links + hoch + Feuer = 21)

Beispiel:

```
100 A=JOY(0)
110 IF A=0 THEN 100:REM warten auf Joystickbetätigung
120 IF A=1 THEN PRINT „HOCH“
130 IF A=2 THEN PRINT „RUNTER“
```

Blockgrafik

Die Blockgrafik entspricht im Wesentlichen der von GS-BASIC her bekannten. In der Syntax werden aber im Gegensatz zu GS-BASIC (mit Ausnahme von POINT, da das eine Funktion ist) keine Klammern verwendet.

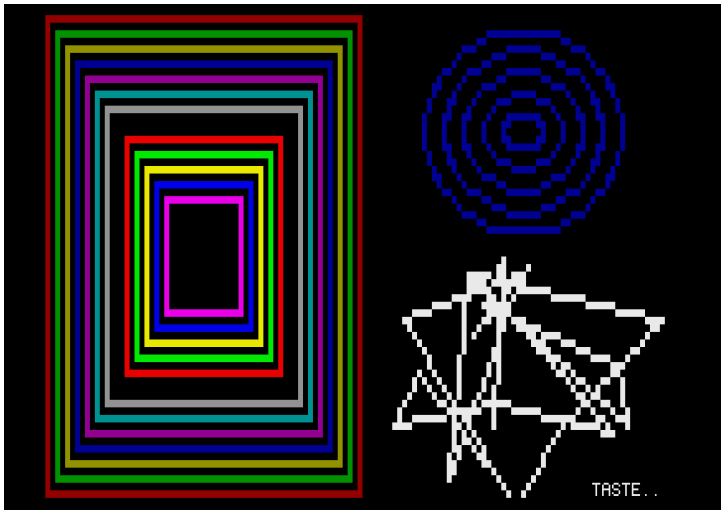
Koordinatenangaben: 0...127 Zeile 0...31 Position 0,0 = links oben
Höhere Werte für Spalte/Zeile werden mit „ILLEGAL FUNCTION“ als Fehler bewertet.

SET/RESET/POINT	
`SET 10,20`	setzt Pixel auf x=10, y=20
`RESET 10,20`	löscht Pixel auf x=10, y=20
`A=POINT(10,20)`	testet, ob Pixel auf x=10, y=20 gesetzt ist Rückgabewerte: 1 = gesetzt, 0= nicht gesetzt

Für ein schnelles Zeichnen von Linien- und Kreisen/Ellipsen sind MC-Routinen implementiert. Es gibt dazu einen „unsichtbaren“ Pixelkursor. Jeweils die letzten benutzten Koordinaten (bei DRAW das Ende-Paar) werden gespeichert und von DRAWTO aus Ausgangspunkt benutzt. CLS [n] setzt den Pixelkursor auf 0,0.

DRAW	
`DRAW 10,10,20,20`	zeichnet eine Linie von 10,10 nach 20,20
DRAWTO	
`DRAWTO 0,0`	zeichnet Linie vom Pixelkursor weiter nach 0,0
CIRCLE	
`CIRCLE 63,31,31[,5]`	zeichnet einen Kreis um x=63, y=31 Mittelpunktkoordinaten werden für DRAWTO aus Ausgang gesetzt. mit viertem Parameter ist Ellipse möglich (Wertebereich 1...ca. 20)

Alle Zeichenbefehle sind durch eine vorherige COLOR-Anweisung auch in Farbe möglich. Einschränkungen siehe dort. Beispiel DEMO4:



Pseudografik

Nichts Neues für GS-Basic-Kundige, aber wenn gewünscht nun auch in Farbe...

`POKE &1000,205` `PRINT CHR$(128)`	Alle Codes >=80h können herkömmlich als Pseudografikzeichen auf den Schirm gebracht werden.
`PRINT" ";`	Das direkte Platzieren von Grafikzeichen in einer PRINT-Anweisung ist ebenso möglich. Beachte Hinweis Grafiktaste!

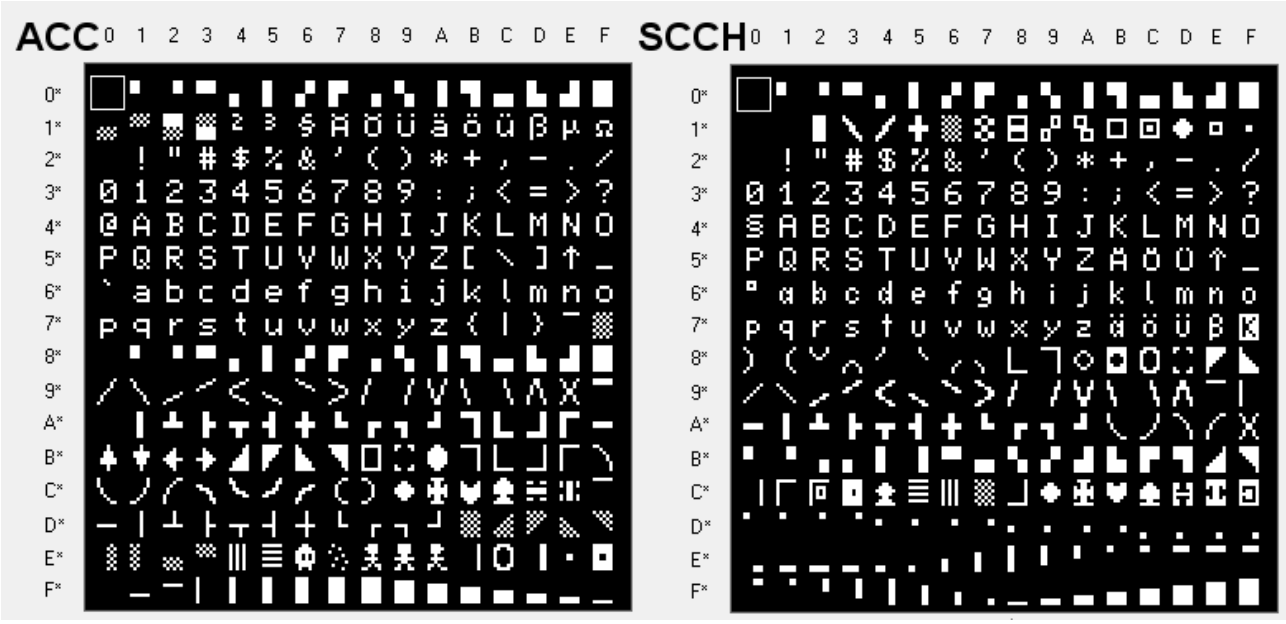
`LIST`
 	
`10 CLS`
`20 PRINT "♠ ♣ ♥ ♦"`
 	
`OK`
	Im Bildschirm-Listing erscheinen die Grafikzeichen...
	
	Auf dem Drucker ist das (ohne speziellen Treiber) nicht möglich, es wird die ASCII-Entsprechung (CODE-80h) gedruckt.

	Mit RUN gibt's natürlich auch Farbe auf dem Schirm, je nach zuvor wirksamer COLOR-Anweisung...

Das Aussehen der Pseudografikzeichen hängt vom installierten Zeichengenerator-EPROM ab. Sind zwei Zeichensätze vorhanden, so kann (wie im Monitor mit ^Z) mit der Anweisung MODE umgeschaltet werden:

MODE 0	„oberer“ Zeichensatz (Grundeinstellung bei Start)
MODE	wechselt zwischen oberem und unterem Zeichensatz

Die beiden (erweiterten) Zeichensätze im Überblick:



Drucken

Folgende Befehle senden Ausgaben direkt an einen Drucker (ohne parallele Ausgabe an den Bildschirm):

LPRINT	Ausdruck aus Programm heraus drucken
LLIST	Listing drucken

Beim Ausführen von LPRINT und LLIST wird nicht geprüft, ob ein Drucker angeschlossen bzw. online ist. Ggf. hängt das Programm dann und wird erst fortgesetzt, wenn der Drucker sein „bereit“ meldet.

Als Druckertreiber für V.24 fungiert die im jeweiligen Monitor enthaltene Routine. Gesendet wird mit den jeweiligen Einstellungen des Monitors, also standardmäßig: 9600 Baud, 8 Datenbits, 1 Stopp-Bit, keine Parität, kein Handshake.

Mit Kaltstart von *AC1-BASIC6* wird ein Drucker-Init ausgeführt. Im Falle des eingebauten Druckertreibers ist dieser nicht nötig (standardmäßig mit „MS30“ belegt).

Es kann auch ein eigener Druckertreiber verwendet werden, der an folgender Adresse einzubinden ist:

```
#2006 C3 xx xx    Sprung zur Druckerausgabe
#2009 C3 xx xx    Sprung zur Druckerinitialisierung
```

Der Treiber erhält das auszugebende Zeichen im Register C. Er muss die Register IX, HL, DE und BC sichern. Ist eine Druckerinitialisierung nötig, so ist deren Einsprungadresse auf #2009 einzutragen.

Problem: Grafik

- Beim Drucken von Grafik- oder Steuerzeichen (direkt oder in Listings), können je nach Druckertreiber Probleme auftreten. Anstelle der Grafik wird i.d.R. das zugeordnete ASCII-Zeichen (also Grafikzeichen mit Bit7=0) gedruckt.
- Bei Steuerzeichen (0...1F) reagiert der Drucker ggf. mit den entsprechenden Funktionen; es können daher „unerklärliche“ Effekte auftreten.

Druckerbezogen sind weiterhin folgende Befehle, die aber nicht direkt an den Drucker gesendet sondern bei der Ausgabeaufbereitung berücksichtigt werden:

LNULL n	Anzahl Nullzeichen für Drucker setzen, z.B.:
	`LNULL 6,&20 :REM „Linker Rand“, 6 Leerzeichen drucken`
	`LNULL 0 :REM hebt letzte Einstellung auf`

LPOS(0)	Funktion: liefert akt. Spaltenposition Drucker
LWIDTH n	Druckbreite einstellen (nach n Zeichen erfolgt automatisches CRLF)

Maschinencode

Nicht völlig neu, aber in den entsprechenden Originalanleitungen (z.B. GS-BASIC) etwas knapp dokumentiert sind einige Dinge, wozu hierzu ergänzende Ausführungen speziell mit AC1-Bezug erfolgen sollen.

Für den Umgang mit Maschinencode-Programmteilen aus BASIC heraus stehen die folgenden hardwarenahen Anweisungen zur Verfügung. Dabei sollte man aber genau wissen, was man tut, denn damit kann man schnell sein System „lahmlegen“!

CLEAR

Mit dieser Anweisung kann die Speicherausnutzung geändert werden.

`CLEAR`	Ohne Parameter werden nur alle Variablen auf Null gesetzt. An Stringraum und RAM-Grenzen wird nichts geändert.
`CLEAR 500`	So wird nur der Stringraum gesetzt. Hier werden 500 Bytes für Strings reserviert (Standardwert = 50 Bytes). Das bisherige RAM-Ende bleibt unverändert. Zulässiger Wertebereich für Stringraumgröße: (0)...32767
`CLEAR 500,&8000`	Reserviert 500 Byte für Strings und setzt das obere RAM-Ende auf 8000h. Damit lässt sich Platz reservieren, um Maschinencode unterzubringen.

Die Angaben für den Stringraum und die RAM-Obergrenze können auch als numerische Variablen vorliegen.

PEEK/DEEK/POKE/DOKE

Mit diesen Anweisungen ist ein lesender bzw. schreibender Zugriff auf Speicherzellen möglich.

8 Bit:

A=PEEK(adr)	Der Wert an Speicheradresse (adr) wird in die Variable A geholt (0...255).
POKE adr,A	Der Wert der Variablen A (0...255) wird auf Speicheradresse adr geschrieben.

16 Bit:

B=DEEK(adr)	Der Wert an Adresse (adr+adr1) wird in die Variable B geholt.
DOKE adr,B	Der Wert der Variablen B wird auf Speicheradresse adr (niederwertiger Teil) und adr1 (höherwertiger Teil) geschrieben.

Die Adressen lassen sich als numerische Variablen, hexadezimal (Präfix &) oder dezimal angeben. Rückgabewerte >=8000h bei DEEK erscheinen als negative Zahlen!

INP/OUT/WAIT

Mit den Befehlen kann ein direkter Portzugriff erfolgen.

A=INP (4)	Es wird der aktuell an Port 4 (PIO1A: 04 = Tastatur) anliegende Wert abgefragt.
OUT 5,A	An Port 5 (PIO1B: 05 = Userport) wird mit der Wert der Variablen A ausgegeben.
WAIT 5,A[,B]	Der Port 5 wird gelesen, dieser Wert mit der Variablen B logisch UND-verknüpft und das Ergebnis nochmals mit Variable B logisch XOR-verknüpft. Das erfolgt solange (also „Warten“), bis das Resultat = Null ist.

CALL

Mit dem Befehl wird eine Maschinencode-Routine aufgerufen. In der Syntax gibt es wieder zwei Möglichkeiten für die Angabe der Adresse: dezimal oder hexadezimal. Um z.B. ein Programm auf Adresse 1900h aufzurufen: CALL 6400 oder CALL&1900

Eventuelle Parameter müssen an festzulegenden Adressen per POKE/DOKE übergeben bzw. per PEEK/DEEK zurück gelesen werden. Für die Arbeit mit Parametern siehe auch USR().

Der für das BASIC-Programm benötigte Code ist in einem freien Speicherbereich abzulegen. Einige Möglichkeiten:

- Ist der Umfang gering, so empfiehlt sich der Einfachheit wegen ein Mitspeichern als DATA-Zeilen im Programm und POKEn beim Programmstart.
- Der Code wird in einer REM-Zeile untergebracht (Bedingung: es darf keine 0 im Code vorkommen!), womit er auch zusammen mit dem Programm abgespeichert wird
- Entweder man benutzt den freien RAM-Bereich 1900h...1FFFh oder man setzt die obere RAM-Grenze mit CLEAR herab und nutzt diesen Bereich für MC. Die Ablage kann über im Programm abgespeicherte DATA-Zeilen (READ -> POKE) oder separates Laden mit BLOAD erfolgen.

USR()

Diese Funktion dient zum Ausführen von Maschinenprogrammen aus BASIC heraus. Dabei kann ein numerischer Parameter direkt übergeben werden, und die Funktion liefert (wenn gewünscht) auch einen berechneten Wert zurück.

Es handelt sich um eine Funktion und sie ist als solche aufzurufen. Die Syntax ist immer **`A=USR(B)`**, unabhängig davon, ob A weiter verarbeitet wird.

Vor dem Aufruf ist zunächst festzulegen, ab welcher Adresse das Maschinenprogramm ausgeführt werden soll. Dies erfolgt mit der Ablage dieser Adresse in den Speicherzellen **6001/6002h**. Es bietet sich dazu die DOKE-Anweisung an:

`DOKE&6001,&1900` setzt z. B. die Startadresse auf 1900h.

Innerhalb des MC-Programms können nun die gewünschten Operationen vorgenommen werden. Dafür gelten folgende Randbedingungen:

- Der übergebene Parameter muss im Wertebereich von -32768... 32767 liegen.
- Ein Sichern der Register ist nicht nötig, alle Register können verwendet werden.
- Die Übernahme des Parameters erfolgt zu Beginn des MC-Programms durch den Aufruf des BASIC-Systemcalls "**GETW**". Danach steht der Parameter im Registerpaar DE zur Weiterverarbeitung bereit.
- Der Rücksprung zu BASIC erfolgt mit **RET.**
- Soll aus dem MC-Programm ein Wert an BASIC zurückgegeben werden, ist dieses in den Registern A (höherwertiges Byte) und B (niederwertiges Byte) abzulegen und anschließend der BASIC-Systemcall mit "**JP ABPASS"** auszuführen. Ein extra RET ist dann nicht nötig.
- Rückgabewerte >=8000h erscheinen in der BASIC-Variablen als negative Werte,.d.h. 7FFFh =>32767, .8000h =>-32768,..., FFFFh => -1

Werden ggf. weitere Argumente benötigt, so müssen diese zwischen BASIC- und MC-Programm über freie RAM-Adressen (mit POKE/DOKE übergeben und mit PEEK/DEEK zurück lesen) ausgetauscht werden.

Mit der USR-Anweisung können problemlos mehrere verschiedene MC-Routinen aufgerufen werden. Es ist nur sicherzustellen, dass vor jedem Aufruf von USR() die Adresse der gewünschten MC-Routine in den Speicherzellen **6001h/6002** steht.

Beispiel:

```
100 DOKE &6001,&1900:A=USR(B):REM MC-ROUTINE AUF 1900H
110 DOKE &6001,&1A00:C=USR(D):REM MC-ROUTINE AUF 1A00H
```

Die für AC1-BASIC6 geltenden (festen) Adressen der Systemcalls für die Parameterübergabe ins/aus dem Unterprogramm lauten wie folgt:

GETW 2010h
ABPASS 2013h

Beispiel für USR():

Schnelles Füllen des Bildschirms mit zuvor definierten Zeichen. Rückgabe eines Testwerts „4097“.

```
100 CLS
110 PRINT"DEMO ZUR VERWENDUNG VON USR()"
120 PRINT"DER BS WIRD NACHEINANDER MIT ALLEN CODES GEFUELLT."
130 PRINT"START MIT BELIEBIGER TASTE..."
140 RESTORE230:FORI=0TO22:READC:POKE6400+I,C:NEXT:REM MC POKEN
150 PAUSE
160 DOKE&6001,&1900:REM STARTADRESSE MC SETZEN
170 F=1:REM STARTWERT CODEMUSTER
180 A=USR(F)
190 F=F+1:IF F<256THEN180
200 CLS
210 PRINT"RUECKGABEWERT=";A
220 REM MC-PROGRAMM
230 DATA205,16,32,123,33,0,16,119,17,1,16,1,255,7,237,176
240 DATA62,16,6,1,195,19,32
250 REM 1900h  CD1020 CALL GETW  ;WERTUEBERNAHME (F) => DE
260 REM 1903h  7B     LD A,E     ;D NICHT BENUTZT (F:MAX.FF)
270 REM 1904h  210010 LD HL,1000h;BS-ANFANG
280 REM 1907h  77     LD (HL),A
290 REM 1908h  110110 LD DE,1001h
300 REM 190Bh  01FF07 LD BC,07FFh
310 REM 190Eh  EDB0   LDIR
320 REM 1910H  3E10   LD A,10h   ;RUECKGABEWERT HIGH
330 REM 1912h  0601   LD B,01h   ;RUECKGABEWERT LOW
340 REM 1910h  C31320 JP ABPASS
```

BLOAD

Damit können Binärdateien mit der Endung BIN (Maschinencode oder auch anderer beliebiger Inhalt) auf eine bestimmte Adresse geladen werden. Syntax:

BLOAD *“name“,adresse*

Der Name kann als Literal oder Stringvariable angegeben werden. Die Adresse ist entweder dezimal, mit &-Präfix hexadezimal oder als Variable anzugeben.

Hier sollte man genau wissen, was man tut.
Es wird nicht auf das Überschreiben anderer Bereiche geprüft!

Anwendungsgebiete:

- um eine von BASIC zu benutzende MC-Routine an ihren Platz zu bringen
- Laden von Pseudo-Grafik direkt auf den Bildschirm
- Laden von Spielständen

Beispiel: Lädt das Uhrenstellprogramm auf Adresse 1900h und führt es aus

```
10 REM DEMO BLOAD
20 CLS: PRINT"LADEN UHREN-STELL-PROGRAMM"
30 BLOAD "SETC1900",&1900
40 PRINT"AUSFÜHREN..."
50 CALL &1900
60 PRINT"FERTIG!
```

BSAVE

Die BSAVE Anweisung sichert eine *anzahl* Bytes ab *adresse* als Binärdatei. Als Dateiendung wird automatisch „BIN“ verwendet. Syntax:

BSAVE *“name“,adresse,anzahl*

Der Name kann als Literal oder Stringvariable angegeben werden. Adresse und Anzahl der Bytes sind entweder dezimal, mit &-Präfix hexadezimal oder als Variable anzugeben.

Beispiel: sichert den aktuellen Bildschirminhalt

```
10 REM DEMO BSAVE
20 CLS: PRINT"BLA BLA BLA"
40 NAM$=“TEST“:ANZA=2048
50 BSAVE NAM$,&1000,ANZA
60 PRINT"FERTIG!
```

Hinweise:

- Beim Sichern eines Bildschirminhaltes wird der Farb-RAM nicht einbezogen. Dieser muss bei Bedarf separat gesichert werden. Dazu ist mit der OUT-Anweisung der Speicherbereich 1000h...17FFh auf den Farbspeicher umzuschalten und eine zweite BSAVE-Anweisung auszuführen. Für das Rückladen der beiden Dateien gilt Analoges.
- Beim Sichern des Bildschirminhaltes ist darauf zu achten, dass der Dateiname vorher noch nicht existiert, ansonsten wird die erscheinende „Überschreiben“-Rückfrage mit im Bild gespeichert!
- Im Gegensatz zu LOAD/SAVE können die Anweisungen BLOAD und BSAVE wie gezeigt in einem Programm verwendet werden. Tritt dabei kein Fehler auf, wird das Programm mit der nächsten Anweisung fortgesetzt.
- Mit BLOAD/BSAVE kann die fehlende Unterstützung „Laden/Speichern von Feldern“ nachgebildet werden, siehe nachfolgendes Beispiel.

VARPTR

Mit dieser Funktion können die aktuellen Speicheradressen von numerischen oder Stringvariablen bestimmt werden.

Syntax: **VARPTR(*variablenname*)**

Dabei ist zu beachten, dass Speicheradressen >=8000h wie bei DEEK() oder USR() als negative Werte zurückgegeben werden. Umrechnung also ggf. so:

```
AD=VARPTR(A$):IFAD<0THENAD=65536+AD
```

Für die Anwendung dieser Funktion ist u.a. die Kenntnis der Variablenablage erforderlich. Die ermittelte Adresse weist immer auf das erste Byte der folgenden Strukturen:

numerisch: 48 Bit-Gleitkommazahl		**String, max. 255 Byte lang**	
Byte 1	Mantisse LSB	Byte 1	1. Zeichen des Strings
Byte 2	Mantisse	Byte 2	2. Zeichen
Byte 3	Mantisse	Byte 3	3. Zeichen
Byte 4	Mantisse	...	....
Byte 5	Mantisse MSB, Bit 7=Vorzeichen	...	...
Byte 6	Exponent		nicht mit 00-terminiert!

Beispielanwendung Felder Sichern/Laden: „BESTENLISTE“ eines Spiels

Gegeben sei:
- Spielername in NM$(I), zugehörige Punktezahl in PT(I)
- 14 Byte für Name, 2 Byte für Punkte
- Nutzung Speicher ab 1900h als Zwischenablage

Sichern:

```
A$=NM$(I)                          :REM FELDWERT IN PUFFERVARIABLE HOLEN
AD=VARPTR(A$):IFAD<0THENAD=65536+AD:REM ZEIGER DARAUF
FOR K=0 TO LEN(A$)-1               :REM ALLE ZEICHEN DES NAMENS
 POKE&1900+I*16+K,PEEK(AD+K)       :REM UMKOPIEREN AUS PUFFER A$ => ABLAGE
NEXT
DOKE&1900+I*16+14,PT(I)            :REM PUNKTE ABLEGEN
...                                :REM WEITERE EINTRÄGE ANALOG
BSAVE"BESTE",&1900,160             :REM ABLAGE KOMPLETT ABSPEICHERN
```

Laden:

```
BLOAD"BESTE",&1900                 :REM LADEN KOMPLETT IN ABLAGE
A$=SPACE$(16)                      :REM PUFFERVARIABLE SCHAFFEN
D=VARPTR(A$):IFAD<0THENAD=65536+AD :REM ZEIGER DARAUF
FOR K=0 TO 13                      :REM ALLE ZEICHEN DES NAMENS
 POKE AD+K,PEEK(&1900+I*16+K)      :REM UMKOPIEREN ABLAGE => PUFFER
NEXT
NM$(I)=A$:PT(I)=DEEK(&1900+16*I+14):REM FELD MIT PUFFER BESTÜCKEN
...                                :REM WEITERE EINTRÄGE ANALOG
```

Anmerkungen:

- Die Methode erscheint umständlich, es kann jedoch das in *AC1-BASIC6* fehlende Laden und Sichern von Feldern damit sehr flexibel nachgebildet werden.
- Die Verwendung von VARPTR und das Umkopieren per PEEK/POKE ist schneller als eine entsprechende Stringverarbeitung.

Bedienungshinweise

Im Gegensatz zu GS-BASIC lässt sich der Kursor immer nur in einer Kommandozeile (bzw. der Editorzeile) bewegen. Fehlerhafte Eingaben können nur mit der Backspace- oder Kursor-links-Taste korrigiert werden, womit das letzte eingegebene Zeichen gelöscht wird.

Allgemeine Steuertasten

Taste	**Wirkung**
Kursor links oder Backspace	löschen letztes Eingabezeichen
^C	Abbruch Programm, LIST, INPUT- oder Zeilen-Eingabe
^Q	Fortsetzen angehaltenes Programm oder LIST
^R	Anzeige der momentanen Zeilennummer bei laufendem Programm
^S	Anhalten laufendes BASIC-Programm oder LIST

Funktionstasten

Wie auch GS-BASIC besitzt *AC1-BASIC6* die Möglichkeit der Definition von acht Funktionstasten. Standardmäßig werden folgende Kommandos gesetzt:

KEY1	14h	Strg+T	LIST	sofortige Kommandoausführung
KEY2	15h	Strg+U	RUN	sofortige Kommandoausführung
KEY3	16h	Strg+V	LOAD“	noch Dateinamen ergänzen+ ENTER
KEY4	17h	Strg+W	SAVE“	noch Dateinamen ergänzen+ ENTER
KEY5	18h	Strg+X	DIR	sofortige Kommandoausführung
KEY6...KEY8	19h...1Bh	Strg+Y...Strg+Ä	-	unbelegt

Die Kommandos werden beim Kaltstart von *AC1-BASIC6* im RAM abgelegt und lassen sich z.B. mit dem Programm „Funktionstasten“ oder „F1-F8-KEY“ (Bedienung beachten!) neu definieren. Die aktuelle Belegung kann mit der Anweisung **KEY** angezeigt werden.

Anmerkung zur PS/2-Tastatur:
Durch entsprechende Programmierung lassen sich Strg+T...Strg+Ä auf die Funktionstasten legen.

Grafiktaste

Ist am AC1 eine Grafiktaste installiert, so kann diese auch unter *AC1-BASIC6* benutzt werden. Der Umgang unterscheidet sich jedoch etwas von GS-BASIC:

- Bei gedrückter Grafiktaste wird jedes Tastaturzeichen (auch ENTER!) an beliebiger Zeichenposition als Grafikzeichen gesetzt (kein „Gänsefüßchen“-Modus).
- Grafikzeichen in REM-Zeilen sollte man vermeiden, da diese bei LIST in entsprechende Token übersetzt werden.

ESC-, NMI-Taste

- ESC liefert bei Abfrage mit INKEY$ den Code 27 = 1Bh. Eine gesonderte Funktion ist damit nicht verbunden.
- NMI bricht „ausweglose“ oder in MODE1 laufende Programme ab und übergibt die Steuerung an die Kommandoeingabe (bleibt im BASIC!).

Quelltextbearbeitung

Generell reagiert „AC1-BASIC6 USB“ bei der Quelltexterstellung wie die meisten Interpreter:

- Eingaben in einer Kommandozeile, die mit einer Zahl beginnen, werden als Zeilennummer interpretiert. Das nächste nichtnumerische Zeichen ist das erste Zeichen des Zeileninhalts.
- Wird nur eine Zahl gefolgt von Enter eingegeben, so wird diese Zeile ohne Meldung/ Warnung gelöscht (falls vorhanden).
- Die von GS-Basic her bekannte Möglichkeit, Steuerzeichen direkt in eine PRINT-Anweisung einzugeben („im Gänsefüßchenmodus“) gibt es unter *AC1-BASIC6* nicht.

AUTO

Automatische Zeilennummererzeugung
AUTO ab Zeile 10, 10er Schritte
AUTO 100 ab Zeile 100, 10er Schritte
AUTO 5,1 ab Zeile 5, 1er Schritte
ENTER ohne Eingabe einer Zeile oder ^C brechen den AUTO-Modus ab.

COPY

Vorhandene Zeilen können damit unter neuer Zeilennummer dupliziert werden.
COPY 100=10 dupliziert Zeile 10 als neue Nummer 100
COPY 100=50-70 dupliziert Zeile 50-70 als neue Nummer 100 in 10er Schritten
COPY 100,1=50-70 dupliziert Zeile 50-70 als neue Nummer 100 in 1er Schritten

DELETE

Es wird die angegebene Zeile oder der Zeilenbereich gelöscht.
DELETE 10
DELETE 10-90

EDIT zeilennummer

Der ursprüngliche Editor (vegleichbar mit ED.COM des CP/M) ist für „Gelegenheitsanwender“ kaum praktikabel. Aus diesem Grunde wurde er durch einen einfachen „WYSIWYG“-Zeileneditor ersetzt. Er bringt die gesamte Zeile sichtbar zur Bearbeitung auf den Schirm.

Taste	Wirkung
Kursortasten	navigieren vor/zurück in der Zeile (nach Zeilennummer bis letztes Zeichen)
^D = Entf	löschen eines Zeichens am Kursor (Rest rückt nach)
^E =Einfg	einfügen eines (Leer-)Zeichens am Kursor, Rest rückt auf
^Z	geht hinter das aktuelle Zeilenende, ab dort kann nun die Zeile fortlaufend weiter ergänzt werden
^C	Abbruch der Bearbeitung, Zeile bleibt unverändert
<ENTER>	übernimmt Änderung
übrige Tasten>=20h	überschreiben alter Inhalt, Kursor rückt weiter

LIST

listet ein Programm auf den Bildschirm (LIST) bzw. auf den Drucker (**LLIST**).
LIST listet alle Zeilen komplett
LIST 100 listet nur Zeile 100
LIST 10-100 listet die Zeilen 10 bis 100
LIST 10- listet alle Zeilen ab 10 bis zum Ende
LIST -100 listet alle Zeilen bis 100
Überschreitet die Zeilenanzahl die mit **LINES n** eingestellte Anzahl (Standard 20), so hält das Listing an. Mit beliebiger Taste wird fortgesetzt, ^C bricht ab.

NEW
Das aktuell im Speicher befindliche BASIC-Programm wird gelöscht. Das empfielt sich immer vor dem Laden eines neuen Programms, ansonsten wird ggf. ein nicht beabsichtigtes MERGE ausgeführt (siehe auch unter LOAD).

RENUMBER	
RENUMBER	Nummeriert das Programm neu komplett in 10er Schritten
RENUMBER 5	beginnend ab 5 in 10er Schritten
RENUMBER 5,5	beginnend ab 5 in 5er Schritten
RENUMBER 5,1,100	beginnend ab100 in 1er Schritten

Ausgabeformatierung

Die bekannte PRINT-Anweisung bringt Strings und numerische Werte auf den Schirm bzw. auf den Drucker (LPRINT).

- Ein Komma teilt die Ausgabezeile in 14 Spalten große Zonen auf, in denen numerisch Werte rechtsbündig und Texte linksbündig angezeigt oder gedruckt werden.
- Ein Semikolon trennt nur die Elemente der Liste und nimmt keine Positionierung vor.
- Neben Komma und Semikolon lassen sich die Befehle TAB und SPC(zusätzlich zum Positionieren nutzen.

Die neue Erweiterung **USING** ... ermöglicht eine flexible Formatierung der Ausgabe.

PRINT USING	
PRINT USING <zeilennummer>; < ausgabeliste > **PRINT USING X$; < ausgabeliste >** Es werden Zahlen oder Zeichenketten in einem angegebenen Format gedruckt. Ein %-Zeichen wird vor die Ausgabe gestellt, wenn die zu druckende Zahl größer ist als das angegebene Format. Das Format wird in einer Zeile (und dort durch ! gekennzeichnet) oder in einem String spezifiziert. Zur Formatsteuerung benutzt man folgende Zeichen:	
#	Platzhalter für eine Ziffer
+	Platzhalter für das Vorzeichen der Zahl. Das Vorzeichen wird immer ausgegeben, auch wenn es positiv ist.
-	wie +, steht es aber am Ende, werden positive Zahlen ohne Vorzeichen gedruckt
**	leere (Vorkomma-)Positionen werden mit Nullen gefüllt, Platzhalter für 2 Ziffern
$$	Ein $ wird unmittelbar vor die erste Zahl gedruckt
**$	Kombination aus ** und $$
,	an alle 3 Stellen wird ein Komma eingefügt (muss links vom Dezimalpunkt stehen).
^^^^	Zahlen werden im Exponentialformat ausgegeben
'	leitet ein Stringformat ein. Das Format wird durch die folgenden Zeichen festgelegt: L linksbündig R rechtsbündig C zentriert E Linksangleich mit Erweiterung, falls der String zu lang ist.

Beispiele:

```
PRINT USING "##.##";PI                    druckt 3.14
A$ = "+#.##^^^^"
PRINT USING A$;PI                         druckt +3.14E+0
10 !'RRRRRRRRR      :REM 10 Zeichen breit, Strings rechtsbündig drucken
20 PRINT USING 10;"XX"                    druckt „        XX“
30 PRINT USING 10;"X"                     druckt „         X“
40 PRINT USING "**##:**#:**";A;A;A        druckt 0000:000:00 (bei A=0)
```

WIDTH
stellt die Zeilenbreite am Bildschirm (bzw. Drucker bei **LWIDTH**) **WIDTH 20** Nach der Ausgabe von 20 Zeichen wird automatisch ein CR/LF eingefügt. **WIDTH 64,** Ein nachgestelltes Komma unterdrückt die automatische CR/LF-Ausgabe. Die Angabe gilt bis zum nächsten Kaltstart.

Genauigkeit

Der mögliche Zahlenbereich der 48-Bit-Fließkommaarithmetik reicht von ca. 2,9·10E-39 bis 1,7·10E+38. Werden diese Grenzen bei Rechenoperationen überschritten, erfolgt eine „ARITHMETIC OVERFLOW“-Fehlermeldung.

Die maximal ausgebbare Genauigkeit beträgt 11 Stellen, intern ist sie höher. Signifikant sind aber maximal 12 Stellen. Normalerweise wird als „klassische“ Dezimalzahl ausgegeben. Die Exponentialdarstellung wird erst dann verwendet, wenn die Ausgabe 11 Stellen überschreiten würde:

```
?99999999999+1
 1E+11
```

PRECISION
Stellt die Ausgabegenauigkeit auf x Stellen ein, beeinflusst auch die STR$-Darstellung. **PRECISION 6**

Die PRECISION-Anweisung beeinflusst nur die Darstellung auf dem Bildschirm. Intern rechnet *AC1-BASIC6* immer in der maximalen Genauigkeit. Die Ausgabegenauigkeit kann weiterhin mit den folgenden Befehlen beeinflusst werden (x steht für einen numerischen Wert):

INT(x)	gibt nur den ganzzahligen Anzeil zurück, es wird nach den Standardregeln gerundet
FIX(x)	gibt nur den ganzzahligen Anzeil zurück, es wird nicht gerundet
FRAC(x)	entfernt ganzzahligen Anteil und gibt nur die Nachkommastellen zurück

Obwohl dieser Interpreter intern mit 48 Bit rechnet (GS-Basic: 32 Bit) können nicht alle Zahlen exakt dargestellt werden. Das zeigt sich speziell bei Dezimalbrüchen. An folgenden Beispielen werden die Grenzen sowie Vor- und Nachteile deutlich:

Eingabe	**angezeigtes Rechenergebnis/Bemerkung**
`X=SQR(2)` `? X^2`	angezeigt wird „2“. Sollte selbstverständlich sein, aber einfache Taschenrechner können dies nicht!
`? 5.0001 * 10^4`	50000.999999 Hier zeigt sich ein Problem der "höheren Genauigkeit" Unter GS-Basic wird glatt „50001“ angezeigt!
`? LOG(0.99995)`	-5.0001249077E-05 GS-BASIC ist „schlechter“: -4.97637E-05

`10 FOR I= -1 TO 1 STEP .1`
`20 PRINT I`
`30 IF I=0 THEN PRINT "0!"`
`40 NEXT`
	Infolge endlicher Genauigkeit der Abbildung von Dezimalbrüchen wird hier der Wert Null nie erreicht, kleinster Wert: 1.1368683772E-12
	Wenn möglich und sinnvoll, sollte in solchen Fällen als Schrittweite ein „dualer“ Wert angegeben werden, um das Problem zu umgehen.
	Mit STEP 0.125/0,0625/... wird die Null exakt durchlaufen!

Fehlerbehandlung

AC1-BASIC6 bietet im Gegensatz zu vielen anderen Interpretern eine umfangreiche Fehlerbehandlung.

- Fehlermeldungen werden im Gegensatz zu GS-BASIC als Klartext ausgegeben.
- Tritt in einer Zeile ein Syntax-Error auf, so wird der Programmablauf wie üblich unterbrochen. Es wird jedoch nun automatisch der Zeileneditor mit der fehlerhaften Zeile aufgerufen. Die entsprechende Stelle in der Zeile muss man selber finden :-)
- Andere Fehler wie Division durch Null können während der Laufzeit „behandelt“ werden. Dazu stehen folgende Anweisungen und Funktionen zur Verfügung:

ON ERROR GOTO zeile	Einleitung der Fehlerbehandlung (am Programmanfang!)
ERR	reservierte Variable, enthält den Fehlercode
ERL	reservierte Variable, gibt die Zeilennummer zurück, in welcher der Fehler aufgetreten ist
ERROR n	simuliert das Auftreten eines Fehlers mit der Nummer n
RESUME **RESUME NEXT** **RESUME zeile**	Fortsetzung mit der fehlerverursachenden Anweisung Fortsetzung nach der fehlerverursachenden Anweisung Fortsetzung ab bestimmter Zeilennummer

Damit können Laufzeitfehler abgefangen und Programmabbrüche durch geeignete Behandlung vermieden werden.

Beispiel (teilweise entnommen aus [1]):

```
10 ON ERROR GOTO 100:REM BEI AUFTRETEN EINES FEHLERS ZU ZEILE 100 GEHEN
20 INPUT „ZWEI ZAHLEN EINGEBEN:“;A,B
30 C=A/B
40 PRINT C
50 GOTO 10
100 REM FEHLERBEHANDLUNGSPROGRAMM
110 IF ERL=30 THEN PRINT „DIE VARIABLE B WAR 0“
120 RESUME 10:REM FORTSETZUNG MIT NEUEINGABE
```

Geschwindigkeit

Die Anweisung MODE dient darüber hinaus (wie auch bei GS-BASIC) zur Umschaltung der Unterbrechungsmöglichkeit mit ^C:

MODE 1	„Break off“
MODE 2	„Break on “ (Grundeinstellung bei Start)

Im Gegensatz zu GS-BASIC, wo dies gleichzeitig mit einem FAST/SLOW-Modus verknüpft ist, bedeutet dies hier kaum eine Änderung der Verarbeitungsgeschwindigkeit:

Benchmarktest	**Ausgaben**
100 REM BENCHMARK 7 105 MODE2:REM MODE1 120 PRINT "S" 130 K=0 140 DIM MQ(5) 150 K=K+1 160 A=K/2*3+4-5 170 GOSUB250 180 FORL=1TO5 190 MQ(L)=A 200 NEXT L 210 IF K<1000THEN150 220 PRINT"E" 240 END 250 RETURN	100 **REM 1000 STERNE** 105 MODE2:REM MODE1 120 CLS 130 FORI=1TO10000 140 PRINT"*"; 150 NEXT
	Laufzeit MODE/MODE2: ca. 45 sek. **GSB:** **MODE1: ca. 28 sec.** **MODE2: ca. 30 sec.**
Laufzeit MODE1/MODE2: ca. 50 sek. **GSB:** **MODE1: ca. 36 sec.** **MODE2: ca. 60 sec.**	10 **REM 8182 x SET** 20 MODE2:REM MODE1 30 CLS 40 FOR Y=0TO63 50 FOR X=0TO127 60 SET X,Y 70 NEXT:NEXT
	Laufzeit MODE1/MODE2: ca. 45 sek. **GSB:** **MODE1: 38 sek.** **MODE2: 48 sek.**

Es lohnt sich also in *AC1-BASIC6* nicht unbedingt, Mode1 verwenden zu wollen. Man sperrt sich nur aus :-)

***

Allgemein muss festgestellt werden, dass hinsichtlich der Verabeitungsgeschwindigkeit dieser Interpreter im Vergleich zu anderen etwas schlechter abschneidet. Hauptgründe sind die 48-Bit-Fließkommaarithmetik sowie die Farb- und Fensterverwaltung. Speziell die trigonometrischen Funktionen benötigen relativ viel Zeit. Wer also „Action“-Spiele programmieren will, sollte Maschinencode einsetzen:-)

Anlage

Wichtige Systemadressen

Monitor-/System-Bezüge:

#04	Tastatur-Port	**#1000**	Bildwiederholspeicher Farb-RAM
#05	USER-Port	**#17FF**	
#F0	Farbregister	**#1800**	BS-Adresse des Kursors
#FC	USB-Port	**#1805**	Zwischensprung für RST 10H
#0008	UP Zeicheneingabe	**#181F**	Farb-Reset-Byte 00=keine Farbkarte
#0010	UP Zeichenausgabe	**#1820**	V.24-Steuerbyte
#0018	UP Zeichenkettenausgabe	**#1821**	E/A-Byte
#006E	Einsprung nach BYE	**#1822**	Code der zuletzt gedrückten Taste
#0272	UP „Klingelsignal“	**#18D6**	Zwischenablage Dateilänge/Restzeit
#0287	UP „Ton“ Variabel	**#18DA**	Zwischenablage Speicheradresse Load
#0EB4	Joystick-Routine	**#18DC**	Zwischenablage Dateiname (8.3)
#07EB	Zeitschleife MS30	**#18E9**	RTC-Puffer (7 Byte)
#07FD	Rücksprung zum Monitor	**#1F80**	F-Tastenpuffer

Interpreter-(Eprom-)Bereich:

#2000	#C3 ...	Kaltstart
#2003	#C3 ...	Warmstart
#2006	#C3 ...	Zeichen in C zum Drucker, Standard: eingebauter V24-Treiber
#2009	#C3 #EB #07	Sprung zur Druckerinitialisierung, Standardwert: „MS 30“
#200C	#C3 #6E #00	Rücksprung zum Monitor, Standardwert „BASIC-EPROM aus“
#2010	#C3 ...	Sprung GETW: holt USR-Argument in MC-Routine
#2013	#C3 ...	Sprung ABPASS: gibt Wert an USR-Funktion zurück
#2016	#BF #FF	Höchste zu benutzende RAM-Zelle, Standardwert: #FFBF
#201B	#14 #00	Zeilenanzahl für LIST, Standardwert: 20
#201D	#0B	Formfaktor Kreis, Standardwert: 11 ggf. hier zu ändern, wenn ein Kreis auf verwendetem Bildwiedergabe-gerät nicht „rund“ erscheint

Gelb markierte Parameter können ggf. an eigene Bedürfnisse anpasst (gepatcht) werden.

Einige BASIC-Arbeitszellen:

#6001	xx xx	USR-Adresse (mit DOKE zu beschreiben vor USR-Aufruf)
#6003	xx xx	aktuell mit LINES n eingestellte Zeilenanzahl
#6005	xx	aktuell wirksamer Formfaktor für Kreis
#6006	xx	aktuell mit COLOR v,h eingestelltes Farbbyte
#6007	xx	aktuell mit CLS h eingestellte Hintergrundfarbe
#60A6	xx xx	Zeiger auf Beginn BASIC-Programm
#60A8	xx xx	Zeiger auf Ende BASIC-Programm (Adresse Byte nach der 3. Null) =Beginn Variablenbereich
#60AA	xx xx	Zeiger auf Beginn der Felder
#60AC	xx xx	Zeiger auf Ende der Felder
#6300	xx xx	Beginn BASIC-Programm

F-Tastenbelegung:

```
1F80h DEFM  "LIST",0Dh,0      ;Strg+T
      DEFM  "RUN",0Dh,0       ;Strg+U
      DEFM  "CLOAD",22h,0     ;Strg+V
      DEFM  "CSAVE",22h,0     ;Strg+W
      DEFM  "DIR",0Dh,0       ;Strg+X
      DEFM  0,0,0             ;Strg+Y, Z und Ä nicht belegt
      DEFB  0                 ;abschließende 0 => Ende F-Tasten-Abfolge

;0    = Ende der Tastenfolge
;0Dh  = Kommando-Ausführung sofort
```

Tokenübersicht

	80	90	A0	B0	C0	D0	E0	F0
x0	END	REM	BSAVE	NEW	UNTIL	TAB(	XOR	
x1	FOR	STOP	RANDOMIZE	AUTO	ERROR	TO	>	
x2	NEXT	OUT	LINES	COPY	RESUME	FN	=	
x3	DATA	ON	LWIDTH	DIR	PAUSE	SPC(	<	
x4	COLOR	KEY	LNULL	MODE	DOKE	THEN		
x5	BLOAD	WAIT	WIDTH	SOUND	CLS	NOT		
x6	INPUT	DEF	LVAR	LIST	CURSOR	STEP		
x7	DIM	POKE	WINDOW	LLIST	DRAW	+		
x8	READ	PRINT	'	RENUMBER	CIRCLE	-		
x9	LET	CLEAR	PRECISION	DELETE	SET	*		
xA	GOTO	FNRETURN	CALL	EDIT	RESET	/		
xB	FNEND	SAVE	ERASE	DEG	CD	DIV		
xC	IF	!	SWAP	RAD	BYE	MOD		
xD	RESTORE	ELSE	LINE	WHILE	CONT	^		
xE	GOSUB	LPRINT	RUN	WEND	USING	AND		
xF	RETURN	TRACE	LOAD	REPEAT	PI	OR		Tabelle2

	80	90	A0	B0	C0	D0	E0	F0
x0	SGN	TAN	MID$					
x1	INT	ATN	INKEY$					
x2	FIX	PEEK	STRING$					
x3	ABS	FRAC	ERR					
x4	USR	LGT	ERL					
x5	FRE	SQU	POINT					
x6	INP	BIN$	INSTR					
x7	POS	HEX$	TIME$					
x8	LPOS	LEN	JOY					
x9	GETCL	STR$	DEEK					
xA	SQR	VAL	VARPTR					
xB	RND	ASC						
xC	LOG	SPACE$						
xD	EXP	CHR$						
xE	COS	LEFT$						
xF	SIN	RIGHT$						

Tabelle 2: Funktionen, alle mit Vorbyte FF

Alphabetische Übersicht der Anweisungen, Funktionen und Operatoren

Ausdruck	Bedeutung	Syntaxbeispiel
!	Kommentar (ganze Zeile) Formatzeile für PRINT USING	`! VERFASSER:` `!'LLLLL`
'	Zwischenkommentar, nur bis nächste Anweisung gültig	`'TEST:A=1`
^	Operator: Potenzierung	`A=2^B`
-	Operator: Subtraktion	`A=B-C`
*****	Operator: Multiplikation	`A=B*C`
/	Operator: Division	`A=B/C`
&	Bezeichner für Hexadezimal-/Binärliterale	`A=&1900` `B=&&11001100`
+	Operator: Addition	`A=B+C`
<	Operator: Bedingung „kleiner als“	`A<B`
=	Operator: Bedingung „gleich“ Wertzuweisung	`IF (A=B)` `C=10`
>	Operator: Bedingung „größer als“	`A>B`
ABS	Funktion: Betrag	`A=ABS(B)`
AND	Operator: log. UND	`A = B AND C`
ASC	Funktion: Code des ersten Zeichens	`A=ASC(A$)`
ATN	Funktion: Arcustangens	`PRINT 4*ATN(1)`
AUTO	Zeilen automatisch nummerieren	`AUTO 10,5`
BIN$	Funktion: erzeugt Zeichenkette als Binärausdruck	`A$=BIN$(15)`
BLOAD	Laden Binärdatei an bestimmte Adresse	`BLOAD „TEST“,&1900`
BSAVE	Sichern Binärdatei ab bestimmte Adresse bestimmte Menge	`BSAVE „TEST“,&1900,&100`
BYE	Verlassen von BASIC	`BYE`
CALL	MC-Programm aufrufen	`CALL &1900`
CD	USB-Verzeichniswechsel	`CD „BASIC“`
CHR$	Funktion: wandelt num. Wert in Textzeichen	`A$=CHR$(44)`
CIRCLE	Zeichnet Kreis (oder Ellipse mit zusätzlichem 4. Parameter)	`CIRCLE 63,31,31`
CLEAR	Variablen löschen, Stringraum setzen, RAM-Grenze setzen	`CLEAR 500,&8000`
CLS	Bildschirm löschen für Farb-BWS wahlweise mit Löschfarbe als Parameter	`CLS` `CLS 5`

COLOR	für Farb-BWS: Einstellen Vorder- und Hintergrundfarbe
	nur 1 Parameter: nur Vordergrund ändern
	ohne Parameter: auf RESET-Farben zurückstellen
		`COLOR 1,7`
		`COLOR 1`
		`COLOR`

CONT	Programm fortsetzen	`CONT`
COPY	Zeilen kopieren	`COPY 20=10`
COS	Funktion: Kosinus berechnen	`X=COS(0.5)`
CURSOR	Cursorpositionierung Spalte, Zeile	`CURSOR 15,10`
DATA	Datenelement definieren	`DATA 10,“HAUS“`
DEEK	Funktion: Doppelspeicherplatz auslesen	`A=DEEK(&1900)`
DEF	Funktion definieren	`DEF FN NEU(X)`

DEG	Umschalten auf Altgrad bei trig. Funktionen	`DEG`
DELETE	Zeile(n) löschen	`DELETE 10-50`
DIM	Array dimensionieren	`DIM A$(100)`
DIR	USB: Verzeichnisinhalt anzeigen	`DIR`
DIV	Operator: Division zweier ganzer Zahlen	`A=B DIV C`
DOKE	Doppelspeicherplatz beschreiben	`DOKE &1900,&1000`
DRAW **DRAWTO**	Schnelles Linienzeichnen, merkt sich Endpunkt DRAWTO: vom letzten Punkt aus	`DRAW 0,0,127,63` `DRAWTO 10,10`
EDIT	Zeileneditor aufrufen	`EDIT 10`
ELSE	Klausel für IF...THEN	`IF A=1 THEN B=0 ELSE C=1`
END	Programmende	`END`
ERASE	Löschen Variable oder Array	`ERASE A$`
ERL	reservierte Variable: liefert Zeilennummer, wo Fehler	`A=ERL`
ERR	reservierte Variable: liefert Fehlercode	`A=ERR`
ERROR	simuliert Fehler	`ERROR 33`
EXP	Funktion: Exponential	`A=EXP(1)`
FIX	Funktion: ganzer Anteil (ohne Rundung)	`A=FIX(3.14)`
FN	Bezeichner für benutzerdefinierte Funktion	`FN XXXX(0)`
FNEND	Ende benutzerdefinierte Funktion	`FNEND I`
FNRETURN	vorzeitiger Ausstieg aus benutzerdefinierter Funktion	`FNRETURN N`
FOR	Schleifenkopf	`FOR I=1...`
FRAC	entfernt ganzen Teil und gibt Bruchteil zurück	`A=FRAC(3.14)`
FRE	Funktion: freier Speicherplatz (Programmbytes/Stringraum)	`A=FRE(0)` `B=FRE(„X“)`
GETCL	Funktion: ermittelt Farbbyte an BS-Adersse	`A=GETCL(&1000)`
GOSUB	Aufruf Unterprogramm	`GOSUB 100`
GOTO	Unbedingter Sprung	`GOTO 100`
HEX$	Funktion: erzeugt Zeichenkette als HEX-Ausdruck	`A$=HEX$(10)`
IF	Bedingung folgt	`IF A=B`
INKEY$	Funktion: Tastaturabfrage ohne warten	`A$=INKEY$`
INP	Funktion: Port abfragen	`A=INP(4)`
INPUT	Eingabe per Tastatur	`INPUT A, A$` `INPUT „Wert=“;A`
INSTR	Funktion: Teilstringsuche (Bestimmen Position)	`A=INSTR(“HAUS“,“U“)`
INT	Funktion: ganzer Anteil (mit Runden)	`A=INT(3.14)`
JOY	Funktion: Joystickabfrage	`A=JOY(0)`
KEY	Anzeige der F-Tastenbelegung	`KEY`
LEFT$	Funktion: holt linken Teil eines Strings	`A$=LEFT$(B$,4)`
LEN	Funktion: ermittelt die Länge eines Strings	`A=LEN(A$)`
LET	Zuweisung (kann entfallen)	`LET A=1`
LGT	Funktion: dekadischer Logarithmus (Basis 10)	`A=LGT(4)`

LINE	Vorsatz für INPUT (um Sonderzeichen eingeben zu können)	`LINE INPUT „TEXT:“;A$`
LINES	Anzahl der Zeilen bei LIST einstellen	`LINES 20`
LIST	Programm auflisten	`LIST 10-50`
LLIST	Programmlisting an Drucker ausgeben	`LLIST`
LNULL	Anzahl Nullzeichen für Drucker setzen	`LNULL 10,32`
LOAD	Laden Basicprogramm per USB	`LOAD „TEST“`
LOG	Funktion: natürlicher Logarithmus (Basis e)	`A=LOG(10)`
LPOS	Funktion: akt. Spaltenposition Drucker	`A=LPOS(0)`
LPRINT	Zeichenausgabe an Drucker	`LPPRINT „HALLO“`
LVAR	Variableninhalte anzeigen	`LVAR`
LWIDTH	Druckbreite einstellen	`LWIDTH 50`
MID$	Funktion: Teilstring entnehmen	`A$=MID$(B$,2,4)`
MOD	Operator: Rest bei Division bestimmen	`A= 10 MOD 3`

MODE	Zeichensatz umschalten,
	^C-Unterbrechungsmöglichkeit aus-/ einschalten
		`MODE`
		`MODE 0`
		`MODE 1`
		`MODE 2`

NEW	vorhandenes Programm im Speicher löschen	`NEW`
NEXT	Schleifenende	`... NEXT`
NOT	Operator: logisches „Nicht“	`A= NOT B`
ON	Sprungverteiler für GOTO und GOSUB	`ON A GOTO...`
OR	Operator: logisches „Oder“	`A= B OR C`
OUT	Portausgabe	`OUT 5,100`
PAUSE	Anhalten für n*0,1 sec	`PAUSE 10`
PEEK	Speicherplatz lesen	`A=PEEK(&1900)`
PI	reservierte Variable: 3.1415...	`A=PI`
POINT	Funktion: Blockgrafikpixel testen	`A=POINT(10,10)`
POKE	Speicheradresse beschreiben, für Farb-BWS: auch aktuelles Farbbyte wird automatisch mit gepokt	`POKE &1900,0`
POS	Funktion: akt. Spaltenposition Bildschirm	`A=POS(0)`
PRECISION	Anzeigegenauigkeit (Stellenzahl) einstellen	`PRECISION 4`
PRINT	Ausdruck anzeigen	`PRINT „HALLO“`
RAD	Umschalten auf Radiant bei trig. Funktionen	`RAD`
RANDOMIZE	Zufallsgenerator neu initialisieren	`RANDOMIZE`
READ	Datenelement einlesen	`READ A$`
REM	Kommentar	`REM VERFASSER...`
RENUMBER	Programm neu nummerieren	`RENUMBER 10,5,10`
REPEAT	Schleifenkopf	`REPEAT`
RESET	Blockgrafik-Pixel rücksetzen merkt sich Koordiante für DRAWTO	`RESET 10,10`
RESTORE	Datenelement-Zeiger setzen	`RESTORE 100`
RESUME	Fortsetzen nach Fehlerbehandlung	`RESUME`

RETURN	Rückkehr aus Unterprogramm	`RETURN`
RIGHT$	Funktion: rechten Teilstring entnehmen	`A$=RIGHT$(B$,4)`
RND	Funktion: Zufallszahl 0...<1 erzeugen	`A=RND(0)`

RUN	Programm starten
	Starten ab Zeilennummer
	Programm per USB laden + starten
		`RUN`
		`RUN 10`
		`RUN „name“`

SAVE	Programm per USB sichern	`SAVE „name“`
SET	Blockgrafik-Pixel setzen, merkt sich Koordinate für DRAWTO	`SET 10,10`
SGN	Funktion: Vorzeichen bestimmen	`A=SGN(B)`
SIN	Funktion: Sinus	`A=SIN(0.5)`
SOUND	Tonsignal ausgeben, nur `SOUND = „Bell“`	`SOUND 100,150`
SPACE$	Funktion: erzeugt Zeichenkette mit n Leerzeichen	`A$=SPACE$(10)`
SPC(	Leerzeichen ausgeben	`PRINT SPC(20)`
SQR	Funktion: Quadratwurzel	`A=SQR(2)`
SQU	Funktion: berechnet Quadrat	`A=SQU(2)`
STEP	Schrittweite der FOR-Schleife	`STEP 5`
STOP	Programm anhalten	`STOP`
STR$	Funktion: wandelt numer. Ausdrucks in String	`A$=STR$(3.14)`
STRING$	Funktion: n Zeichen als String ausgeben	`A$=STRING$(5,“*“)`
SWAP	Variableninhalte vertauschen	`SWAP A,B`
TAB(	Ausgabe Tabulator	`PRINT TAB(5)`
TAN	Funktion: Tangens	`A=TAN(0.5)`
THEN	bestimmt Anweisung nach erfüllter Bedingung	`IF A=B THEN C=1`

TIME$	Funktion: RTC auslesen:
	Datum, Zeit, Wochentag
		`PRINT TIME$(1)`
		`PRINT TIME$(2)`
		`PRINT TIME$(3)`

TO	bestimmt Schleifenendwert	`FOR I=1 TO 10`
TRACE	Protokollierung Programmausführung auf Schirm	`TRACE`
UNTIL	Schleifenbedingung (am Ende)	`UNTIL A=10`
USING	formatierte Ausgabe	`PRINT USING „#.##“;A`
USR	Funktion: benutzerdefiniertes MC-Programm	`A=USR(B)`
VAL	Funktion: wandelt String in numer. Wert	`A=VAL(B$)`
VARPTR	Funktion: ermittelt Speicheradressen von Variablen	`A=VARPTR(A)` `A=VARPTR(A$)`
WAIT	Warten auf Portereignis	`WAIT 5,10,10`
WEND	Schleifenende zu WHILE	`WEND`
WHILE	Schleifenkopf	`WHILE A<B`
WIDTH	Ausgabebreite einstellen	`WIDTH 40`
WINDOW	Ausgabefenster für PRINT definieren	`WINDOW 1,30,0,63`
XOR	Operator: Exklusiv-Oder	`A=B XOR C`

Liste der Fehlermeldungen

Fehlertext	Bedeutung	Nr.
ARITHMETIC OVERFLOW	Überlauf bei Berechnung (>1.70141 184 E+38)	06
CAN'T /0	Division durch Null	11
CAN'T CONTINUE	Fortsetzung nicht möglich	17
FILE NOT FOUND	Datei nicht gefunden	19
FNRETURN W/O FUNCTION CALL	FNRETURN ohne FN-Aufruf	23
ILLEGAL DIRECT	unzulässiges Direktkommando	12
ILLEGAL FUNCTION	unzulässige Funktion, Parameterfehler	05
MISSING STATEMENT NUMBER	Anweisungsnummer fehlt (Zeilennummer)	24
NEXT W/O FOR	NEXT ohne FOR	01
NO STRING SPACE	kein Platz für String	14
NO SUCH DIR	kein solches Verzeichnis vorhanden	20
OUT OF DATA	kein DATA-Element mehr	04
OUT OF MEMORY	Speicherplatz reicht nicht	07
RE-DIMENSIONED ARRAY	Feld darf nicht erneut dimenioniert werden	10
RECOVERED	Wiederherstellung nach Warmstart Basic	22
RESUME W/O ERROR	RESUME ohne Fehler	27
RETURN W/O GOSUG	RETURN ohne GOSUB	03
RTC FAILURE	Fehler Echtzeituhr	28
STRING TOO LONG	String zu lang	15
SUBSCRIPT OUT OF RANGE	Feld-Parameter außerhalb zulässiger Bereich	09
SYNTAX ERROR	Sytax-Fehler	02
TOO COMPLEX	Ausdruck zu komplex	16
TYPE MIS-MATCH	Typ unpassend (String/numerisch)	13
UNDEFINED STATEMENT	Zeilennummer exisiert nicht	08
UNDEFINED USER CALL	undefinierte Benutzerfunktion FN...	18
UNTIL W/O REPEAT	UNTIL ohne REPEAT	26
USB ERROR	USB-Fehler (Sammelmeldung)	21
WEND W/O WHILE	WEND ohne WHILE	25

Meldungen ohne Nummer:

*EXTRA LOST	Zuviele Daten eingegeben, Rest wird ignoriert
FILE ALREADY EXIST'S OVERWRITE (Y)	Datei bereits vorhanden, Überschreiben?
INVALID INPUT	String statt Zahl eingegeben
NO BASI	kein BASIC-Programm (zu speichern)

Import von Programmen aus GS-BASIC

Ist man GS-BASIC gewohnt, so möchte man vielleicht gerne einige Programme auch unter *AC1-BASIC6* benutzen und mit Farbe „aufhübschen“... Da die Interpreter unterschiedliche Token verwenden, ist ein direktes Laden nicht möglich. Bei folgenden Anweisungen gibt es darüber hinaus Unterschiede in der Syntax:

Wirkung	**GS-BASIC**	**AC1- BASIC6**
Pixel setzen	`SET (x,y)`	`SET x,y`
Pixel rücksetzen	`RESET (x,y)`	`RESET x,y`
Kursor positionieren	`LOCATE (x,y)`	`CURSOR x,y`
Tastaturabfrage ohne Warten	`INKEY A$`	`A$=INKEY$`
Protokollierung ein	`TRON`	`TRACE 1`
Protokollierung aus	`TROFF`	`TRACE 0`
MC-Programm-Aufruf hex.	`CALL*1900`	`CALL&1900`
Laden/Sichern Felder	`CLOAD* „name“` `CSAVE* „name“`	`(nicht vorhanden)`
dez. Adressen >=8000h	`negative Wertangabe!`	`normal`

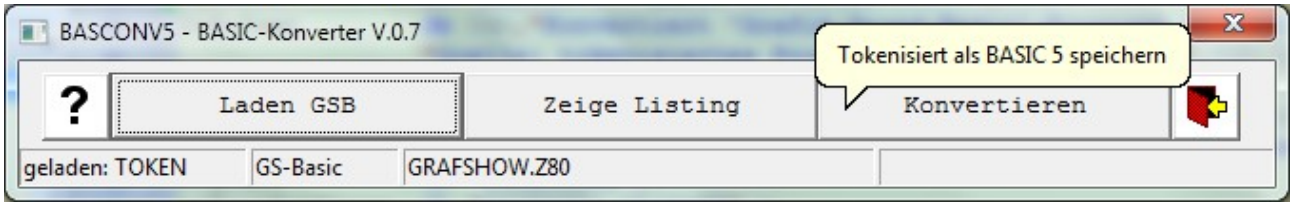
Für die Umwandlung nach *AC1-BASIC6* gibt es folgende Möglichkeiten:

1. Handarbeit:
   - Vom GS-BASIC-Programm ein ASCII-Listing erstellen (z.B. mit JKCEMU).
   - Listing in *AC1-BASIC6* eintippen, dabei die teilweise unterschiedliche Syntax beachten

2. JKCEMU:
   - ASCII-Listing des GS-BASIC-Programms erstellen und als Datei abspeichern
   - AC1-BASIC6 in JKCEMU starten
   - stückweise das ASCII-Listing über die Zwischenablage und „Bearbeiten/Einfügen...“ auf den Bildschirm holen
   - Nacharbeiten vornehmen (Syntax)
   - Funktioniert langsam, aber relativ problemlos (mit Ausnahme von Grafikzeichen, diese lassen sich nicht über die Zwischenablage transportieren).
   - unter JKCEMU abspeichern und per USB-Stick auf den AC1

3. Mein Windows-„BASIC-Konverter“:

Für WINDOWS-Nutzer habe ich ein kleines Werkzeug erstellt, was die abweichende Syntax einiger Anweisungen berücksichtigt, die Token umcodiert und die Pointer umrechnet:



Das Tool läuft ab Win-XP und auch unter 64Bit-Systemen. Es muss nicht installiert werden, einfach in irgendein Verzeichnis legen. Mit „Laden GSB“ eine GS-BASIC-Programmdatei wählen und „Konvertieren“ drücken, das war's...

Akzeptiert werden beim Laden:

a) „blankes“ tokenisiertes Programm ab 60F7h
b) mit Systemzellen gespeichertes Programm ab 6000h
c) Variante a) oder b) als z80-Datei
d) Text-Datei als ASCII-Listing (fehleranfällig, je nach Schreibweisen im Quelltext!)

Im gleichen Verzeichnis des Konverters wird automatisch die neue Datei abgelegt. Diese trägt den Grundnamen der geladenen Datei, aber die Endung *.ABC. Existiert bereits eine gleichnamige Datei, so kann überschrieben, ein neuer Name gewählt oder abgebrochen werden.

Zu kontrollieren und ggf. zu ändern ist lediglich die maximal zulässige Länge des Dateinamens von acht Zeichen; hier wurde kein Automatismus benutzt.

Diese neue Datei sollte im Normalfall unter *AC1-BASIC6* ladbar und meist auch sofort auszuführen sein...

Es lassen sich (zu Kontrollzwecken) auch *AC1-BASIC6*-Dateien laden und als Listing anzeigen (Konvertieren ist dann gesperrt, weil nicht nötig).

Besonderheiten & Einschränkungen:

Trotz korrektem Abtippen, Übernahme per JKCEMU oder „formal richtiger“ Konvertierung müssen Programme nicht unbedingt lauffähig sein:

- Enthält das Basic-Programm Maschinencode im Bereich 2000...3FFFh, wird es ohne Änderungen nicht funktionieren, da in diesem Bereich ein Teil des BASIC-Interpreters liegt.

- GS-BASIC benutzt für für dezimale Adressangaben >=8000h bei POKE/DOKE/CALL/USR negative Werte. Diese werden von AC1-BASIC6 als Fehler bewertet und sind entsprechend abzuändern: wenn Adresse negativ, dann `65536+Adresse`

- Von GS-BASIC her gewohnte Steuerzeichen in PRINT-Anweisungen (z.B. die direkte Kursorpositionierung oder Invers-Umschaltung) werden unter *AC1-BASIC6* zwar ordnungsgemäß gelistet und auch ausgeführt (vgl. Steuerzeichen) Man darf die Zeilen jedoch nicht mit dem EDIT-Befehl ändern, sonst gehen die Steuerzeichen verloren! Besteht also die Notwendigkeit, eine solche Zeile unter *AC1-BASIC6* zu ändern, so ist sie neu zu schreiben und dafür die CURSOR- und/oder CHR$-Anweisungen zu benutzen. Beispiel:

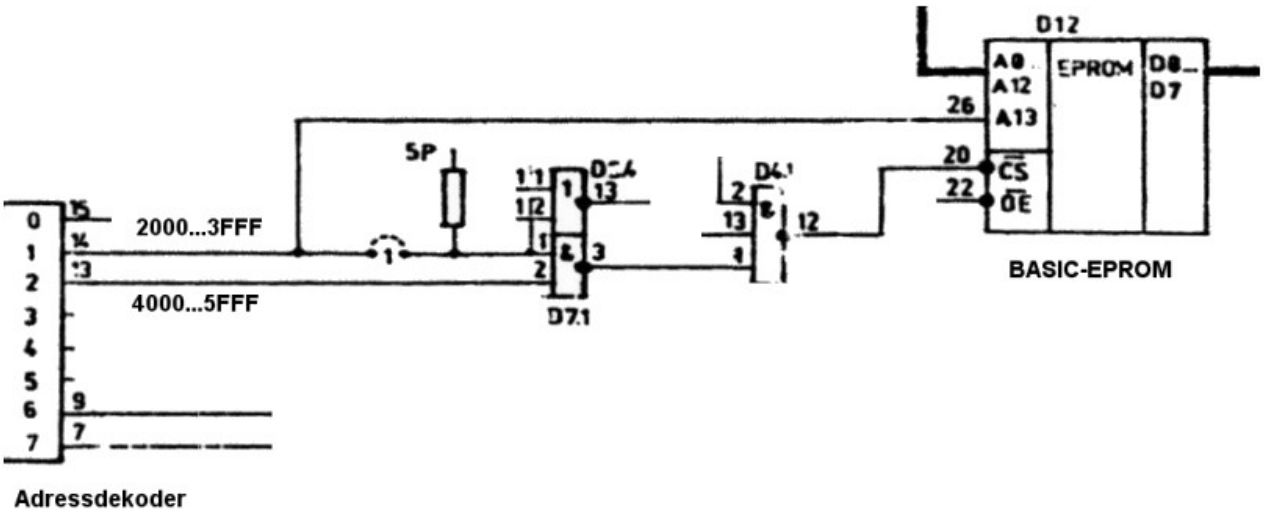

```
230 CURSOR 40,30:PRINT CHR$(17);“BITTE WARTEN...“;CHR$(16)      oder
230 PRINT CHR$(14);“3040“;CHR$(17);“BITTE WARTEN...“;CHR$(16)
```

- Prinzipiell sind auch GSB-Programme vom LLC importierbar. Ob sie auf dem AC1 lauffähig sind, hängt jedoch vom Inhalt ab. Direktzugriffe auf Ports und den Bildwiederholspeicher (des LLC2) müssen logischerweise auf die Werte des AC1 händisch umgestellt werden. Das gilt insgesondere für Programme mit „schwer durchschaubarem“ MC-Anteil in REM-Zeilen oder DATA-Anweisungen. Auch die SOUND-Anweisung ist betroffen. Da der LLC2 eine Taktfrequenz von 3 MHz besitzt, sind alle Tonlängenangaben entsprechend zu verkürzen (Faktor ca. 0,66).

AC1-BASIC6 auf EPROM

Der Interpreter ist (wie GS-BASIC) in EPROM lauffähig. Er kann in einen 27(C)128 gebrannt und im Zusammenhang mit MODUL1 benutzt werden. Modul 1 ist bereits für den Einsatz eines 16k-BASIC-EPROMS konzipiert. Die Brücke (1) ermöglicht, dass der BASIC-EPROM auch bei Adressen 2000...3FFF aktiviert wird.

Schaltungsauszug:



Achtung:	Es gibt verschiedene Varianten von MODUL1, die sich u.a. auch in der Ansteuerung des BASIC-EPROMS unterscheiden.

Original SCCH: siehe Schaltungsauszug	A13 des BASIC-Eproms wird vom Ausgang Q1 (Pin 14) des Adressdekoders gesteuert (low bei Adressen 2000...3FFF). Hier gibt es keine Probleme. EPROM wechseln, 16k-Brücke setzen, fertig.
Version 1:	A13 des BASIC-Eproms liegt am Adressbus A13. Ist der 16k-Jumper am Adressausgang vorhanden, ist auch diese Version kein Problem. Es muss nur der EPROM „seitenverkehrt“ gebrannt werden, d.h. zuerst folgt der Teil 4000...5FFFh, dann der von 2000...3FFFh. EPROM wechseln, Jumper setzen, fertig.
Version 2:	A13 des BASIC-Eproms liegt fest auf 5P (da nur für 8k-Interpreter vorgesehen). Hier ist nötig: • Abtrennen von A13 des BASIC-Eproms (Pin 26) von 5P und Leitung zu Adressdekoder Q1 (Pin 14). • Kontrolle der Schaltung, ob Q1 ebenfalls den BASIC-EPROM per /CS aktiviert, ggf. Änderung/Erweiterung nötig!
Version 3:	A13 des BASIC-Eproms ist nicht belegt (da nur für 8k-Interpreter vorgesehen). • Brücke von A13 des BASIC-Eproms (Pin 26) zu Adressdekoder Q1 (Pin 14) • Jumper "2000...3FFF" setzen

Hinweise:

1. *AC1-BASIC6* wird als z80-Datei zur Verfügung gestellt. Für das Brennen in einen EPROM ist der Header (32 Byte am Anfang) wegzulassen.

2. Die Software ist so gestaltet, dass wie bisher das GS-BASIC nun auch das AC1-BASIC6 mit den Kennbuchstaben „b“ (kalt) bzw. „r“ (warm) aus dem Monitor gestartet werden kann.

3. Die Rücksprungadesse zum Monitor ist anpassbar (Standard: 006E), siehe Systemzellen!

4. Es ist auch möglich, den nächst größeren EPROM-Typ 27(C)256 zu verwenden. Da im Layout Pin 27 (A14) meist auf 5P liegt, ist der Interpreter in die **obere Hälfte** zu brennen. Setzt man einen zusätzlichen Schalter ein, so lassen sich GS-Basic und AC1-BASIC6 sogar im gleichen Eprom unterbringen.

5. Lässt sich AC1-BASIC6 nach des Brennen Eproms und Stecken des Jumpers nicht starten (und man ist sich nicht sicher, welche MODUL1-Version vorliegt), so kann man folgendes testen.

   - EPROM aktivieren, z.B. mit **`W 14 2`** aus dem Monitor
   - Kontrolle Speicherbelegung mit **`D 5FE0`**

```
 # D5FE0
*
 <5FE0> (H) 5FE0>  00 4C 49 53 54 0D 00 52  * .LIST..R *
            5FE8>  55 4E 0D 00 4C 4F 41 44  * UN..LOAD *
            5FF0>  22 00 53 41 56 45 22 00  * ".SAVE". *
            5FF8>  44 49 52 0D 00 00 00 00  * DIR..... *
```

   - Stehen diese Inhalte (Funktionstasten-Namen) dort nicht, wohl aber ab Adresse **`#3FE0`**, so ist es nicht die Originalschaltung und der EPROM-Inhalt ist entsprechend zu vertauschen.
   - Ist bei korrekt gebranntem EPROM dieser Inhalt weder auf **`3FE0`** noch auf **`5FE0`** zu sehen, liegt ein Problem mit der Aktivierung des EPROMs vor.

Historie/Anmerkungen

BASIC auf dem AC1

1. Der erste BASIC-Interpreter für den AC1 war das im FUNKAMATEUR 12/84 abgedruckte „2k-MINI-BASIC“. Es ermöglichte den Einstieg in diese Programmiersprache am AC1. Bald genügte er aber den Ansprüchen nicht mehr.
2. Der Berliner AC1-Klub erstellte einen 8k-Interpreter.
3. Vom SCCH gab es das „Grafik-Sound-BASIC“. Im Wesentlichen erfüllen beide noch heute ihren Zweck, mit Ausnahme der Unterstützung aller Neuheiten... Darüber hinaus ist die Rechengenauigkeit („single precision“) nicht in allen Fällen ausreichend.
4. „BACOBAS“ (F. Heyder/B. Nickel) war eine weitere Variante, die im Befehlsumfang etwa dem GS-BASIC entsprach, jedoch einen eingebauten Bascoder hatte. Damit konnte man in BASICODE arbeiten.
5. Der für den AC1 angepasste „*12k-Interpreter*“ basiert auf einem TDL-/ZAPPLE-Basic (Version 2.x). Er bringt zwar eine höhere Rechengenauigkeit (Gleitkommaformat mit 48 Bit = „anderthalbfach genau“, 11 Ausgabestellen), weist aber einige Mängel auf:
   - unvollständige/fehlerhafte AC1-Anpassung (z.B. INKEY$-Funktion)
   - unbequeme Korrekturfunktion /x/ bei Eingabefehlern
   - „Prähistorischer“ Zeileneditor
   - eingeschränkte Kompatibilität zu anderen Interpretern
6. Versuche, ein neues BASIC (15k) für den AC1 anzupassen gab es um ca. 1989. Nach der vorliegenden Beschreibung handelte es sich um den „Hübler“-Interpreter als Grundlage. Das Projekt fand jedoch kaum Verbreitung und fiel dann der „Wende“ zum Opfer...

Der „Hübler“-Interpreter im „*BASIC-Kleincomputer mit Grafik*“ [1] hat u.a. ein verbessertes TDL-/“ZAPPLE“-12-k-BASIC, Version 3.x [2] zur Grundlage und bietet einen wesentlich höheren Funktionsumfang und Features, die man sonst kaum findet. Allerdings sind seine Programme nicht kompatibel zu anderen Interpretern.

Kommentierte Quellen standen mit einer Ausnahme (2er-TDL-Version, fehlerhaft, nicht zum Assemblieren geeignet) nicht zur Verfügung. Aus diesem Grunde wurde der „Hübler-Interpreter“ reassembliert, analysiert und als Grundlage für ein neues „*AC1-BASIC6*“ verwendet. Da es gerade bei der Nutzung von Farbe keine vergleichbaren Programme für den AC1 gibt, wurde auf die Herstellung irgendwelcher Kompatibiliäten verzichtet.

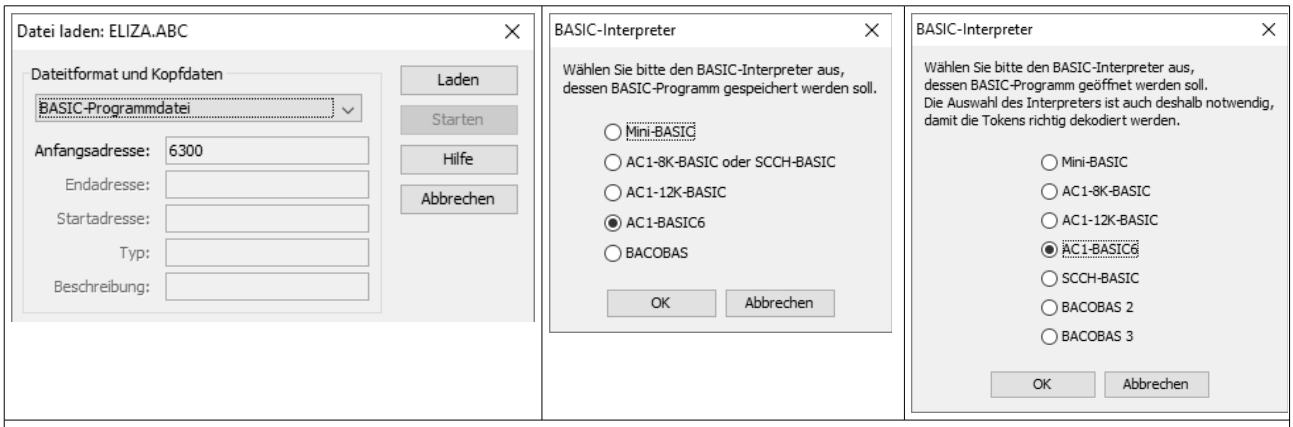
Für den Import und die Weiternutzung von GS-Basic-Programmen gibt es jedoch auch ein Werkzeug!

Modifikationen am „Hübler-“Interpreter:

- USB-Schnittstelle statt Kassettenroutinen
- Unterstützung des Farb-BWS (COLOR-Anweisung)
- Neue Funktionen, z.B. TIME$ (RTC-Abfrage)
- Anpassung an GS-BASIC:
  - Modifikation einiger Anweisungen (z.B. CLEAR)
  - Aufnahme einiger von GS-BASIC her „gewohnter“ Dinge (z.B. MODE, SOUND)
  - Umgestaltung der originalen Grafik auf Blockgrafik 128*64 und Implementierung von Linie und Kreis als schnelle MC-Routinen
- Implementierung Fenster-Technik (WINDOW-Anweisung)
- eingebauter V.24-Druckertreiber (Monitor-basiert) für zahlreiche Druckbefehle, z.B. LLIST
- Entfernung der sequenziellen Dateibefehle (die ohnedies ein Zusatz erfordert hätten)
- EPROM-fähig gemacht (selbstmodifizierenden Code geändert, Arbeitszellen+Puffer an den RAM-Anfang ab 6000h verlagert)

Arbeit mit AC1-BASIC6 USB unter JKCEMU

AC1-BASIC6 wird direkt von JKCEMU unterstützt! (Danke Jens!)



BASIC-Datei laden, speichern und im Editor öffnen per Menü-Befehle

Literatur:

Diese Anleitung entstand unter Verwendung auszugsweiser/modifizierter Beschreibungen aus [1] und [2].

[1]	*BASIC-Kleincomputer mit Grafik*“, Autor: Bernd Hübler „Mikroelektronik in der Amateurpraxis 3“
[2]	TDL-/“ZAPPLE“-12-k-BASIC, Manual z.B. hier: http://www.classiccmp.org/cini/pdf/TDL/Zapple%2012k%20Basic%20User%27s%20Manual.PDF
[3]	„commented BASIC listing of the TDL 12K Extended "Zapple" BASIC“, z.B. hier: http://www.tiffe.de/Robotron/Zapple-Basic/

In Vers. 1.1. wurde korrigiert/ergänzt:

- TIME$-Funktionen schrieben ASCII-Darstellung auf USER-Sprünge (18F0...)
  => ASCII-Darstellung entfernt, Rohdaten bleiben erhalten
- PRINT USING druckte <immer> einen * (String war zu kurz bemessen)
- neuer Formatstring für PRINT USING: ** => leere (Vorkomma-)Positionen werden mit Nullen gefüllt, Platzhalter für 2 Ziffern. Bei Nachkommastellen ist das nicht nötig, ein # erzeugt automatisch eine 0, wenn nicht belegt. Beispiel:
  40 PRINT USING "**##:**#:**";A;A;A :REM ergibt 0000:000:00 (bei A=0)
- Hinweis zu JKCEMU geändert, da nunmehr volle Unterstützung!
- „RTC FAILURR“ (Rechtschreibfehler)

Danke an die „Betatester“, besonders an „deaf_ac1“ mit seinen Ideen und konstruktiven Hinweisen zur Weiterentwicklung!

gefertigt:	WeRo, 2015
ergänzt:	2016