AC1 – Grafik - Sound- Pascal

Version 1.6

Stand 14.04.2020

Bei dieser PASCAL-Version handelt es sich um eine Portierung von "BLS-(NASCOM)-Pascal", die direkt im Monitor-Modus des AC1 lad- und ausführbar ist. Es wird kein CP/M benötigt!

Ein integrierter Treiber ermöglicht die Nutzung von USB für Speichern und Laden. Kassettenarbeit gibt es nicht.

Unter Ausnutzung aller spezifischen Möglichkeiten des AC1 können damit Pascal-Programme oft einfacher als im "Standard-PASCAL" gestaltet werden – aber diese Quelltexte sind nicht auf anderen Rechnern bzw. unter anderen Pascalversionen zu verwenden.

Der Umgang ist mit dem unter AC1-(GS-)BASIC vergleichbar: Es lassen sich Quelltexte erstellen, modifizieren, sichern und laden. Vor der Ausführung des Anwenderprogramms wird der Quelltext jedoch compiliert, d.h. in direkten Maschinencode umgesetzt. Daher läuft ein vergleichbares Programm wesentlich schneller als in BASIC.

Ist das Programm dann ausgetestet und fehlerfrei, kann eine eigenständig ausführbare Datei erzeugt werden, siehe dazu auch das "Hallo Welt"-Beispiel.

Quelltexte anderer Pascal-Versionen können unter bestimmten Voraussetzungen geladen werden. Meist sind dann Anpassungen erforderlich, da dort verwendete Standardfunktionen oder Prozeduren im AC1-PASCAL nicht existieren oder andere Namen tragen.

Inhaltsverzeichnis

Eckdaten AC1-PASCAL

"Es handelt sich bei dieser PASCAL-Version zwar nur um eine Untermenge des Standard-PASCAL, dafür aber um einen echten Compiler (kein Zwischencode)… Besonders hervorzuheben sind die Stringfunktionen (in Standard-PASCAL nicht enthalten), die 11-stellige Rechengenauigkeit und die Kombinationsmöglichkeiten mit Maschinenprogrammen über die Befehle CODE, EXTERNAL, CALL und MEM…" *Quelle: 80-Bus Journal April 1983, Ausgabe 4*

Ein solch kleiner Compilerkern - quasi ein "Mini-Pascal" - erfordert gegenüber z.B. Turbo-Pascal natürlich Abstriche in den Funktionalitäten. Neben einigen fehlenden Datentypen (z.B. kein CHAR und BYTE), fehlender Möglichkeit der Definition eigener Typen, fehlenden Standardfunktionen und -Prozeduren gibt auch keine Units und Includedateien. Was es alles nicht gibt, stellt man spätestens dann fest, wenn man versucht, ein Turbo-Pascal-Programm zu importieren…

Dafür bietet das auf den AC1 portierte Pascal einige Erweiterungen in Richtung Grafik, String-verarbeitung, direkten Speicherzugriff und "Sound", d.h. alles, was man von (AC1-)Basic kennt und mehr!

Der konkrete Funktionsumfang ist durch die Schlüsselworte in Anlage B4 gekennzeichnet.

Als Datentypen stehen zur Verfügung:

- Real,
- Integer,
- Boolean,
- String und
- Array.

Real-Genauigkeit und -Bereiche (besser als GS-Basic):
48Bit-FP-Arithmetik, mehr als 11 signifikante Stellen

```
negativ:    -1.7014118346E+38 <=R<= -2.9387358770E-39
Null:       R = 0
positiv:    2.9387358770E-39  <=R<= 1.7014118346E+38
```

Speicheraufteilung:

$1980...$1FFF	Arbeitszellen und Programmstack
$2000...$5FFF	Laufzeitmodul, Verwaltung, Editor, Compiler (ROM-fähig!)
$6000...$FFFF	Quelltext und Code

Voraussetzungen:

- SCCH-Monitor (10/88, 8, 10, 11)
- 64k RAM
- NMI-Taste
- Hardware mit PIO und VDIP für USB-Unterstützung
- ggf. Hardware für "Inversdarstellung"

Programmanpassungen (Patchtabelle):

Adr.	Standard	Bedeutung
$201E-$201F	80FF	genutztes RAM-Ende ($FF80), wenn Start ohne Argument
$2020	4	Tabulatorweite im Editor

Verwaltung

Der Start aus dem Monitor erfolgt mit `# J 2000`

Als Parameter kann eine maximal zu nutzende RAM-Obergrenze angegeben werden (< $FFFF).

Es werden der Programmtitel, der maximal verfügbare RAM sowie das Menü ausgegeben. Anschließend wird ein evtl. im Speicher befindlicher Quelltext gelöscht. Nach Anzeige des Bereitschaftszeichens (>) stehen folgende Kommandos zur Verfügung:

(B)EENDEN	Beenden von AC1-PASCAL
(C)OMPILIEREN	Compiler starten
(D)RUCK	Ausdrucken des Quelltextes per V.24
(E)DIT	Editor starten
(F)ILE	ausführbares File auf USB erzeugen
(I)NFO	Infos zum Programm
(L)ADEN	Laden eines Quelltextes
(NEU)	neues Programm, Löschen des aktuellen Programms im Speicher
(R)AM	Anzeige des belegten Speichers im RAM
(S)ICHERN	Sichern des aktuellen Quelltextes
(T)EST	Ausführen im Testmodus sofern zuvor noch nicht compiliert, erfolgt das Compilieren automatisch
(U)SB	USB-Stick wechseln/neu initialisieren/DIR anzeigen
<ENTER>	<ENTER> ohne einen Kommandobuchstaben bewirkt das Löschen des Bildschirms und Auflistung der Kommandos.

Es ist ausreichend, den jeweiligen Anfangsbuchstaben einzugeben und mit <ET> zu quittieren. Eine Ausnahme bildet das Löschkommando 'NEU', welches aus Sicherheitsgründen auszuschreiben ist. Parameter sind nicht erforderlich/wirksam.

Falsche Eingaben werden mit '`Kommando?`' quittiert.

Nach Verlassen mit 'B' (oder einem Notausstieg mit RESET) kann ein Warmstart mit `# J 2003` erfolgen (nicht versehentlich `# J 2000`, dann wäre ein ungespeicherter Quelltext futsch...) Sofern zuvor nichts vom Quelltext oder dem Compiler überschrieben wurde, ist die Weiterarbeit möglich.

Editor

Mit dem Kommando 'E' erscheint der Quelltexteditor. Als "Vollschirm-Editor" ermöglicht er die gleichzeitige Betrachtung der Nachbarzeilen und einen einfachen Zeilenwechsel.

Dargestellt werden 27 Zeilen. Sind mehr vorhanden, dann erfolgt ein automatisches Rollen, wenn der Kursor den oberen bzw. unteren Rand überschreiten möchte. Pro Bildschirmzeile können 63 Zeichen eingegeben werden.

Nach Beenden des Editors mit ESC merkt sich dieser den aktuellen Textausschnitt sowie die Kursorposition und stellt das bei Neueintritt wieder her. Bei Compilierfehlern wird der Editor automatisch aufgerufen und der Kursor (etwa) an die Fehlerstelle gesetzt.

Tastatur-Funktionen im Editor:

Code	STRG	Taste	Bedeutung	
01	^A	Pos 1	Kursor an den Textanfang	
04	^D	Entf	Zeichen unter Kursor löschen, Rest rückt auf	
05	^E	Einfg	Einfügen Leerzeichen an Kursorposition, Rest rückt nach	
06	^F		Finden Suchbegriff (ab akt. Zeile), max. 15 Zeichen	
07	^G		Gehe zu Zeile Nummer n	
08	^H	←	Kursor nach links	nur innerhalb einer Zeile, keine Überschreitung links/rechts
09	^I	→	Kursor nach rechts	
0A	^J	↓	Kursor nach unten	bei Überschreitung der ersten bzw. letzten Zeile wird gescrollt
0B	^K	↑	Kursor nach oben	
0C	^L		Leerzeile einfügen	
0D	^M	ENTER	Anfang der nächsten Zeile, am Textende wird eine neue Zeile erzeugt	
0E	^N		Nächstes ab akt. Zeile finden	
0F	^O	→|	Tabulator 4 Zeichen, variabel (siehe Patchtabelle)	
11	^Q	Bild ↑	Kursor an Seitenanfang bzw. eine Seite zurück	
13	^S		Streichen Zeile	
15	^U	Bild ↓	Kursor 1 Seite weiter	
18	^X		Kursor an Zeilenende	
19	^Y		Kursor an Zeilenanfang	
1A	^Z	Ende	Kursor an das Textende	
1B	^Ä	Esc	Editor verlassen	
7F		Backspace	Zeichen links vom Kursor löschen	

Anmerkungen:

- In der aktuellen Version belegt jede Editorzeile unabhängig von ihrer tatsächlichen Länge 64 Zeichen im Puffer. Damit war ein relativ einfacher Editor möglich, er ist aber "Platzfresser" (max. ca. 500 Programmzeilen, je nach Größe des erzeugten Codes). Sichern und Laden erfolgt jedoch im "komprimierten Format", d.h. ohne die überflüssigen Leerzeichen am Zeilenende und mit 0Dh/0Ah als Zeilentrenner.
- Die Tastatur arbeitet mit "CAPS-LOCK", d.h. normal sind Großbuchstaben. Um Kleinbuchstaben zu erzeugen, ist zusätzlich die SHIFT-Taste zu betätigen.
- Es gibt nur den "Überschreiben"-Modus, d.h. die Zeichen rechts von der Eingabeposition rücken beim Schreiben nicht nach.
- Die Suchfunktion arbeitet immer ab der aktuellen Kursorposition und nur vorwärts. Ggf. ist also der Kursor zuvor mit "POS1" an den Textanfang zu setzen. Wurde nichts (mehr) gefunden, verbleibt der Kursor an der letzten aktuellen Position.
- Alle Änderungen am Quelltext werden sofort vorgenommen. Es gibt keine "Rückgängig"-Funktion (außer man lädt eine zuvor gesicherte Version komplett neu).

Compiler

Der Compiler übersetzt den ASCII-Quelltext in ausführbaren U880-Code. Nach Aufruf mit 'C' wird der Quelltext zunächst auf (Schreib-)Fehler untersucht. Werden welche gefunden, so ergeht eine Fehlermeldung:

		40	BEGIN erwartet
01	Syntaxfehler (z.B. Semikolon fehlt)	41	THEN fehlt
02	'=' erwartet	42	CASE-Selektor muss Ganzzahl oder String sein
03	':' erwartet	43	OF fehlt in CASE-Anweisung
04	'[' erwartet	44	END fehlt
05	']' erwartet	45	DO fehlt in WHILE-Anweisung
06	'(' erwartet	46	Ganzzahl-Variable erwartet
07	')' erwartet	47	TO der DOWNTO fehlt in FOR-Anweisung
08	',' erwartet	48	DO fehlt in FOR-Anweisung
09	'.' erwartet	49	Marke wurde nicht deklariert
10	'.." erwartet	50	TO fehlt in INIT-Anweisung
11	':=" erwartet		
20	Untere Feldgrenze > obere	60	Stringtyp hier nicht erlaubt
21	Überlauf in Feldvereinbarung	61	Ganzzahlausdruck erwartet
22	'OF' fehlt in Felddeklaration	62	Stringausdruck erwartet
23	Unzulässiges Zeichen im Bezeichner	63	Typfehler im Ausdruck
24	Stringlänge darf nicht 0 sein	64	Unbekannter Bezeichner im Ausdruck
25	Unbekannter Datentyp	65	Syntaxfehler, numerischer Konstantenüberlauf oder Stringkonstante enthält Wagenrücklauf-zeichen
		66	Stringkonstante zu lang
30	Ganzzahl-Konstante erwartet	70	Typfehler in Zuweisung oder Parameterliste
31	String-Konstante erwartet	71	Unbekannter Variablenbezeichner
32	Real-Konstante erwartet	72	Unbekannter Feldbezeichner
33	Ganzzahl-Konstante 0<=i<=255 nötig		
		80	LABEL deklariert und referenziert, fehlt aber
		99	Unerwartetes Quelltextende

Nach Quittierung der Fehlermeldung mit der Leertaste erscheint der Editor. Der Kursor wird automatisch etwa an die Stelle positioniert, an der der Fehler im Quelltext aufgetreten ist. Fehlt z.B. ein Ende-Semikolon, so steht der Kursor dann meist am Anfang der nächsten Zeile.

In einigen Fällen kann der Compiler nicht exakt feststellen, wo der eigentliche Fehler liegt und gibt eine ggf. unpassende Fehlermeldung und Fehlerposition hinter der eigentlichen Fehlerstelle aus. Beispiel: unerlaubtes Verlassen einer FOR-Schleife mit GOTO (auf GOTO sollte und kann ganz verzichtet werden!)

Programmausführung

Nach erfolgreicher Compilierung ("OK"-Meldung) befindet sich das ausführbare Programm im Speicher (im Anschluss an den Quellcode). Dieses kann nun mit 'T' im Testmodus gestartet werden kann. Es ist auch möglich, auf 'C'-Kommando zu verzichten und sofort mit 'T' zu beginnen. Dann wird zunächst compiliert und anschließend das erzeugte Programm gestartet. Für einen erneuten Programmstart muss nicht compiliert werden. Nach jedem "Besuch" im Editor wird das ausführbare Programm gelöscht, sodass beim nächsten 'T' zunächst wieder compiliert wird.

Bezogen sich o.a. Fehler auf das Compilieren, so können weitere beim Ausführen des Objektcodes auftreten ("Laufzeitfehler"):

01	Gleitkommaüberlauf	10	Ergebnisstring wäre länger als 255 Zeichen oder die Position in einer MID-Funktion ist negativ oder Null
02	Division durch Null		
03	Quadratwurzel aus negativer Zahl	20	Feldindex außerhalb definiertem Bereich
04	Logarithmus zu nicht positiver Zahl		
05	reelle Zahl bei Konvertierung zu Integer zu groß	99	Speicherüberlauf, zu wenig RAM

Da der Quellcode syntaktisch richtig ist, gestaltet sich die Fehlersuche in diesen Fällen etwas schwieriger. Ein Sprung in den Editor wie beim Compilieren erfolgt nicht.

Hinweise:

- Ein mit 'T' gestartetes Programm kann (sofern im Programm selbst kein Ausstieg vorgesehen ist oder es versehentlich in eine Endlosschleife geriet) mit NMI abgebrochen werden, was einen Warmstart von AC1-PASCAL bewirkt.
- Der ausführbare Code liegt im Testmodus direkt nach dem Quelltext (d.h. an nicht vorausbestimmbarer Adresse) im Speicher.

Erzeugung eines lauffähigen Files

Wurde das Anwenderprogramm mit Compile/Test soweit getestet, dass es wunschgemäß und fehlerfrei funktioniert, kann mit dem 'F'-Kommando ein eigenständig lauffähiges File erzeugt und auf USB-Medium abgelegt werden. Dieses benötigt das AC1-PASCAL dann selber nicht.

- Die Files haben generell die gleichen Namen wie das aktuell geladene File mit Ausnahme der anderen Endung *.BIN und werden automatisch auf das USB-Medium geschrieben.
- Alle so erzeugten Files sind auf einem AC1 mit den o.a. Mindesteigenschaften lauffähig.
- Eine Prüfung auf Existenz eines gleichnamigen Files erfolgt nicht; es wird daher immer die letzte mit 'F' erzeugte Version gespeichert.
- Die Startadresse der BIN-Files ist immer $2000. Das BIN-File ist aktuell mindestens 5433 Bytes lang, wovon 5420 Bytes auf das integrierte Laufzeitmodul entfallen.
- Wurde das 'F'-Kommando mit einem fehlerhaften Quelltext aufgerufen, so erfolgt wie beim "normalen" Compilieren ein Abbruch und Sprung in den Editor.
- Das BIN-File kann nicht aus dem AC1-Pascal heraus geladen/gestartet werden. Dazu ist AC1-PASCAL zu beenden und das File per Monitor zu laden.
- Tritt bei der Ausführung des Anwenderprogramms ein Laufzeitfehler auf, so erfolgt nach der Fehlermeldung ein Abbruch und Rücksprung in den Monitor.
- Fehlt das USB-Medium oder die nötige Hardware, ergeht wie beim Laden/Speichern von Quelltexten die Meldung "`ND USB-Fehler!`".

Besonderheiten

Tastatur-Eingaben

READ(x) und READLN(x)

Beide Anweisungen sind identisch bis auf die Tatsache, dass die mit "LN" nach der Eingabe den Kursor an den Anfang der nächsten Zeile setzt.

- Eingabe eines Wertes per Tastatur je nach Datentyp von x (String/Integer/Real), <ENTER> übernimmt
- Wurde die Variable als STRING[1] definiert, so ist kein <ENTER> erforderlich und es erfolgt auch keine Anzeige des eingegebenen Zeichens.
- Wurde x vorher ein Wert zugewiesen, so wird dieser als Voreinstellung übernommen, wenn nur <ENTER> gedrückt wird.
- Bei unzulässiger Eingabe wird die Anzeige gelöscht, dann ist eine Neueingabe erforderlich.
- Korrektur des zuletzt eingegebenen Zeichens mit Backspace ($7F) möglich.
- ^C bricht eine begonnene Eingabe ab. Wie bei nur <ENTER> enthält die Variable noch den alten Wert (Voreinstellung).

KEYBOARD

- x:=KEYBOARD liest ein Tastaturzeichen
- wartet nicht auf Tastenbetätigung
- wenn Taste gedrückt: ASCII-Wert, wenn keine Taste: Rückgabewert 0 (FALSE)
- Nutzung z.B. für "Warten auf Tastendruck": `REPEAT UNTIL KEYBOARD;`

Bildschirmausgaben

WRITE und WRITELN

Mit den Anweisungen können Zahlen, Einzelzeichen oder Texte (direkt oder als Ausdrücke) ausgegeben werden. Beide Anweisungen sind identisch bis auf die Tatsache, dass mit "LN" nach der Ausgabe der Kursor an den Anfang der nächsten Zeile gesetzt wird.

Mit `WRITE(CHR(x));` werden alle Zeichen im Bereich x=$20...$7E (normaler ASCII-Zeichensatz) angezeigt. Die Ausgabe von Codes im Bereich $01...$1F bewirken die entsprechenden AC1-Bildschirmsteuerfunktionen. Soweit die nötige Hardware dazu vorhanden ist, kann also auch z.B. invers ausgegeben oder ein Drucker ein-/ausgeschaltet werden.

AC1-Pseudografik (Codes ab $80) ist normalerweise mit WRITE nicht möglich. Es erscheinen dann Zeichen mit dem Code x-128. Falls eine Ausgabe von Pseudografikzeichen dennoch gewünscht wird, ist zuvor mit `MEM[$1820]:=MEM[$1820] OR $80` auf Grafikausgabe umzuschalten. Dann liefert z.B. WRITE(CHR($E6)) im Standardzeichensatz anstatt eines 'f ' das Geistsymbol .
Spätestens am Programmende sollte man mit `MEM[$1820]:=MEM[$1820] AND $7F` die Ausgabe wieder auf "normal" zurückstellen.

Erreichen Ausgaben den unteren Bildschirmrand, wird der gesamt Bildschirminhalt automatisch nach oben gescrollt.

Kursorpositionierung

Die direkte Kursorpositionierung mit `SCREEN(x,y)` ist im Bereich 0...x...63 und 0...y...31 möglich. Enthält eine der beiden Koordinaten einen ungültigen Wert außerhalb dieser Bereiche, so wird die letzte Kursorposition beibehalten.

Alternativ können AC1-spezifisch auch benutzt werden:

- das Direktpositionierungskommando des Monitors: `WRITE(CHR(14),'1005NAME:';` schreibt z.B. die Zeichenkette 'NAME:' auf Zeile 10 / Spalte 5.
- die "zu-Fuß-Methode", indem die AC1-Kursor-Systemzelle per `mem[$1800]:=...` und `mem[$1801]:=...` auf die gewünschte Bildschirmadresse gesetzt wird ($17FF= links oben, $1000= rechts unten).

Blockgrafik

Die AC1-Blockgrafik (128*64 Pixel) wird mit dem `PLOT(x,y,m)`-Befehl ausgegeben, wobei:
x = x-Koordinate 0...127
y = y-Koordinate 0...63
m= Modus → 0=Setzen, 1=Rücksetzen, 2=Invertieren des Pixels

0,0 ist das Pixel links oben. Bei ungültiger Koordinatenangabe (keine Ganzzahl oder außerhalb der Bereiche) hat der Befehl keine Wirkung.

Ob ein Pixel gesetzt ist oder nicht, lässt sich mit dem `POINT(x,y)`-Befehl testen. Bei gesetztem Pixel wird TRUE (-1) zurückgegeben, bei nicht gesetztem Pixel oder ASCII-Zeichen an der entsprechenden Bildschirmposition ein FALSE (0).

Zu den PLOT-/POINT-Anweisungen gibt es gegenüber AC1-BASIC eine Erweiterung: Es kann nicht nur auf dem Bildschirm sondern auch "unsichtbar" in einen anderen Speicherbereich (gleicher Größe) gezeichnet werden. Dazu ist vor der ersten PLOT-/POINT-Anweisung anzugeben, welcher Bereich genutzt werden soll. Das erfolgt über einen Eintrag in Arbeitszelle $1A00/1A01. Dort ist (wegen des rückläufigen Beschreibens) jeweils die höchste Speicheradresse einzutragen:

```
INIT mem[$1A00] TO $FF,$17;
INIT mem[$1A00] TO $FF,$EF;
```

Zeichnen auf Bildschirm $1000...$17FF
Zeichnen in einen Puffer $E800...$EFFF

Im TEST-Modus wird standardmäßig mit "Bildschirm" gearbeitet, sofern nichts anderes festgelegt wurde. Sollen eigenständig lauffähige Files mit PLOT-/POINT-Anweisungen erzeugt werden, so ist das Ziel auch bei "Bildschirm" unbedingt anzugeben!

Mit einer kleinen und schnellen Assembler-Routine lässt sich der Pufferbereich auf den Bildschirm übertragen bzw. schnell löschen (siehe auch Kapitel INLINE-Assembler):

Puffer auf Bildschirm holen:

```
PROCEDURE COPY;
CODE $21,$FF,$EF,$11,$FF,$17,$01,$00,$08,$ED,$B8;
```

```
LD  HL,$EFFF
LD  DE,$17FF
LD  BC,$0800
LDDR
```

Puffer im Hintergrund löschen:

```
PROCEDURE CLEAR;
CODE $21,$FF,$EF,$36,$20,$54,$5D,$1B,$01,$00,$08,$ED,$B8;
```

```
LD  HL,$EFFF
LD  (HL),' '
LD  D,H
LD  E,L
DEC DE
LD  BC,$0800
LDDR
```

Benutzt wird diese Methode z.B. im Programm "CUBUS".

Direkter Bildspeicherzugriff

Über das vordefinierte mem-Array kann auf alle Speicher-Adressen lesend und schreibend zugegriffen werden. Das gilt auch für den Bildwiederholspeicher. So lässt sich z.B. mit `mem[$17FF]:=$0D` ein Block- oder Pseudografikzeichen direkt auf den Bildschirm bringen.

Organisation des Bildwiederholspeichers:

```
oberste Zeile  $17FF...$17C0      ;insgesamt
2. Zeile       $17BF...$1780      ;32 Zeilen
…                                 ;zu je
unterste Zeile:$103F...$1000      ;64 Zeichen
```

Bei der Anpassung von originalen NASCOM-Pascal-Quellprogrammen, welche auch die Methode des direkten Bildschirmzugriffs benutzen, ist zu beachten, dass dieser Rechner eine völlig andere Adressierung des Bildwiederholspeichers hat.

Zeichensätze

Auch unter AC1-PASCAL lässt sich der Zeichensatz umschalten, sofern im AC1 zwei Sätze definiert sind und die Steuerung per PIO B3 vorhanden ist: `OUT(5,(INP(5) EXOR 8));`

Ist ein Zeichensatz mit eckigen Klammern vorhanden (AC1-Standard, "unten" im EPROM), so wird dieser im Editor benutzt. Damit erscheinen in der Quelltextbearbeitung keine deutschen Umlaute, jedoch die benötigten eckigen und geschweiften Klammern. Sollen in auszugebenden Strings deutsche Umlaute erscheinen, so ist im Quelltext die entsprechende Klammer anzugeben. Für den String 'Ä Ö Ü ä ö ü ß' erscheint im Quelltext also `'[ \ ] { | } ~`. Da sich die Benutzung deutscher Umlaute in Grenzen halten dürfte, sind diese Ausnahmen sicher zu tolerieren.

Tonausgabe

Klingelton

Der bekannte AC1-(SCCH-)Klingelton kann wie folgt aufgerufen werden:

```
WRITE(CHR(7));
```
oder
```
CALL($0272);
```

SOUND

Die neue Prozedur `SOUND(H,L)` ermöglicht eine Einzeltonausgabe, wie sie auch vom GS-BASIC her bekannt ist. H und L sind Ganzzahlausdrücke (0...255). Die SOUND-"Noten" aus GS-Basic-Programmen können damit auch in AC1-PASCAL angewendet werden.

Beispielswerte:

H= Tonhöhe	**140**	**125**	**118**	**104**	**93**	**87**	**78**
Frequenz	440	494	523	589	659	599	782
Notenwert (ca.)	a	h	c	d	e	f	g

L= Tonlänge	**0**	**1**	**2**	**5**	**10**	**25**	**255**
Sekunden (ca.)	0,04	0,08	0,12	0,24	0,44	1	10

Anwendungsbeispiel

Arbeit mit Maschinencode

Grundprobleme

1. Für eigene Routinen muss ein geeigneter Speicherbereich festgelegt werden, den PASCAL unberührt lässt, z.B.:
   a) Einschränkung des oberen RAM-Endes (Start von AC1-PASCAL mit Parameter) ,
   b) Nutzung des freien Bereiches $1900...$198F oder
   c) Nutzung eines Arrays.
2. Der Maschinencode muss in den Speicher gebracht werden, z.B. mit der INIT mem-Anweisung: `INIT mem[$1900] TO $3E,$0C,$D7,$C9`
3. Eventuelle Parameterübergaben müssen organisiert werden.

Einige wichtige Adressen

AC1-Monitor:

$0287	Tonerzeugung	Register B=Tonlänge, C=Tonhöhe
$07EB	Pause 30 ms	
$07EE	Ausgabe A als Hex-Zahl	
$07F1	Ausgabe HL als Hex-Zahl	

Arbeitszellen:

$1800	enthält Bildschirmadresse der aktuellen Ausgabeposition
$1900… $198F	frei für z.B. MC-Routinen
$19A2	Workspace-Stackpointer (WSP) → für Parameterübergabe
$19D0	Beginn Eingabepuffer
$1A00	enthält Speicheradresse für Arbeitsbereich PLOT/POINT, Standard 17FF=Bildschirm

Sprungverteiler am Programmanfang:

$2000	Kaltstart	
$2003	Wiederherstellen (Warmstart)	
$2006	(noch frei)	
$2009	USB initialisieren	
$200C	Bildschirminhalt sichern	vorher Dateiname auf $19C0 ablegen, siehe Beispiel!
$200F	Bildschirminhalt laden	
$201E-$2020	Voreinstellwerte	

CALL und EXTERNAL

Für die Ausführung der MC-Routinen stehen die CALL-Anweisung und die EXTERNAL-Spezifikation einer Prozedur zur Verfügung, die prinzipiell das gleiche bewirken. EXTERNAL ermöglicht jedoch eine Parameterübergabe per Stack (siehe unten).

`CALL($07EB);`	ruft ein Maschinenprogramm auf der angegebenen Adresse auf, hier eine Pause von 30ms
`PROCEDURE PAUSE;` `EXTERNAL $07EB;`	

Inline-Assembler

Kleine Maschinenroutinen kann man auch direkt im Quelltext einer Prozedur oder Funktion einbinden und dann beim Aufruf der Prozedur oder Funktion sofort ausführen, ohne z.B. CALL zu bemühen. Eine asm-Asweisung wie im "großen" Turbopascal fehlt leider. Als Ersatz gibt es die CODE-Spezifikation in einer Prozedur oder Funktion. Hierbei beinhaltet diese lediglich die hexadezimalen Codebytes. Wie bei EXTERNAL sind auch bei CODE sowohl der Deklarationsteil als auch der Anweisungsteil leer. Beispiel:

```
PROCEDURE CLS;
      CODE $3E,$0C,$D7;
```

```
LD  A,0Ch  ;Steuerzeichen BS-Löschen
RST 10h    ;AC1-Zeichenausgabe
```

```
BEGIN
      CLS;
      WRITELN('HALLO WELT');
END.
```

```
Anwendung der Prozedur:
Löscht den Bildschirm
```

Es ist auch möglich, 16-Bit-Werte an die Maschinencode-Routine zu übergeben:

```
PROCEDURE TEST(X1,X2: INTEGER);
```

Das Auslesen der Parameter erfolgt mit Hilfe des "Workspace-Stackpointers" (WSP) in Systemzelle $19A2. Damit lassen sich die Parameter wie folgt ermitteln (der zuletzt angegebene Parameter steht oben im Stack):

```
LD      IX,(19A2h)     ;akt. WSP, enthält Zeiger auf Parameterliste
LD      A,(IX-2)       ;→ Parameter X2      (niederwertiger Teil)
...
LD      A,(IX-3)       :                    (höherwertiger Teil)
...
LD      A,(IX-4)       ;→ Parameter X1
...
```

Anwendungsbeispiel: Bildschirm schnell komplett mit vorgegebenem Zeichen füllen:

```
PROCEDURE MUSTER(X: INTEGER);
CODE $DD,$2A,$A2,$19,
     $DD,$7E,$FE,
     $21,$FF,$17,
     $77,
     $54,
     $5D,
     $1B,
     $01,$00,$08,
     $ED,$B8;
BEGIN
     MUSTER($2A);
END.
```

```
LD    IX,($19A2)    ;Zeiger auf Parameter
LD    A,(IX-2)      ;Parameter auslesen
LD    HL,$17FF
LD    (HL),A        ;und verarbeiten
LD    D,H
LD    E,L
DEC   DE
LD    BC,$0800
LDDR

Bildschirm voller Sterne zeichnen...
```

Hinweise:

- Die CODE-Spezifikation darf nur voll verschiebliche Programmteile enthalten, da die Prozedur nach der Compilierung an unterschiedlichen Adressen zu liegen kommen kann.
- Bei CODE ist eine abschließende RET-Anweisung ($C9) im Gegensatz zur mit EXTERNAL angegebenen Routine nicht erforderlich (aber unschädlich).
- Auch in einer Funktion ist CODE möglich. Wird dabei nur ein Parameter verarbeitet, so ist dieser auf (WSP-2) abzuholen und der Rückgabewert auf (WSP-4) abzulegen, bei mehr Parametern entsprechend verschoben. Siehe dazu auch das Beispiel Datum und Zeit.
- Es empfiehlt sich bei Benutzung von Assemblerroutinen das Quellprogramm vor Compilierung/Testen zu sichern, denn man vertut sich evtl. schnell einmal...☺

Sichern und Laden

Quelltexte

Die Verwaltung ermöglicht mit Hilfe des integrierten USB-Treibers ein Sichern und Laden von Quelltexten per USB. Damit ist AC1-PASCAL unter (fast) jedem Monitor zu betreiben, die nötige Hardware vorausgesetzt. Getestet wurde 10/88, Mon. 8, Mon.10 und Mon.11.

Beim ersten Aufruf eines Sichern- oder Ladekommandos wird der USB-Treiber initialisiert. Das dauert einen Moment. Dafür steht er dann bei jedem weiteren Aufruf eines USB-Kommandos sofort zur Verfügung.

Nach USB-Stickwechsel oder undefiniertem USB-Verhalten kann mit dem Kommando `U <ENTER>` die Schnittstelle neu initialisiert werden und es erscheint das Inhaltsverzeichnis.

Ist kein USB-Stick angesteckt (oder fehlt die nötige Hardware dazu ganz), so erfolgt eine Meldung `ND USB-Fehler!`

Der Quelltext wird als einfache ASCII-Datei mit 0Dh/0Ah am Ende jeder Zeile gesichert.

Bei Sichern und Laden wird ein Dateiname nach `Name:` verlangt.

- max. 8 Zeichen, keine Leer- und Sonderzeichen benutzen
- keine Endung angeben (.PAS wird automatisch ergänzt)
- wird nur <ENTER> betätigt, so erscheint das USB-Inhaltsverzeichnis (unsortiert, aber gefiltert auf *.PAS-Dateien)
- Strg+C bricht den Vorgang komplett ab, Rückkehr zur Kommandoebene.

Sichern

Ein erstellter oder modifizierter Quelltext wird mit dem Kommando `S <ENTER>` abgespeichert. Nach Angabe des Dateinamens wird geprüft, ob dieser verfügbar ist. Existiert der angegebene Dateiname bereits, so erfolgt eine "`Ueberschreiben (J)`"-Abfrage. Bestätigt man diese mit "J", so wird die existierende Datei mit dem aktuellen Quelltext überschrieben, andernfalls abgebrochen.

Laden

Ein auf USB-Medium gespeicherter Quelltext wird mit `L <ENTER>` geladen. Wurde der anschließend anzugebende Dateiname nicht gefunden, so ergeht eine entsprechende Meldung und man befindet sich wieder im Kommandomodus.

"Fremde" PASCAL-Quelltexte können ebenfalls geladen werden. Bedingungen:

- Dateiendung *.PAS
- reines ASCII-Textformat, Zeilentrenner 0Dh/0Ah (oder nur 0Dh), auch am Textende! (ggf. händisch hinzufügen)
- keine Zeile darf länger als 64 Zeichen sein (sonst Abbruch mit "Restanzeige")

Bildschirminhalte

Für das Sichern und Laden des Bildschirminhaltes als Datei auf USB gibt es CALLs mit speziellen Adressen, siehe Beispiel.

BIN-Files

Das Sichern der erzeugten eigenständig lauffähigen Files (siehe hier) erfolgt automatisch. Ein Laden/Ausführen aus AC1-PASCAL ist nicht möglich.

Drucken

Steht die nötige Hardware zur Verfügung (PIO2 und Pegelwandler), kann mit der monitorinternen V24-Ausgabe auf einem Drucker mit serieller Schnittstelle gedruckt werden, ohne dass ein extra Treiber erforderlich wird. Dabei wird die aktuelle Monitoreinstellung benutzt. Achtung: Bei nicht angeschlossenem oder eingeschaltetem Drucker hängt der AC1 bei Druckausgaben fest (RESET).

Quelltexte

Das Kommando 'D' ermöglicht das Ausdrucken des Quelltextes. Beim Aufruf wird zunächst die aktuell im Monitor eingestellte Baudrate angezeigt:

```
>D
Drucken mit  9600 Bd. per V.24? (J)
```

Angezeigter Wert und Druckereinstellung müssen übereinstimmen. Erst dann ist die Abfrage mit 'J' zu bestätigen, jede andere Taste bricht ab.

Nachfolgend ergeht eine Abfrage "`Einzelblatt? (J)`". Wird diese mit 'J' beantwortet, so hält die Ausgabe nach 65 Zeilen an. Nach manuellem Blattwechsel ist eine beliebige Taste zum Fortsetzen zu drücken. Andernfalls wird von Endlospapier ausgegangen, wobei nach jeweils 65 Programmzeilen eine Trennlinie gedruckt wird.

Hinweise:

1. Der Monitor muss dafür angepasst sein, d.h. die entsprechende Timing-Korrektur für 4800 und 9600 Bd. vorliegen. Ansonsten erfolgt kein korrekter Druck!

Monitor 10/88	Mon. 8 und Abkömmlinge	Mon. 10 und Mon. 11
orig. modif. 00F9: E5 6E 00FA: E1 DD 23 00FB: 7F 23 0E39: E5 6E 0E3A: E1 DD 23 0E3B: 7F -	orig. modif. 0F9: E5 BE 0FA: E1 DD 23 0FB: 7F - E39: E5 BE E3A: E1 DD 23 E3B: 7F -	keine Modifikation erforderlich! Achtung! Der AC1 darf keine WAIT-Zyklen benutzen, diese verfälschen das Timing! Von Mon.11 gibt es aber eine angepasste Version, die auch mit WAIT-Einschüben korrekt arbeitet.

2. Das V24-Steuerbyte im Monitor ($1820) muss unbedingt auf "X-Draht" eingestellt sein (z.B. 04h für 9600 Bd), ansonsten wird die Rückmeldung "Drucker beschäftigt" (Signal CTS am AC1) ignoriert und der Text verstümmelt gedruckt (Pufferüberlauf).
3. Am Drucker ist die Einstellung "Seitenformatierung entsprechend Formularlänge" (automatische Trennlinien) abzuschalten und "Autolinefeed" zu aktivieren.

Druckausgaben im Programm

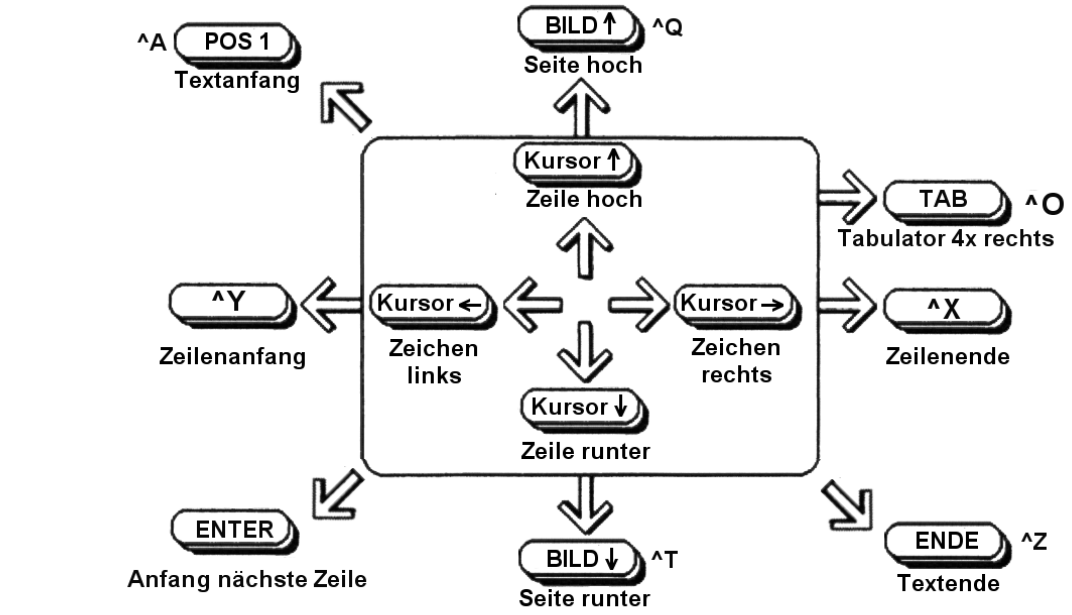
Mit `WRITE(CHR(24))` bzw. `WRITE(CHR(25))` schaltet man den Monitor-V24-Druck ein bzw. aus. Bei eingeschaltetem Drucker werden alle mit WRITE/LN vorgenommenen Ausgaben zusätzlich auf den Drucker gebracht.
Hinweise:

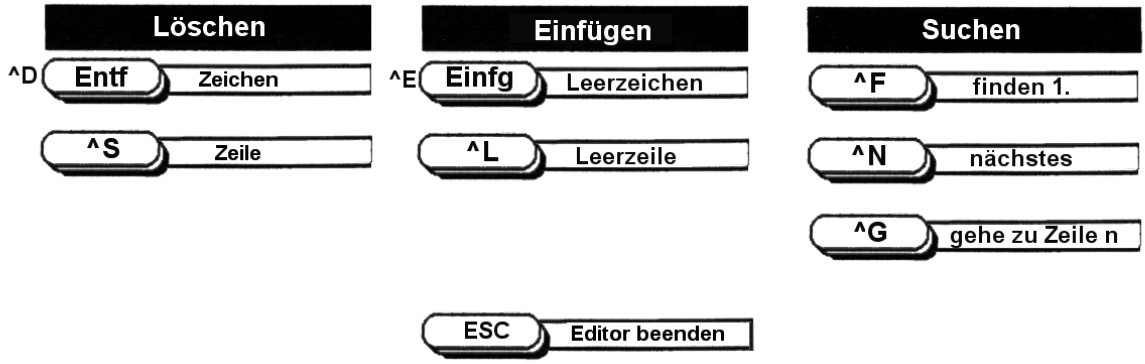
1. Ist eine ausschließliche Druckausgabe gewünscht, so muss man zuvor mit `MEM[$1821]:=…` das Monitor-IO-Byte entsprechend um- und nach dem Druck zurückstellen:
   $11= nur Bildschirm (Standard)
   $21= nur Drucker
   $31= Bildschirm+Drucker.
2. Vorsicht bei Bildschirm-Positionierkommandos (und anderen Steuerzeichen) im Programm. Je nach benutzter Methode können unvorhergesehene Effekte am Drucker eintreten!
3. Es empfiehlt sich weiterhin, den Drucker zu Beginn durch Ausgabe von ESC@ zurückzusetzen und/oder auf die gewünschten Parameter (z.B. Schrift) zu programmieren (siehe jeweiliges Druckerhandbuch).

Anlagen

A: Tastenbelegung Editor

Die meist sinnfällig gewählte Tastenbelegung resultiert aus der Benutzung einer PS/2-Tastatur mit den entsprechenden Sondertasten:





B: Programmiergrundlagen

B1: Konventionen

Für die Erstellung der Quelltexte in AC1-PASCAL gilt:

- Hexadezimalzahlen werden durch ein vorangestelltes $-Zeichen definiert.
- Zeichenketten sind in einfache Hochkommas einzuschließen (`'TEXT'`).
- Für Kommentare kann anstatt der herkömmlichen geschweiften Klammern auch `(*` bzw. `*)` benutzt werden.
- Der Compiler unterscheidet bei den reservierten Worten sowie vordefinierten Standardbezeichnern für Funktionen und Prozeduren nicht zwischen Groß- und Kleinbuchstaben. Bei selbst definierten Namen von Konstanten, Variablen, Prozeduren und Funktionen ist das im Gegensatz dazu sehr wohl der Fall!

B2: Grundgerüst

Ein AC1-PASCAL-Programm ist wie folgt strukturiert:

`PROGRAM` *`programmname`*`;`	Kopf (optional)

`LABEL ...`
`CONST ...`
`VAR ...`
`PROCEDURE …/FUNCTION...`
	Deklarationen (auch in Prozeduren und Funktionen) nur in dieser Reihenfolge!
	1. Markendeklaration
	2. Konstantendeklaration
	3. Variablendeklaration
	4. Prozedur-/ Funktionsdeklaration

`BEGIN`	Hauptprogramm
…	...
`END.`	Programmende

B3: Variablengültigkeit

- Auf eine Variable kann nur in dem Block zugegriffen werden, der die Deklaration enthält.
- Am Programmanfang deklarierte Variablen sind global, d.h. sie gelten in allen Blöcken. In einem untergeordneten Block deklarierte ("lokale") Variablen sind in einem übergeordneten Block "unsichtbar".
- Beim Eintreten in einen Block werden alle im Block deklarierten Variablen gelöscht, d.h. reelle und ganze Zahlen nehmen den Wert 0 an, boolesche Variablen nehmen den Wert ***false*** an und Zeichenfolgen nehmen den Wert ***empty*** an.

B4: Reservierte Worte

AND	**EXTERNAL**	**OTHERS**
ARRAY	**FOR**	**PROCEDURE**
BEGIN	**FUNCTION**	**PROGRAM**
BOOLEAN	**GOTO**	**REAL**
CASE	**IF**	**REPEAT**
CODE	**INIT**	**SHIFT**
DIV	**INTEGER**	**STRING**
DO	**LABEL**	**THEN**
DOWNTO	**MOD**	**TO**
ELSE	**NOT**	**UNTIL**
END	**OF**	**VAR**
EXOR	**OR**	**WHILE**

B5: Vordefinierte Standardbezeichner (Funktionen/Prozeduren)

abs	**left**	**read**
addr	**ln**	**readln**
arctan	**load**	**right**
call	**maxint**	**round**
chr	**mem**	**save**
concat	**mid**	**sin**
cos	**odd**	**sound**
empty	**ord**	**sqr**
exp	**out**	**sqrt**
false	**pi**	**succ**
frac	**plot**	**true**
inp	**point**	**trunc**
int	**pred**	**write**
keyboard	**random**	**writeln**

load/save: Laden und Speichern von Arrays ist aktuell nicht enthalten.

B6: Operatoren und deren Rangfolge

- Der NOT-Operator hat die höchste Priorität,
- gefolgt von den Multiplikationsoperatoren (* / DIV MOD und SHIFT),
- den Additionsoperatoren (+ - OR EXOR)
- und schließlich (mit dem niedrigsten Vorrang) die Vergleichsoperatoren (= <> > < >= <=)

Einen Potenzoperator wie in BASIC (^) gibt es nicht. Für ganze Zahlen kann die Operation auf Multiplikation und Division zurückgeführt werden. Für REAL-Zahlen (natürlich auch bei ganzen Zahlen möglich) nimmt man z.B.

x hoch y => **exp(y*ln(x)).**

B7: Syntaxbeispiele

	Bedeutung	Syntaxbeispiel	Bemerkungen
abs	Betrag	A:=abs(B)	B: Real/Integer A: wie B
addr	Speicheradresse Variable	A:=addr(B)	A: Integer
AND	logisches UND	A:=B AND C	A,B,C: Integer/Boolean
arctan	Arcustangens	A:=arctan(B)	A/B: Real, B im Bogenmaß
ARRAY… OF…	Datentyp "Feld"	VAR X: ARRAY[1..10] OF REAL	ab Index 0 möglich

**BEGIN
…
END**
	"Klammer" für Anweisungsblock	BEGIN
		…
		END;
			hinter letztem END
			steht '.'

BOOLEAN	Datentyp "logisch"	VAR X: BOOLEAN;	
call	MC-Routine aufrufen	call(A)	A: Integer

**CASE … OF
…
OTHERS:**
	Verzweigung	CASE A OF
		  1: WRITE('1')
		  …
		OTHERS: WRITE('sonst')
			A: Integer/Boolean/String,
			nur =, kein > oder <

chr	Zahl zu String	A:=chr(B)	A: String, B=Integer
CODE	Inline-Assembler	CODE $C9	in Prozedur oder Funktion
concat	String verknüpfen	A:=concat(B,C)	A/B/C: String
cos	Cosinus	A:=cos(B)	B: Real/Integer, A: Real B im Bogenmaß
DIV	Ganzzahldivision	A:=B DIV C	A,B,C: Integer
empty	Leerstring	A:=empty	A: String
EXOR	logisches XOR	A:=B EXOR C	A,B,C: Integer
exp	Exponentialfunktion e^x	A=EXP(B)	A:Integer oder Real, B:Real
EXTERNAL	Definition MC-Routine	PROCEDURE X; EXTERNAL Y;	Y=MC-Adresse im Speicher
false	logisch falsch (0)		Boolean
FOR … (DOWN)TO … DO	Schleife	FOR A:=0 (DOWN)TO B DO	TO=aufwärts, DOWNTO=abwärts
frac	Nachkommaanteil	A:=frac(B)	A/B: Real Vorzeichen bleibt erhalten
FUNCTION	Funktionsdeklaration	FUNCTION X(Y:INTEGER): REAL;	Angabe Datentyp bei Argument und Ergebnis
GOTO	Sprung zu Marke		
IF… THEN … ELSE …	Bedingung	IF A THEN ... ELSE ...	A: Boolean
ln	natürlicher Logarithmus	A=ln(B)	B: Real/Integer, A: Real
INIT	Array füllen	INIT X TO A,B,C…	X/A/B/C: gleicher Datentyp, füllen ab Element 1
inp	Portabfrage	A:=inp(B)	A,B: Integer (0...255)
int	Ganzzahliger Anteil	A:=int(B)	B: Real/Integer, A: Real
INTEGER	Datentyp Ganzzahl	VAR X: INTEGER;	
keyboard	Tastaturabfrage	A:=KEYBOARD	A: Integer (keine T. → 0) sonst Tastencode
LABEL	Deklaration Marke	LABEL M1;	
left	linker Stringteil	A:=left(B,C)	A,B: String, C:Integer
length	Stringlänge	A:=len(B);	B: String, A: Integer
maxint	Konstante	32767	
mem	Speicherzugriff	A:=mem[$17FF] mem[$17FF]=$0D	lesend, A: INTEGER schreibend
mid	Mittelteil aus String	A:=mid(B,C,D) A:=mid(B,C)	A,B: String C (ab…) Integer D (soviel) Integer ohne D: ab...bis Ende
MOD	Modulo-Operator (= Rest bei Ganzzahldivision)	A:=B MOD C	A,B,C: Integer

NOT	logisches NICHT	A:=NOT B	A,B: Integer, Boolean
odd	Test ob geradzahlig	A:=odd(B)	B: Integer, A: Boolean A=true, wenn geradzahlig
OR	logisches ODER	A:=B OR C	A,B,C: Integer, Boolean
ord	ASCII-Wert 1. Zeichen	A:=ord(B)	A: Integer, B:String wenn B='', dann A=0
out	Portausgabe	out(A,B)	A,B: Integer (0...255)
pi	Konstante	3.1415...	Real
plot	Pixel setzen	plot(X,Y,M)	X,Y: Koordinaten, M: Modus Beachte Pufferangabe!
point	abfragen ob Pixel gesetzt	A:=point(X,Y)	A: Boolean X=0...127,X:0...63
pred	Vorgänger i-1	A:=pred(B)	A,B: Integer
PROCEDURE	Deklaration	PROCEDURE X (VAR Y:REAL);	auch ohne VAR
PROGRAM	Programmkopf	PROGRAM TEST;	optional!
random	Zufallswert	A:=random A:=random(B)	A: Real, 0...<1 A,B: Integer, B=Max.Wert-1
read	Eingabe	read(A)	A: Integer,Real, String
readln	-"- mit Wagenrücklauf	readln(A)	A: Integer,Real, String
REAL	Datentyp	VAR X: REAL;	
REPEAT **…** **UNTIL …**	Schleife	REPEAT … UNTIL A	
right	rechter Stringteil	A:=right(B,C)	A,B: String, C: Integer
round	Runden Real zu Integer	A:=ROUND(B)	B: Real, A: Integer
screen	Kursorpositionierung	screen(X,Y)	X,Y: Integer
SHIFT	bitweise Verschiebung von B um C Bits	A:=B SHIFT C	A,B,C: Integer C>0 → links C<0 → rechts
sin	Sinus	A:=sin(B)	B: Real/Integer, A: Real B im Bogenmaß
sound	Tonausgabe	SOUND(H,L)	H,L: Integer H=Höhe, L=Länge
sqr	Quadrat	A:=SQR(B)	B: Real/Integer, A:REAL
sqrt	Quadratwurzel	A:=SQRT(B)	B: Real/Integer, A:REAL
STRING	Datentyp Zeichenkette	VAR X: STRING[10]	
succ	Nachfolger i+1	A:=SUCC(B)	A,B: Integer
true	logisch wahr (-1)		Boolean
trunc	ganzzahliger Anteil	A:=trunc(B)	A: Integer, B: Real
VAR	Variablendeklaration		
WHILE … **DO …**	Schleifenkopf	WHILE A DO …	A: Boolean
write	Ausgabe auf Schirm	write(A,B)	Formatierung Anzahl Vor-/Nachkommastellen möglich z.B. A:2:3
writeln	-"- mit Wagenrücklauf	writeln(A,B)	

Die Anweisungen **left**, **right** und **mid** sind reine Funktionen und geben einen Teil der Zeichenkette zurück. Eine Benutzung zum Verändern der Zeichenkette (wie bei Basic) ist nicht möglich.

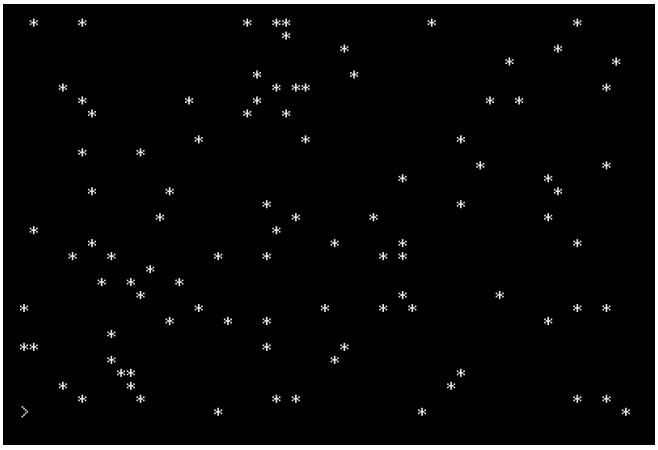
C: Tipps

C1: Zufallsfunktion

Es stehen zwei verschiedene Funktionen zur Verfügung:

`A:=RANDOM;` liefert eine REAL-Zahl im Bereich 0...<1
`A:=RANDOM(B);` liefert eine INTEGER-Zahl im Bereich 0...(B-1)

Selbst bei mehrfachem Aufruf hintereinander ohne nutzerbedingte Pausen ist die Verteilung der Zufallszahlen relativ gut:



```
BEGIN
 WRITE(CHR(12));
 FOR Z:=1 TO 100 DO BEGIN
  SCREEN(RANDOM(64),RANDOM(32));
  WRITE('*');
 END;
END.
```

C2: Datum und Zeit

Steht im AC1 eine RTC zur Verfügung (i.d.R. auf der GIDE-Platine), so ist einfaches Auslesen der Zeit wie folgt möglich:

```
PROGRAM ZEIT; {BENOETIGT GIDE/RTC}
VAR SEK,MIN,STD: INTEGER;
FUNCTION HOLE(NR:INTEGER):INTEGER;
{LIEST UHRENREGISTER NR}
CODE $DD,$2A,$A2,$19,$DD,$46,$FE,$0E,$85,$ED,$78,$E6,$0F,$DD,$77,$FC;
BEGIN
 WRITE(CHR(12));
 IF HOLE(0)=15 THEN WRITELN('KEINE RTC GEFUNDEN') ELSE
 REPEAT
  SCREEN(28,15);           {MITTE BILDSCHIRM}
  SEK:=HOLE(1)*10+HOLE(0); {SEKUNDEN 10ER UND SEKUNDEN 1ER}
  MIN:=HOLE(3)*10+HOLE(2);
  STD:=HOLE(5)*10+HOLE(4);
  WRITELN(STD:2,':',MIN:2,':',SEK:2);
 UNTIL KEYBOARD;
END.
```

Der Funktion "HOLE" wird das auszulesende Uhrenregister übergeben. Sie arbeitet so:

```
DD 2A A2 19       LD    IX,WSP      ;Parameterstack
DD 46 FE          LD    B,(IX-2)    ;Nr. Uhrenregister holen (wurde übergeben)
0E 85             LD    C,85h       ;Basisadresse
ED 78             IN    A,(C)       ;RTC-Register lesen
E6 0F             AND   A,0Fh       ;nur NWT
DD 77 FC          LD    (IX-4),A    ;Rückgabewert in Parameterstack schreiben
```

Die Nummern der Uhrenregister sind der GIDE/RTC-Beschreibung zu entnehmen (z.B. 00=Sekunden-Einer,...0Bh=Jahr-Zehner).

Benutzung z.B. für Zeitmessung in Benchmark-Tests.

C3: Grafik (Linie, Kreis und Ellipse)

```
PROCEDURE LINE(X0,Y0,X1,Y1,Z1: INTEGER);

VAR I,DX,DY,D,AX,AY,BX,BY: INTEGER;
BEGIN
 DX:=X1-X0; DY:=Y1-Y0; BX:=0; AY:=0; AX:=1; BY:=1;
 IF DX<0 THEN BEGIN AX:=-1; DX:=-DX END;
 IF DY<0 THEN BEGIN BY:=-1; DY:=-DY END;
 IF DX<DY THEN BEGIN
   I:=DX; DX:=DY; DY:=I; BX:=AX; AX:=0; AY:=BY; BY:=0;
 END;
 D:=DX SHIFT -1;
 FOR I:=0 TO DX DO BEGIN
  PLOT(X0,Y0,Z1);
  X0:=X0+AX; Y0:=Y0+AY; D:=D+DY;
  IF D>DX THEN BEGIN D:=D-DX; X0:=X0+BX; Y0:=Y0+BY;
  END;
 END;
END;
```

```
X0,Y0=Anfangspunkt
Y1,Y1=Endpunkt
X<=0<=127
y<=0<=63
0,0=links oben

Z1=Modus:
0=reset,1=set,2=invert
```

```
PROGRAM KREIS;
VAR X,Y: INTEGER;
    I: REAL;
BEGIN
   WHILE I<PI/2 DO BEGIN
   X:= 63+ ROUND(42*SIN(I));Y:= 31+ ROUND(31*COS(I));
   PLOT(X,Y,1);   PLOT(128-X,Y,1);
   PLOT(X,64-Y,1);PLOT(128-X,64-Y,1);
   I:=I+0.025;
  END;
END.
```

```
einfach, aber langsam!

Faktoren 42 und 31
bestimmen, wie groß und
wie "rund" der Kreis wird
```

```
PROGRAM KREIS;

PROCEDURE CIRCLE(X,Y,R: INTEGER);
VAR   X1,X2,Y1,Y2,R2,D: INTEGER;
BEGIN
  Y2:=0;D:=ROUND(R*0.7);
  X1:=R;X2:=R*R;
  R2:=X2+D;
  PLOT(X,Y,1);
  FOR Y1:=0 TO D DO  BEGIN
    IF Y2+X2>R2 THEN BEGIN
      X2:= X2-X1-X1+1;X1:= X1-1;
    END;
    PLOT(X-X1,Y-Y1,1);PLOT(X-X1,Y+Y1,1);
    PLOT(X+X1,Y-Y1,1);PLOT(X+X1,Y+Y1,1);
    PLOT(X-Y1,Y-X1,1);PLOT(X-Y1,Y+X1,1);
    PLOT(X+Y1,Y-X1,1);PLOT(X+Y1,Y+X1,1);
    Y2:=Y2+Y1+Y1+1;
  END;
END;
BEGIN
 CIRCLE(64,31,16);
END.
```

```
schneller, aber am AC1
wird der Kreis wegen
seiner nicht
quadratischen Pixel
nicht rund...
```

```
PROGRAM KREIS;
CONST XFMI=1;XFMA=127;YFMI=1;YFMA=63;
VAR   R: INTEGER;
PROCEDURE KREIS(X0,Y0,R,Z1:INTEGER);
VAR I,X,Y,S: INTEGER;
  NEGX,NEGY,SWAP: BOOLEAN;
PROCEDURE KPLOT(X,Y: INTEGER);
BEGIN
  IF SWAP THEN BEGIN I:=X;X:=Y;Y:=I END;
  IF NEGY THEN Y:=-Y;
  IF NEGX THEN X:=-X;
  X:=X+X0;Y:=((Y*6) DIV 8)+Y0;
  IF ((X>XFMI) AND (X<XFMA) AND (Y>YFMI) AND (Y<YFMA)) THEN
  PLOT (X,Y,Z1);
END;
BEGIN
  FOR NEGX:=TRUE TO FALSE DO
    FOR NEGY:=TRUE TO FALSE DO
      FOR SWAP:=TRUE TO FALSE DO BEGIN
        X:=R;Y:=0;S:=-R;
        REPEAT
          KPLOT(X,Y);S:=S+Y+Y+1;Y:=Y+1;
          IF S>0 THEN BEGIN S:=S-X-X+2;X:=X-1 END;
        UNTIL Y>X;
      END;
END;
BEGIN
 WRITE(CHR(12)); KREIS(63,31,10,1);
END.
```

```
am AC1 auch rund, aber
schnell nur bei kleinen
Radien (1...15)

bei Randüberschreitung
wird abgeschnitten
```

```
PROCEDURE PLOT4(X,Y:INTEGER);
BEGIN
   PLOT(XM+X,YM+Y,1);PLOT(XM+X,YM-Y,1);
   PLOT(XM-X,YM-Y,1);PLOT(XM-X,YM+Y,1);
END;

PROCEDURE ELLIPSE(RX,RY:INTEGER);
VAR X,Y,XCH,YCH,ERR,TWOA,TWOB,STOPX,STOPY: INTEGER;
BEGIN
   TWOA:=2*RX*RX;TWOB:=2*RY*RY;
   X:=RX;Y:=0;XCH:=RY*RY*(1-2*RX);YCH:=RX*RX;
   ERR:=0;STOPX:=TWOB*RX;STOPY:=0;
   WHILE (STOPX>=STOPY) DO BEGIN
      PLOT4(X,Y);Y:=Y+1;STOPY:=STOPY+TWOA;
      ERR:=ERR+YCH;YCH:=YCH+TWOA;
      IF ((2*ERR + XCH)>0) THEN BEGIN
         X:=X-1;STOPX:=STOPX-TWOB;
         ERR:=ERR+XCH;XCH:=XCH+TWOB;
      END;
   END;
   X:=0;Y:=RY;XCH:=RY*RY;YCH:=RX*RX*(1-2*RY);
   ERR:=0;STOPX:=0;STOPY:=TWOA*RY;
   WHILE (STOPX<=STOPY) DO BEGIN
      PLOT4(X,Y);X:=X+1;STOPX:=STOPX+TWOB;
      ERR:=ERR+XCH;XCH:=XCH+TWOB;
      IF ((2*ERR+YCH) >0 ) THEN BEGIN
         Y:=Y-1;STOPY:=STOPY-TWOA;
         ERR:=ERR+YCH;YCH:=YCH+TWOA;
      END;
   END;
END;
```

```
nötig wäre LONGINTEGER,
so können nur Ellipsen/
Kreise bis zu einem
Radius von ca. 25 Pixeln
dargestellt werden,
ansonsten Zahlenbereichs-
überschreitung und daraus
resultierende "Effekte"

im Hauptprogramm:
VAR XM,YM: INTEGER;
{KREISMITTELPUNKT}
    RX,RY: INTEGER;
{RADIEN}

Aufruf:
XM:=...; XM:=...;
ELLIPSE(RX,ROUND(RX/1.35)

Mit Faktor 1.35 ergibt
sich am AC1 etwa ein
"runder" Kreis.
Bei Benutzung von
Monitoren mit festem
16:9-Seitenverhältnis ist
das entsprechend
abzuändern.
```

C4: Sound

```
PROGRAM SOUNDDEMO;
VAR H,L,I,TEMPO: INTEGER;
    TON: ARRAY [1..58] OF INTEGER;
BEGIN
  WRITELN('SPIELE SOUND...');TEMPO:=5;
  INIT TON TO
  {TOENE AUS GS-BASIC-POGRAMM MIT 'SOUND X,Y'
   1.WERT=TONHOEHE
   2.WERT=TONLAENGE}
       238,2,200,1,200,4,238,1,224,2,200,1,
       200,4,200,1,158,4,178,1,200,2,224,1,
       178,3,200,4,200,1,149,2,200,1,200,4,
       238,1,238,2,224,1,178,4,178,1,200,4,
       224,1,238,2,224,1,224,3,238,4;
  FOR I:=1 TO 58 DO
     BEGIN
        H:=TON[I];L:=TEMPO*TON[I+1];
        SOUND(H,L);I:=I+1;
     END;
END.
```

Tontabellen aus GS-BASIC können benutzt werden.

Sofern in der BASIC-Vorlage keine ganzzahligen Werte für die Tonlängen benutzt werden, sind diese zu verdoppeln und daraus resultierend das Tempo anzupassen.

C5: Bildschirminhalt sichern/laden

Mit nachfolgenden Prozeduren kann der Bildschirminhalt als Datei per USB gesichert bzw. zurückgeladen werden. An die jeweilige Prozedur ist lediglich der Dateiname als String zu übergeben, Bedingungen siehe oben (8+3, keine Sonderzeichen).
Beispiel: `SAVESCR('TEST.DAT');`
Die Dateierweiterung 'DAT' ist frei wählbar, sollte aber um Verwechslungen zu vermeiden weder PAS noch BIN lauten.

```
PROCEDURE SAVESCR(S:STRING[13]);
VAR I: INTEGER;
BEGIN
   FOR I:=1 TO LENGTH(S) DO MEM[$19CF+I]:=ORD(MID(S,I,1));
   MEM[$19CF+I]:=$0D;CALL($200C); {SAVE}
END;
```

```
PROCEDURE LOADSCR(S:STRING[13]);
VAR I: INTEGER;
BEGIN
   FOR I:=1 TO LENGTH(S) DO MEM[$19CF+I]:=ORD(MID(S,I,1));
   MEM[$19CF+I]:=$0D;CALL($200F); {LOAD}
END;
```

S: DATEINAME (8.3)

$19CF: Eingabepuffer -1

Hinweise:

- Existiert die bei LOADSCR angegebene Datei nicht, so meldet der USB-Treiber lediglich "Nicht gefunden" und das Programm wird danach fortgesetzt.
- Existiert bei SAVESCR eine gleichnamige Datei bereits, so wird sie ohne Rückfrage überschrieben! Soll nicht immer nur das letzte Bildschirmbild gesichert werden, so ist eine geeignete Vergabemethode für den Dateinamen (z.B. mit fortlaufendem Index) zu wählen.
- Infolge der rückläufigen BWS-Organisation und der aufsteigenden Adressierung beim Laden von USB erfolgt der Bildaufbau von unten nach oben.
- Sowohl bei LOADSCR als auch bei SAVESCR erfolgt in allen Fällen zunächst eine automatische Neuinitialisierung der USB-Schnittstelle. Das bringt zwar einen kleinen Zeitverzug, gewährleistet aber die Aktualität.
- Ist kein USB-Stick angeschlossen oder fehlt die USB-Hardware ganz, so ergeht die Meldung "`ND USB-Fehler!`", wenn LOADSCR oder SAVESCR ausgeführt werden soll. Das Programm wird anschließend fortgesetzt.

C6: Allgemeine Tipps

C6.1 Warteschleife:

Wird in einem Programm eine länger andauernde Warteschleife durchlaufen, so reagiert während dieser Zeit die Tastatur nicht. Folgende Lösung vermeidet dies, indem die Wartezeit bei Tastendruck sofort beendet wird:

```
FOR I:=1 TO 200 DO IF KEYBOARD THEN I:=200;
```

Der Schleifenendwert (hier 200) ist je nach Erfordernis (Wartezeit) auszuwählen.

C6.2 Datenablage:

Eine DATA-Anweisung wie in BASIC gibt es nicht, dafür aber das Array (String-, Real- oder Integer-Werte). Der Zugriff kann entweder mit dem Feldindex oder über berechnete Adresse erfolgen:

```
VAR  FELD[1..100] OF INTEGER;{100 Integer-Werte = 200 Bytes]
WRITE(FELD[2]);              {druckt Inhalt Element 2 aus}
X:=ADDR(FELD[3]);            {liefert Adresse von Element 3}
```

Sollen lediglich 8-Bit-Daten verwendet werden, so braucht das Array nur halb so groß deklariert zu werden. Der Zugriff für das Lesen/Schreiben eines Bytes erfolgt dann jedoch nicht mit FELD[n] (es würden da immer 2 Bytes gelesen) sondern nur über die Adresse:

```
X:=MEM[ADDR(FELD[4])];       {liest das 4. Byte}
```

C6.3 Problem bei INTEGER-Rechnungen

Problem: Integerzahlen addieren (oder in einem Ausdruck anderweitig verknüpfen), deren Ergebnis > 32767 (MAXINT) ist. Dazu wird ein Real benötigt, soweit klar:

Beispiel:

falsch:	richtig wäre z.B.:

`VAR T: REAL;`
`  I: INTEGER;`
`BEGIN`
`  I:=30000;`
`  T:=I+I;`
 	
`  WRITELN(T);`
`END.`
	`VAR T: REAL;`
	`  I: INTEGER;`
	`BEGIN`
	`  I:=30000;`
	`  T:=I;`
	`  T:=T+I;`
	`  WRITELN(T);`
	`END.`

Erwartet: T= 6.0000000000E+04	
Ist: T=-5.5360000000E+03	Ist: T= 6.0000000000E+04

Ursache: Im falschen Beispiel wird zunächst eine Addition zweier Integerwerte durchgeführt und dann das Ergebnis in eine REAL-Variable geschrieben. Der Compiler wertet aber leider beim Umgang mit INTEGER das Auftreten von Überlauf oder Unterlauf nicht aus. So wird dem REAL ein falsches Rechenergebnis zugewiesen.

Lösung: Zuerst erst nur einen Integer-Wert nach Real übernehmen und die weiteren Integers mit der Real-Variablen verknüpfen:

```
T:=I;
T:=T+I;
```

Generell: Ausdrücke mit INTEGER-Zahlen (siehe '+' oben) immer auf Grenzwerte prüfen, d.h. ob deren Ergebnis MAXINT überschreiten kann. Wenn das möglich ist, dann Rechenmethode entsprechend wählen oder gleich REAL nehmen!

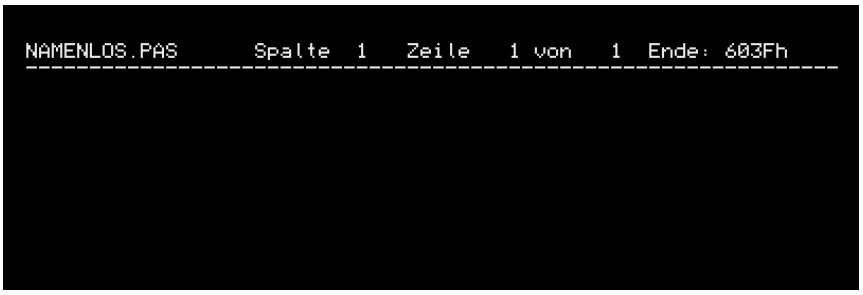
D: Benchmarktest

Einer der Benchmarktests aus den NASCOM-Unterlagen, angepasst für AC1. Der eigentliche Test ist gelb markiert. Ist am AC1 eine RTC verfügbar, so wird die Laufzeit automatisch gemessen. Andernfalls muss man eine herkömmliche Uhr benutzen...

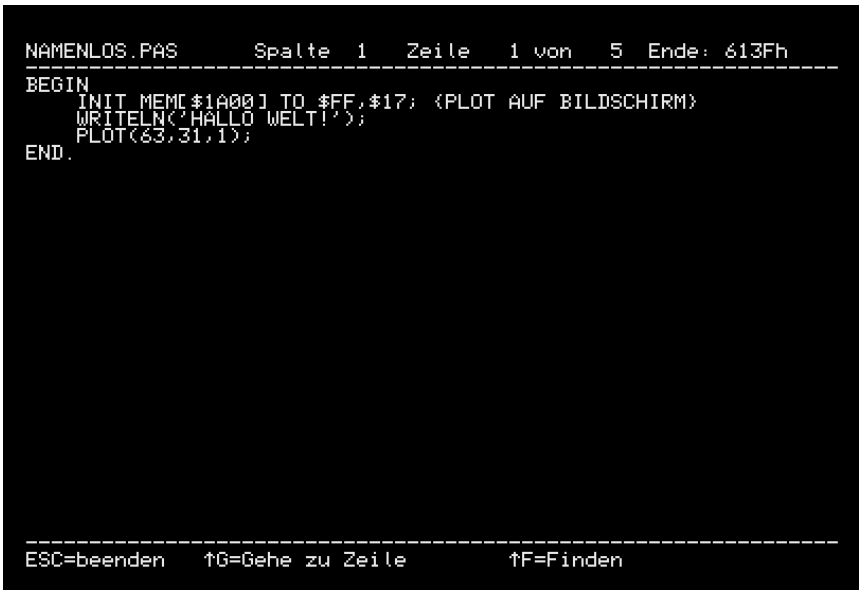
```
PROGRAM BENCHMARK VECTOR;
VAR I: INTEGER;
    T0,T1,T2: REAL;

PROCEDURE VEKTOR;
VAR I,J,K: INTEGER;
    MATRIX: ARRAY[0..10] OF INTEGER;
BEGIN
    MATRIX[0]:=1;
    FOR K:=1 TO 10000 DO
    FOR J:=1 TO 10 DO MATRIX[J]:=MATRIX[J-1];
END;

FUNCTION HOLE(WERT:INTEGER):INTEGER;
CODE $DD,$2A,$A2,$19,$DD,$46,$FE,$0E,$85,$ED,$78,$E6,$0F,$DD,$77,$FC;

PROCEDURE ZEIT; {BENOETIGT GIDE/RTC}
VAR SEK,MIN,STD: INTEGER;
BEGIN
    SEK:=HOLE(1)*10+HOLE(0);
    MIN:=HOLE(3)*10+HOLE(2);
    STD:=HOLE(5)*10+HOLE(4);
    IF STD<10 THEN WRITE('0');
    WRITE(STD,':');
    IF MIN<10 THEN WRITE('0');
    WRITE(MIN,':');
    IF SEK<10 THEN WRITE('0');
    WRITE(SEK);
    T0:=STD;T0:=T0*3600+60*MIN+SEK; {ERMITTLUNG DER TICKS}
END;

BEGIN
   WRITE(CHR(12));
   WRITELN('BENCHMARK-TEST "VECTOR", AUF NASCOM 4MHZ: 62,2 SEK')
   WRITELN;WRITELN('AUF DIESEM AC1:');
   IF HOLE(0)=15 THEN                {TEST OB RTC VERFUEGBAR}
      BEGIN
         WRITELN('KEINE RTC GEFUNDEN, ZEITMESSUNG PER HAND!');
         WRITELN('START MIT BELIEBIGER TASTE...');
         REPEAT UNTIL KEYBOARD;
         WRITELN('TEST LAEUFT...');
         VEKTOR;
         WRITELN('FERTIG');
      END
   ELSE
      BEGIN
         ZEIT;T1:=T0;
         WRITELN(' -> START:',T1:8:0,' SEKUNDEN SEIT MITTERNACHT.');
         WRITELN('TEST LAEUFT...');
         VEKTOR;
         ZEIT;T2:=T0;
         WRITELN(' -> ENDE :',T2:8:0);
         WRITELN('DAUER:',T2-T1:5:0,' SEKUNDEN');
      END;
END.
```

E: Komplettbeispiel "Hallo-Welt"

<table>
<tr><td>1.</td><td>AC1-Pascal
auf $2000 laden
und
auf $2000 starten</td><td> 
</td></tr>
<tr><td>2.</td><td>'E' = Editor aufrufen

Leere Seite wird
angezeigt.</td><td> 
</td></tr>
<tr><td>3.</td><td>Quelltext schreiben:

Es soll der Text
'Hallo Welt'
ausgegeben
und ein Blockgrafikpixel
gesetzt werden.

Bei späterem Laden
erscheint statt
'NAMENLOS.PAS'
der Dateiname.</td><td> 
</td></tr>
<tr><td>4.</td><td>'ESC' = Editor verlassen</td><td>Ein leerer Schirm erscheint.</td></tr>
<tr><td rowspan="2">5.</td><td>'S' = speichern
HALLO eintragen</td><td> 
</td></tr>
<tr><td colspan="2">Das File HALLO.PAS kann nun jederzeit von USB geladen und weiter bearbeitet werden.</td></tr>
<tr><td>6.</td><td>'C' = Compilieren</td><td rowspan="2"> 
</td></tr>
<tr><td>7.</td><td>'R' = RAM anzeigen</td></tr>
<tr><td>8.</td><td>'T' = Testlauf starten
Ausschrift Text erfolgt,</td><td> 
</td></tr>
<tr><td>9.</td><td>'F' = eigenständiges
File erzeugen</td><td> 
</td></tr>
<tr><td>10.</td><td colspan="2">Das File HALLO.BIN kann anschließend jederzeit (z.B. per DVU) geladen und ausgeführt werden, ohne dass AC1-PASCAL nötig ist.</td></tr>
</table>

Analyse des erzeugten Codes

Die nachfolgend angeführten Adressen gelten für den o.a. Quelltext sowie die aktuelle Programm-Version und müssen bei eigenen Experimenten nicht identisch sein!

$2000..352B = Laufzeitmodul (auf $2000 steht ein Sprungbefehl zu Adresse $352C)
$352C..3567 = das eigentliche Anwenderprogramm:

```
352C  CD 49 30    CALL  $3049       ;der PASCAL-Starter
352F  57 35       DEFW  $3557       ;Daten: Programmende+1
3531  7F FF       DEFW  $FF7F       ;Daten: gewähltes RAM-Ende
3533  00 60       DEFW  $6000       ;Daten: Beginn originaler Quelltext
3535  03          DEFB  3           ;Daten: Rückkehrmodus
;
3536  21 00 1A    LD    HL,$1A00    ;Arbeitszelle für PLOT-Bereich
3539  EB          EX    DE,HL
353A  21 45 35    LD    HL,$3545
353D  01 02 00    LD    BC,0002
3540  ED B0       LDIR              ;setzen auf Bildschirmanfang
3542  C3 47 35    JP    $3547
;
3545  FF 17       DEFW  $17FF       ;Daten: Bildschirmanfang
;
3547  DF          RST   $18         ;Folgetext ausgeben
3548              DEFM  'HALLO WELT!'     ;auszugebende Daten
3553  80          DEFB  $80         ;Textende
3554  3E 0D       LD    A,$0D       ;neue Zeile
3556  D7          RST   $10         ;ausgeben
3557  21 3F 00    LD    HL,$003F    ;X-Koordinate 63
355A  E5          PUSH  HL
355B  21 1F 00    LD    HL,$001F    ;Y-Koordinate 31
355E  E5          PUSH  HL
355F  21 01 00    LD    HL,$0001    ;Modus "SET"
3562  CD 82 2F    CALL  $2F82       ;UP Pixel setzen
3565  C3 FD 07    JP    $07FD       ;zurück zum Monitor
```

Das eigentliche Programm ist hier nur 60 Bytes lang. Um es auszuführen, kommt aber (wie bei einigen anderen PASCAL-Versionen) noch das komplette Laufzeitmodul hinzu, obwohl daraus nur das Gelbmarkierte (Starter und Pixel-Set-Routine) benötigt wird.

Der Starter ist auch im Compile/Test-Modus vorhanden und gewährleistet eine Überwachung der Ausführung. Er würde z.B. eine Fehlermeldung erzeugen und das Programm abbrechen, wenn während der Laufzeit die in den Kopfdaten enthaltenen Grenzen überschritten werden.

Als eigenständiges ausführbares Programm funktioniert das zwar, ist jedoch bei solch kurzen Programmen wenig effektiv und alles andere als "optimierter Code"…