AC1-Pascal

Kurzinfo

Version 0.5

Stand 19.10.2019

Bei dieser PASCAL-Version handelt es sich um ein direkt im Monitor-Modus des AC1 lad- und ausführbares Programm. Es wird kein CP/M benötigt! Das Vorbild und die Compiler-Engine lieferte das "BLS-(NASCOM)"-Pascal.

Speicheraufteilung:

$1900...$1FFF	Arbeitszellen und Programmstack
$2000...($BFFF)	Quelltext und Code
$C000...($FFFF)	Laufzeitmodul, Compiler, Editor, Verwaltung

Version 0.5:
Die Arbeitsweise ist mit AC1-(GS-)BASIC vergleichbar: es lassen sich nur Quelltexte erstellen und nach Compilierung ausführen. Ein Demoprogramm ist enthalten und kann mit "D" geladen werden. Quelltexte können per USB gespeichert und geladen werden sofern AC1-Monitor11 benutzt wird.

Plus:

- nur ca. 12kB "schlankes" Programm
- Voll-Schirm-Editor
- Es werden fast alle Eigenschaften des AC1 unterstützt, z.B.:
  - alle Steuerzeichen verfügbar (z.B. invers und Sound)
  - Blockgrafik per PLOT (Setzen/rücksetzen/invertieren von Pixel)
  - direkter Zugriff auf den Bildwiederholspeicher (z.B. für Pseudografikzeichen)
-

Minus: Es fehlt noch vieles, vor allem:

- Suchfunktion,
- Druckfunktion,...

Inhaltsverzeichnis

Verwaltung

Nach Start von AC1-Pascal mit J C000 und der Anzeige des Bereitschaftszeichens (>) stehen folgende Kommandos zur Verfügung:

(C)OMPILE	Aufruf des Compilers
(D)EMO	Demo-Quelltext
(E)DIT	Aufruf des Editors
(L)OAD	Laden eines Quelltextes [1])
(M)EMORY	Anzeige des belegten Speichers
(Q)UIT	Beenden von AC1-PASCAL
(R)UN	Ausführen (sofern zuvor noch nicht compiliert, erfolgt das Compilieren automatisch)
(S)AVE	Sichern eines Quelltextes [1])
(Z)AP	Löschen des aktuellen Programms
<ENTER>	<ENTER> ohne einen Kommandobuchstaben bewirkt das Löschen des aktuellen Bildschirminhalts

[1]) per USB, erfordert Monitor11!

Es ist ausreichend, den jeweiligen Kommandobuchstaben einzugeben und mit <ET> zu quittieren. Eine Ausnahme bildet das Löschkommando "ZAP", welches aus Sicherheitsgründen auszuschreiben ist.

Parameter sind nicht erforderlich/wirksam.

Nach Verlassen mit Q ist ein Warmstart mit J C003 möglich. Es kann dann mit dem vorhandenen Quelltext weitergearbeitet werden, sofern der Bereich nicht überschrieben wurde. Beim Warmstart erfolgt aktuell keine Zeichensatz-Umschaltung, also ggf. zuvor mit STRG+Z im Monitor erst umschalten.

Eckdaten der PASCAL-Version und AC1-Spezifik

Allgemeines

Ein solch kleiner Compilerkern wie beim NASCOM-Pascal erfordert z.B. gegenüber Turbo-Pascal auch Abstriche in den Funktionalitäten. Es fehlen beispielsweise die Datentypen SET und RECORD und die zugehörigen Anweisungen. Dafür bietet er einige Erweiterungen in Richtung Grafik, Stringverarbeitung, direkten Speicherzugriff und "Sound", d.h. er kann alles, was man von (AC1-)Basic kennt und mehr.

Der Funktionsumfang ist durch die Schlüsselworte in Anlage A gekennzeichnet.

Als Datentypen stehen zur Verfügung:

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

Real-Genauigkeit und -Bereiche: 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
```

Konventionen:

- 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.

Tastatur-Eingaben

Zahlen/ Zeichenketten	READ(x) bzw. READLN(x)	- liest Wert ja nach Datentyp von x ein. - unzulässige Eingabe: Anzeige wird gelöscht, dann Neueingabe - ^C bricht Eingabe ab und gibt (wie "nur Enter") 0 bzw. einen Leerstring zurück
Einzelzeichen	KEYBOARD	- liest Tastatur, wartet nicht - Taste: ASCII-Wert, keine Taste: Rückgabewert 0

Bildschirmausgaben

Ist ein Zeichensatz mit eckigen Klammern vorhanden (AC1-Standard, "unten" im EPROM), so wird dieser nach dem Start automatisch aktiviert und nach Beenden wieder deaktiviert. Damit erscheinen in AC1-Pascal keine deutschen Umlaute, jedoch die benötigten eckigen und geschweiften Klammern.

Bei der Standard-Ausgabe mit WRITE(x) bzw. WRITELN(x) stehen per chr-Funktion auch sämtliche Steuerfunktionen (auch INVERS, soweit Hardware vorhanden) des AC1 zur Verfügung. `WRITE(chr(12));` löscht z.B. den Bildschirm. AC1-Blockgrafik oder Pseudografik (Codes >128) ist mit WRITE nicht möglich.

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

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

Ist die Koordinatenangabe ungültig (keine Ganzzahl oder außerhalb der Bereiche), so 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 FALSE (0).

Zu den PLOT-/POINT-Anweisungen gibt es eine Erweiterung. Standardmäßig wirken sie auf den Bildschirmbereich (Adressen $17FFh...$1000). Um Zeichnungen unsichtbar im Hintergrund vornehmen zu können, kann diese Adresse auf einen anderen Speicherbereich umgelenkt werden. Dazu dient der Wert auf Speicherplatz $2006/2007 (Version ab $2000) bzw. $C006/C007 (Version ab $C000). Anzugeben ist jeweils die höchste Speicheradresse. Beispiel:

```
mem[$C006]:=$FF;mem[$C007]:=$7F;      {Zeichen in einen Puffer $7800...$7FFF und}
mem[$C006]:=$FF;mem[$C007]:=$1F;      {Rückstellen auf den Standardwert Bildschirm}.
```

Voraussetzung ist natürlich, dass sich der Compiler im RAM befindet…

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

```
PROCEDURE COPY;
CODE $21,$FF,$7F,$11,$FF,$17,$01,$00,$08,$ED,$B8;
```

bzw. löschen:

```
PROCEDURE CLEAR;
CODE $21,$FF,$7F,$36,$20,$54,$5D,$1B,$01,$00,$08,$ED,$B8;
```

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 kann auch die Direktpositionierung des Monitors benutzt werden: `WRITE(CHR(14),'1005NAME:';` schreibt z.B. die Zeichenkette 'NAME' ab Spalte 10 in Zeile 5.

Inline-Assembler

Mit der CALL-Anweisung und der EXTERNAL-Option einer Prozedur oder Funktion lassen sich Maschinenprogramme ausführen, die zuvor auf geeignete Weise in den Speicher gebracht worden sind.

Solche (kleinen) 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-Anweisung mit direkter Angabe der hexadezimaler Codebytes. Beispiel:

```
PROCEDURE CLS;
      CODE $3E,$0C,$D7;
BEGIN
      CLS;
      WRITELN('HALLO WELT');
END.
```

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

Löscht den Bildschirm
```

Es ist auch möglich, Parameter an die Maschinencode-Routine zu übergeben, wobei nur 8Bit- oder 16Bit-Werte (Adressen oder Datenbytes) als Parameter zulässig sind, z.B.:
`PROCEDURE TEST(X1,X2: INTEGER);`

Das Auslesen der Parameter erfolgt mit Hilfe des "Workspace-Stackpointers" (WSP) in Systemzelle $1992 [1]. Mit dessen Hilfe lassen sich die Parameter z.B. wie folgt abholen:

```
LD      IX,(1992h)     ;akt. WSP, enthält Zeiger auf Parameterliste
LD      A,(IX-2)       ;→ Parameter X2
...
LD      A,(IX-4)       ;→ Parameter X1
```

Beispiel: Bildschirm komplett mit vorgegebenem Zeichen füllen

```
PROCEDURE MUSTER(X: INTEGER);
CODE $DD,$2A,$92,$19,
     $DD,$7E,$FE,
     $21,$FF,$17,
     $77,
     $54,
     $5D,
     $1B,
     $01,$00,$08,
     $ED,$B8;
BEGIN
     MUSTER($2A);
END.
```

```
LD    IX,(1992h)   ;Zeiger auf Parameter
LD    A,(IX-2)     ;Parameter auslesen
LD    HL,17FFh
LD    (HL),A       ;und verarbeiten
LD    D,H
LD    E,L
DEC   DE
LD    BC,0800h
LDDR

Bildschirm voller Sterne zeichnen...
```

Hinweise:

- Die Code-Spezifikation darf nur voll verschiebliche Programmteile enthalten.
- Bei CODE ist eine abschließende RET-Anweisung ($C9) im Gegensatz zu EXTERNAL nicht erforderlich (aber unschädlich).
- Es empfiehlt sich, das Quellprogramm vor Compilierung/Ausführung zu sichern!

1 In originalen NASCOM-Pascal-Quellen ist das Adresse $0C92.

Editor

Nach Aufruf von "E" erscheint der Quelltexteditor. Im Gegensatz zu den meisten anderen (CP/M-)Pascal-Versionen handelt sich hier um einen "Vollschirm-Editor", der die gleichzeitige Betrachtung der Nachbarzeilen ermöglicht. Dargestellt werden 27 Zeilen. Sind mehr vorhanden, dann automatisches Rollen, wenn der Kursor den oberen bzw. unteren Rand überschreiten möchte. Pro Textzeile können 63 Zeichen eingegeben werden.

Nach Beenden des Editors merkt er sich Schirmausschnitt und Kursorposition und stellt diese 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	POS1	Kursor an Textanfang, ggf. Rollen
04	^D	DEL	Zeichen unter Kursor löschen, Rest rückt auf
05	^E	INS	Leerzeichen an Kursorposition einfügen, Rest rückt nach
06	^F		Kursor an Zeilenanfang
08	^H	<-	Kursor nach links [1])
09	^I	->	Kursor nach rechts [1])
0A	^J	v	Kursor nach unten [2])
0B	^K	^	Kursor nach oben [2])
0C	^L		Leerzeile einfügen
0D	^M	ENTER	Anfang der nächsten Zeile, am Textende wird eine neue Zeile erzeugt
11	^Q	Bild ^	Kursor an Seitenanfang bzw. eine Seite zurück
13	^S		Streichen Zeile
15	^U	Bild v	Kursor 1 Seite weiter (auf letzte Zeile, wenn weniger als 27 Zeilen)
17	^W		Tabulator 4 Zeichen
18	^X		Kursor ans Zeilenende
1A	^Z	END	Gehe zum Textende (> 27 Zeilen ->letzte Zeile allein oben)
1B	^Ä	ESC	Editor verlassen

[1]) nur innerhalb einer Zeile, keine Überschreitung links/rechts
[2]) bei Überschreitung der ersten bzw. letzten Zeile wird der Inhalt gescrollt (sofern Zeilen vorhanden sind)

Hinweise:

- In der aktuellen Version belegt jede Editorzeile 64 Zeichen im Puffer. Zeilen- oder Text-Endemarken sind nicht enthalten. Damit war ein relativ einfacher Editor möglich, er ist aber "Platzfresser" (max. ca. 500 Programmzeilen, je nach Größe des erzeugten Codes)
-

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 (nur die Nummer) mit folgender Bedeutung:

01	Syntaxfehler (z.B. Semikolon fehlt)	40	BEGIN erwartet
02	'=' erwartet	41	THEN fehlt
03	':' erwartet	42	CASE-Selektor muss Ganzzahl oder String sein
04	'[' erwartet	43	OF fehlt in CASE-Anweisung
05	']' erwartet	44	END fehlt
06	'(' erwartet	45	DO fehlt in WHILE-Anweisung
07	')' erwartet	46	Ganzzahl-Variable erwartet
08	',' erwartet	47	TO der DOWNTO fehlt in FOR-Anweisung
09	'.' erwartet	48	DO fehlt in FOR-Anweisung
10	'.." erwartet	49	Marke wurde nicht deklariert
11	':=" erwartet	50	TO fehlt in INIT-Anweisung
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 Wagenrücklaufzeichen
		66	Stringkonstante zu lang
30	Ganzzahl-Konstante erwartet		
31	String-Konstante erwartet	70	Typfehler in Zuweisung oder Parameterliste
32	Real-Konstante erwartet	71	Unbekannter Variablenbezeichner
33	Ganzzahl-Konstante 0<=i<=255 nötig	72	Unbekannter Feldbezeichner
		80	Marke deklariert aber nicht definiert
		99	Unerwartetes Quelltextende

Nach Quittierung per Leertaste erscheint der Editor. Der Kursor wird automatisch etwa an der 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.

Programmausführung

Nach erfolgreicher Compilierung ("OK"-Meldung) befindet sich das ausführbare Programm im Speicher (im Anschluss an den Quellcode). Dieses kann nun mit "R" gestartet werden kann. Es ist auch möglich, auf "C" zu verzichten und sofort mit "R" 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 ist wird das ausführbare Programm gelöscht, sodass beim nächsten "R" zunächst wieder compiliert wird.

Bezogen sich o.a. Fehler auf das Compilieren, so gibt es weitere, die beim Ausführen des Objektcodes auftreten können:

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	natürlicher Logarithmus zu nicht positiver Zahl		
05	reelle Zahl bei Konvertierung zu Integer zu groß	99	Speicherüberlauf, zu wenig RAM

Da in diesen Fällen 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 zum Sichern und Laden

AC1-Pascal erfordert in der aktuellen Version zum Sichern und Laden per USB den **Mon.11** (Ralph Hänsel). Ist dieser nicht vorhanden, stehen die Operationen nicht zur Verfügung.

Der Quelltext wird als ASCII-Datei mit Headersave-Vorspann gesichert, wobei:

- der Dateityp auf "E" eingestellt ist.
- die Dateien die Endung *.P80 tragen und
- der beim Speichern erzeugte Dateiinhalt 64 Zeichen pro Zeile beträgt (keine Endemarken)

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

- max. 8 Zeichen, keine Sonderzeichen, keine Endung angeben
- wird nur <ENTER> betätigt, so erscheint das USB-Inhaltsverzeichnis
- Strg+C bricht die Names-Eingabe ab.

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 meldet sich er Monitor mit der "`Overwrite(J)`"-Abfrage. Bestätigt man diese mit "J", so wird die existierende Datei mit dem aktuellen Quelltext überschrieben, andernfalls abgebrochen. Nach dem Sichern wird der Adressbereich angezeigt, z.B. `2000 213F 0000`.

Laden:
Ein auf USB-Medium gespeicherter Quelltext wird mit `L <ENTER>` geladen. Wurde der angegebene Dateiname nicht gefunden, so meldet der Monitor `NAME FILE ERROR` und man befindet sich wieder im Kommandomodus. War das Laden erfolgreich, so werden die Eckdaten angezeigt, z.B. `BLSPAS-QUELLTEXT:E 2000 2071 0000`

"Normale" Quelltexte (wenn diese einen Z80-Header tragen und den Dateityp "E" aufweisen) werden erkannt und ebenfalls geladen. In diesem Fall erscheint eine weitere Meldung "`Dekomprimieren…`". Achtung, es wird bei diesen Dateien nicht auf den korrekten Inhalt getestet!

Anlage A: Schlüsselworte

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**

Vordefinierte Standardbezeichner:

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

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

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 (= <> > < >= <=)