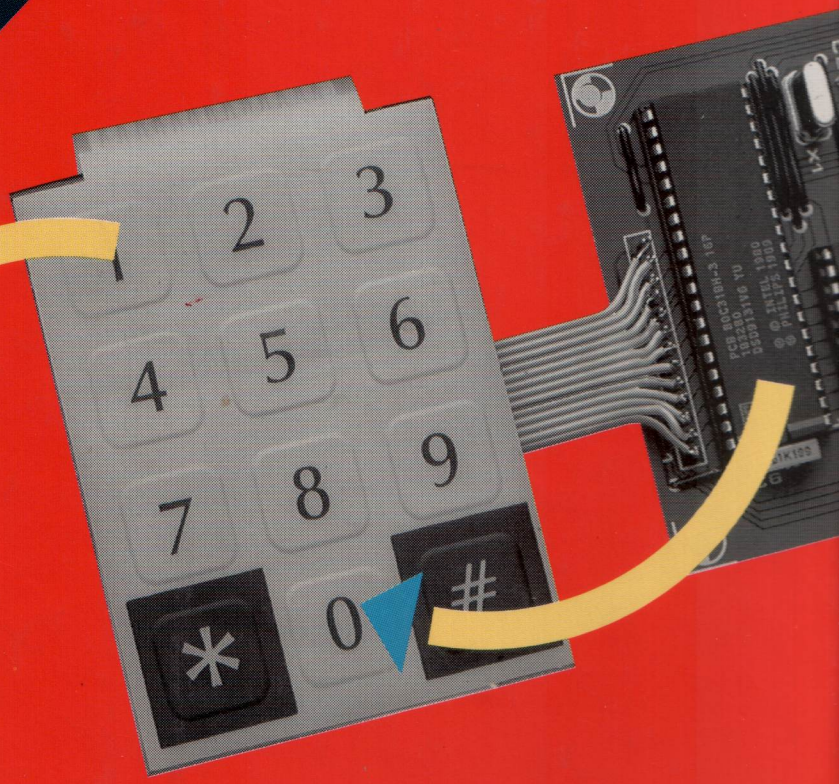
ELEKTOR SCHALTUNGSPRAXIS

# Einplatinen-computer




elektor

# Elektor Schaltungspraxis
## *Einplatinencomputer*

Einplatinencomputer kommen in elektronischen Schaltungen immer häufiger vor und erledigen betriebssicher, anwenderfreundlich, schnell und kostengünstig vielfältige Aufgaben in den Bereichen Steuern und Regeln, Codieren und Decodieren, Signalaufbereitung.

Das vorliegende Buch beschreibt fertig entwickelte Boards auf Basis der Prozessoren 8051/52 und Z80, die in Hochsprache oder in Assembler programmiert werden können. Wichtige weitere Bausteine für die Kleincomputersysteme wie Tastatur und Display sind ebenso enthalten wie nützliche Werkzeuge zum Entwickeln und Testen. Eine Reihe technisch hochinteressanter Anwendungsbeispiele sind nachbaufertig abgedruckt und ausführlich beschrieben, in diesen Beispielen sind gleichzeitig wichtige Anregungen zur Verwirklichung eigener Projekte enthalten.

Aus dem Inhalt:

- 8032/8052-Compuboard, programmierbar in Basic und Assembler, preiswertes Board mit einseitigem Platinenlayout, 32 KB ROM, 32 KB RAM und EPROM-Progammierhardware auf der Platine
- Euro-Z80, Meßcomputer-Board mit PC-kompatibler RS232-Schnittstelle und 8-Bit-A/D-Wandler. Programmierbar in Basic und Assembler
- 8031/32-Prototypenkarte mit Lochraster- und IC-Feld
- EPROM-Emulator für 2764 bis 27512
- Gigahertzzähler, Universalzähler und Signalgenerator
- DRAM-Druckerpuffer mit 1 oder 4 MByte Kapazität
- 12-Bit-A/D-Wandlerkarte mit 16 gemultiplexten Eingangskanälen

Dieses Buch ist in enger Zusammenarbeit zwischen erfahrenen Applikationsingenieuren und dem Elektor-Labor entstanden.

**ISBN 3-928051-34-2**
**Elektor-Verlag GmbH, 5100 Aachen**

Die Bedeutung aller benutzten I/O-Adressen und der einzelnen Bits ist in den ***Tabellen 1*** und ***2*** angegeben.

**Stückliste**

Widerstände:
R1 = 1 M
R2, 3, 4 = 1k, 1 %
R5 = 3k3
RA1 = DIL-Array 220 Ω · 8
RA2 = SIL-Array 1k · 8
RA3 = SIL-Array 1k · 4
RA4 = DIL-Array 1k · 8

Kondensatoren:
C1 = 4µ7/10 V, Tantal
C2 = 100 n
C3, 4 = 10µ/10 V, Tantal
C5 = 100 µ/16 V
C6 = 10 µ/10 V

Halbleiter:
D1 ... D7 = LED rot 3 mm
IC1 ... IC8 = Optokoppler 4N28
IC9 = DAC 0808LCN (SGS-Thomson)
IC10 = 4067 (RCA)
IC11 = ULN 2004A (Sprague)

Außerdem:
RE1 ... RE7 = 5-V-Printrelais, 1xUm
K1 = 6-poliger Pfostenfeldstecker, einreihig
K2, 6 = 5-polige Pfostenfeldbuchse, einreihig
K3, 5, 7, 9, 11 = 4-polige Pfostenfeldbuchse, einreihig
K4, 8 = 12-polige Pfostenfeldbuchse, einreihig
K10 = 5-poliger Pfostenfeldstecker, einreihig
K12 = 12-poliger Pfostenfeldstecker, einreihig
K13 = 4-poliger Pfostenfeldstecker einreihig
K14 = 26-poliger Pfostenfeldstecker, zweireihig
K15 = 16-poliger Pfostenfeldstecker, zweireihig
K16 ... K22 = 3-polige Lötlüsterklemme
Bu1 = Piezosummer
Platine 900059

# 2.4. Basic-Interpreter zum Euro-Z80

*von Dr. Hans Hehl*

Durch die Hardware des EURO-Z80 waren einige Bedingungen für die Programmiersprache gegeben. So standen ein 16-KByte-EPROM für die Sprache und ein 8-KByte-RAM für Programme zur Verfügung. Neben der Programmiersprache benötigte man noch Platz für einen kleinen Editor für Speicherbereiche im EPROM. Zur speziellen Anpassung mußte überdies der Quellcode der Programmiersprache zugänglich sein. Um auf dem PC (MS-DOS) zumindest mit Z80-Emulator die Programmentwicklung zu ermöglichen, durfte der Interpreter keine illegalen Z80-Befehle besitzen.

Bei dem verwendeten EPROM-Interpreter handelt es sich um die EPROM-Version eines Disketten-Basic-Interpreters für das Betriebssystem CP/M. Wer einen Basic-Interpreter zur Anpassung an ein spezifisches System mit Z80 oder HD64180 sucht, findet mit diesem

Disketten-Interpreter nebst vollständigem Quellcode auf Diskette einen sehr flexiblen „Interpreter-Baukasten" mit vielen, für Basic-Interpreter teilweise einmaligen Möglichkeiten. Grundstock sind über 300 Einzelmodule in der Quelle, die fast beliebig kombiniert werden können.

Das Ergebnis einer solchen Anpassung ist das Basic-EPROM des EURO-Z80. Der Befehlssatz wurde in Heft 6/89 in der ***Tabelle 1*** angegeben, dabei sind übrigens die Basic-Befehle ATN, OPTION und LVAR wohl versehentlich nicht aufgeführt worden, POS dagegen ist nicht vorhanden.

**Tabelle 1**

```
10   DEF FNFAC (I)
20   IF I = 0 THEN FNRETURN 1      REM FAC (0) = 1
30   FNEND FNFAC (I-1)*I     REM FAC (I) = FAC (I-1)*I

10   DEF FN REP4 (I$, I)
20       J$ = ''''
30      :  IF I = 0 THEN FNRETURN J$
40         FOR J = 1 TO I
50            J$ = J$ + I$
60       NEXT
70   FNEND J$
80   :
90    INPUT ''ZEICHEN, ANZAHL''; I$, I
100   PRINT FNREP$ (I$, I)
```

## Die Basic-Befehle

Basic-Befehle sind zwar eigentlich standardisiert, aber fast jeder Basic-Interpreter hat sich durch Ergänzungen in seiner Befehlssyntax von diesem Standard entfernt. Es sollen nun nur die Basic-Befehle erläutert werden, die sich vom GW-Basic unterscheiden oder besonders schwierig sind. Ausführliche Beschreibungen aller Basic-Befehle sind im Handbuch der Diskettenversion zu finden.

### BYE

Der Interpreter wird verlassen und zu Adresse 0 im EPROM gesprungen, also ein Kaltstart durchgeführt. Der Monitor meldet sich.

### CALL

Mit diesem Befehl wird zu der angegebenen Adresse in Maschinenspracheroutinen gesprungen. Dabei können mit dem Befehl die angegebenen Werte übertragen werden. Sie werden auf dem Stapel abgelegt und sind mit dem Z80-Befehl **POP** zugänglich. Das BC-Register enthält die Anzahl n der übertragenen Werte, diese sind nur durch den Speicher begrenzt. Im HL-Register ist die Rücksprungadresse angegeben. Wird diese auf den Stapel gebracht, kann mit **RET** zurückgesprungen werden.

### CALL & 7000, 45, 3

Ab Adresse 28672 (7000 h) muß sich ein Maschinenprogramm befinden, dem die Werte 45 und

3 übergeben werden. Register BC enthält 00 02, auf dem Stapel befinden sich 03 00 2D 00. Mit **CALL** C2 wird BASIC verlassen und es meldet sich der Monitor.

**CLEAR**

Es können Variablenwerte gelöscht und Platz für Zeichenketten reserviert werden. Mit dem Befehl **FRE(X)** bzw. **FRE(X$)** wird der noch freie Speicherplatz ermittelt. X ist eine Hilfsvariable und enthält keinen Wert.

**CLEAR** löscht alle Variablen im Speicher. In einem Programm muß dieser Befehl am Anfang stehen, sonst werden nachher eingegebene Werte nicht mehr berücksichtigt. Es sind dabei 100 Bytes für Zeichenketten und 182 Bytes für einen Kanal für Ein- oder Ausgabebetrieb reserviert. Mit **CLEAR 1000** werden nun 1000 Bytes für Zeichenketten reserviert. **PRINT FRE(X$)** ergibt dann den Wert 1000. Es kann der belegte Platz auch wieder verkleinert werden. **CLEAR 0** beseitigt die Standardreservierung von 100 Bytes für Zeichenketten. **CLEAR 1000,2** reserviert zusätzlich zwei Ausgabekanäle.

**DEF FN**

a) Befehlsform: **DEF FN Name (Dummy-Variable)** = math. Funktion

Funktionen müssen mit einem Namen gekennzeichnet sein, der mit FN beginnt und bis zu zwei weitere Zeichen (nach den Regeln für Variablen ausgewählt) enthalten kann. Mehr als zwei weitere Zeichen nach FN sind möglich, werden aber nicht ausgewertet. Die Dummy-Variable dient als Platzhalter.

Beispiele für Funktionsnamen sind FN Q, FNA9, FNZZ usw. und Funktionen können so aussehen:

**DEF FN(X) = I + (K-1)*25+(L-1).**

Aufgerufen wird die am Programmanfang definierte Funktion z. B. mit **PRINT FN(X).**

Zusätzlich ist auch die Verwendung von mehreren Dummy-Variablen sowie von Funktionen für die Verarbeitung von Zeichenketten erlaubt. Funktionen dürfen sich auch über mehrere BASIC-Zeilen erstrecken („multiline"-Funktion), auch mit **GOSUB**-Befehlen. Dabei können andere Funktionen ebenso wie die gerade bearbeitete aufgerufen werden.

b) Befehlsform: **DEF FN Name (Dummy 1, Dummy 2, ..., Dummy n)**

Funktion in mehreren BASIC-Zeilen

**FNEND** (Variable für Funktionswert)

Die Funktion kann jedoch nur einfache Funktionen mit der Befehlsform a) enthalten, keine „multi-line"-Funktion. **FNEND** beendet die Funktion und übergibt den berechneten Ausdruck der angegebenen Variablen, die vom selben Typ (Zahl oder Zeichenkette) wie die Funktionsvariablen sein muß. Beispiele zur Funktionsform b) siehe ***Tabelle 1***.

Der Befehl **FNRETURN** (Variable für Funktionswert) beendet die Funktion und übergibt einen bestimmten Wert an das Programm. Mehrere **FNRETURN**-Befehle sind erlaubt.

Ein Syntaxfehler in der Definition der Funktion wird erst beim Aufruf der Funktion erkannt, wobei dann die Zeilennummer der BASIC-Zeile ausgegeben wird (in der der Funktionsaufruf steht und nicht die Zeile mit der Definition).

**EE**

Die Variable **EE** enthält die Eulersche Zahl e, die Basis der natürlichen Logarithmen (2,7182818285).

**ERROR**
Mit dem Befehl **ERROR XY (0-48)** kann die Fehlermeldung XY ausgegeben werden. Fehlermeldungen sind aber auch beliebig definierbar, wobei die Texte den Nummern 0 - 255 zugeordnet werden können. Entsprechend der gewünschten Abbruchbedingung des Programmes wird die Fehlernummer angegeben, z.B.
**IF A < 5 THEN ERROR 50**
Wenn der Wert der Variablen A größer ist als 5, wird die Zahl 50 als Fehlernummer gespeichert. Zu Beginn des Programmes muß ein Sprungbefehl **ON ERROR GOTO XY** vorhanden sein. Zeile XY lautet dann:
**1000 IF ERR = 50 THEN PRINT "A zu groß": GOTO XZ**
Nach dem Befehl **ON ERROR GOTO** ... darf kein **CLEAR**-Befehl enthalten sein.

**EXP**
Es wird die Exponentialfunktion $Y = e^x$ berechnet, wobei **e** die Eulersche Zahl 2,718... darstellt.
**PRINT EXP(88)** liefert die Anzeige **1.651636255E + 38.**

**Ganzzahldivision: PRINT 5 / 3**
Wird eine Division mit diesem BASIC-Befehl durchgeführt, so entfällt die Verwendung des **INT**-Befehles, wenn ein ganzzahliges Ergebnis benötigt wird. Bei der deutschen Tastatur entspricht dem Schrägstrich der Großbuchstabe „**Ö**".

**INPUT**
Mit diesem Befehl werden Zahlen in numerische **(A0-Z9)** und Zeichenketten in Stringvariablen **(A0$ - Z0$)** eingelesen. Variablenkennzeichnungen mit mehr als zwei Buchstaben sind erlaubt, werden aber nicht unterschieden. <CR> als Eingabe führt nicht zum Programmabbruch, sondern übernimmt den Wert 0 bei numerischen bzw. einen Leerstring bei Stringvariablen. Hexadezimale Zahlen werden mit vorangestellten „**&**"-Zeichen und mit Großbuchstaben eingegeben.
Kommata können in der Eingabe mit der Befehlsform **INPUT LINE "Text"**; als Stringvariable enthalten sein, ohne daß es zu Fehlermeldungen kommt. Es sind nur Stringvariablen erlaubt.

**INSTER**
Befehlsform: **INSTR ("Zeichenkette 1", "Zeichenkette 2", Startzeichen, Zeichenanzahl)**
Dieser Befehl durchsucht eine Zeichenkette (Stringvariable oder Zeichenkette in Anführungszeichen) nach einer zweiten Zeichenkette. Die Suche beginnt ab dem ersten Zeichen oder, falls angegeben, ab dem Startzeichen und erstreckt sich über soviele Zeichen wie mit der Zeichenanzahl angegeben sind.
Mit **PRINT INSTR ("abcdefabcdefgh", "cd", 4)** erhält man die Anzeige **9.**

**LIST**
Befehlsform:
**LIST # Kanal, Zeilennummer oder Zeilenbereich**
Es werden BASIC-Programmzeilen auf dem angegebenen Kanal ausgegeben. Ohne Kanalangabe erfolgt die Ausgabe am Bildschirm, mit **LIST # 2** über den Kanal Nr. 2 auf den Drucker. Mit dem Befehl **OPTION** kann die vorgegebene Zeilenlänge von 80 Zeichen getrennt für Bildschirm und Drucker verändert werden (siehe **OPTION**). Eine Zeilennummer kann durch

einen Punkt ersetzt werden. Die Leertaste stoppt die Ausgabe, mit nochmaligem Drücken wird sie fortgesetzt, mit <CTRL-E> abgebrochen (Meldung: ** **HALT** **). <CTRL-O> verhindert die Bildschirmanzeige, am Ende wird die Systemmeldung **„Fertig“** ausgegeben.

**LOG**
Es wird der natürliche Logarithmus (Logarithmus zur Basis e) berechnet, mit **PRINT LOG(1000)** ergibt sich die Anzeige: **6.907755279.**

**LOG10**
Es erfolgt die Berechnung des dekadischen Logarithmus (zur Basis 10). **PRINT LOG10(1000)** liefert den Wert **3**.

**LVAR**
Befehlsform. **LVAR # Kanal**
Die in einem Programm verwendeten Variablen und ihre Werte können auf den Bildschirm (keine Kanalangabe oder # 0) bzw. auf den Drucker (Kanal # 2) ausgegeben werden, allerdings erst, nachdem mit **RUN** gestartet wurde. Die Ausgabe kann wie beim **LIST**-Befehl unterbrochen bzw. abgebrochen werden. Leider werden keine dimensionierten Variablen angezeigt.

**ON**
Die Kombination ist auch mit **ERROR, ERL** und **ERR** möglich.

**OPTION**
Befehlsform: **OPTION # Kanal, "Parameter", Argumente**
Dieser Befehl ermöglicht die Veränderung der vorgegebenen Werte der Parameter der Ein- und Ausgabekanäle des Interpreters.
a) Parameter **W** (Zeilenlänge): Mit dem Parameter **W** wird als Argument die Anzahl der Zeichen einer Zeile angegeben, nach denen automatisch ein Carriage Return (CR) und ein Linefeed (LF) folgen. Möglich sind Zeichenanzahlen von 14-255 (sonst Fehlermeldung „**nicht erlaubt**"). So können verschiedene Bildschirme und Drucker angepaßt werden.
**OPTION # 0, "W", 60** ergibt eine Bildschirmzeilenlänge von 60 Zeichen. Eine Anpassung der Zeilenlänge bei einem Drucker erfolgt mit **OPTION # 2, "W", 60**.
b) Parameter **N** (Zeichenanzahl, Art nach CR und LF): Im Argument 1 kann die Anzahl der Zeichen, im Argument 2 die Zeichenart als Dezimalwert angegeben werden, die nach CR/LF zusätzlich vom Interpreter ausgegeben werden. Damit können Zeitprobleme bei Peripherieeinheiten umgangen werden. Möglich sind 0-255 Zeichen im Argument 1. Fehlt Argument 2, wird es gleich 0 gesetzt. Beim Bildschirm oder Drucker kann mit diesem Befehl der linke Rand nach rechts verschoben werden.

Der Befehl: **OPTION # 0, "N", 7,32** gibt zu Beginn jeder Bildschirmzeile 7 Leerzeichen (Blanks) aus, was auch für den Druckerkanal Nr. 2 möglich ist.
c) Parameter **Q**: Dieser Parameter für Anfangs- und Schlußzeichen von Zeichenketten wird beim EURO-Z80 nicht benötigt.
d) Parameter **P** (Zeichen nach dem Input-Befehl = Prompt): Wartet der BASIC-Interpreter bei einem Input-Befehl auf eine Eingabe, so wird dies normalerweise durch ein Fragezeichen am Bildschirm angezeigt. Als Argument kann der Dezimalwert eines beliebigen Zeichens eingesetzt werden, mit 0 entfällt dieses. Der Wert 63 ergibt ein Fragezeichen. Mit **OPTION # 0, "P", 42** wird nach dem **INPUT**-Befehl ein Sternchen (*) ausgegeben.

Mit dem Start des Interpreters sind für die jeweiligen Kanäle folgende Voreinstellungen gegeben:
Bildschirm (Kanal # 0): W = **80**, N = **3,0**, Q = **0**, P = **63** Drucker (Kanal # 2): W = **80**, N = **3,0**, Q = **0**

**OUTBYTE**
Befehlsform: **OUTBYTE #** Kanal, Zahl oder Variable oder Zeichenkette
Von einer Zeichenkette wird jedes Zeichen als einzelnes Byte ausgegeben. Steuerzeichen wie CTRL-I und andere lassen sich so an Daten anhängen. Weiterhin lassen sich mit **OUTBYTE # 0** Zahlenkolonnen ohne Zwischenraum am Bildschirm ausgeben wie es bei dem Befehl **PRINT** nicht der Fall ist. **OUTBYTE # 2** spricht den Drucker an.

**PI**
Die Variable **PI** enthält den Wert der Kreiszahl 3,1415926536.

**POKE**
Befehlsform: **POKE Speicheradresse, Wert**
In die Speicherstellen 0 bis 65535 können Dezimalwerte von 0 bis 255 geschrieben werden. Beide Parameter sind auch über Variablen zuweisbar. **POKE &1035, &3E** schreibt den Wert 62d in die Speicherstelle. Die Zahlen wurden hexadezimal angegeben. (Kennzeichnung mit „**&**“, Buchstaben müssen **groß** geschrieben werden!)

**PRECISION**
Ergebnisse von Berechnungen (z. B. Dezimalbrüche) werden mit maximal 11 Zeichen (ohne Punkt) ausgegeben. Diese Anzahl kann verringert werden, wobei die letzte ausgegebene Stelle gerundet ist. Intern wird mit 11 Zahlen weiter gerechnet. Die Zahl 0 oder keine Angabe nach dem Befehl ergeben wieder 11 Zeichen.
Mit **PRECISION 5** werden bei Rechenergebnissen Zahlen mit maximal 5 Zeichen ohne Punkt ausgegeben. **PRINT 10/7** ergibt danach die Zahl **1.4286, PRINT 100/7** ergibt **14.286.** Mit **PRECISION 0** wird nach **PRINT 10/7** die Zahl **1.4285714286** ausgegeben.

**PRINT**
Mit dem **PRINT**-Befehl erfolgt die Ausgabe von Zeichen, Zeichenketten und Variableninhalten. An Stelle des Befehlswortes **PRINT** kann auch die Kurzform, ein Fragezeichen eingegeben werden. In BASIC-Zeilen wird das Fragezeichen jedoch nicht durch **PRINT** ersetzt. **PRINT** oder **PRINT # 0, "ZEICHENKETTE"** spricht über Kanal 0 den Bildschirm, **PRINT # 2, "ZEICHENKETTE"** über Kanal #2 den Drucker an.

**PRIVACY**
Durch die Angabe eines Schlüsselwortes (password) an erster Stelle in irgendeiner Programmzeile sind alle Befehle geschützt, mit denen der Inhalt eines BASIC-Programmes zugänglich gemacht oder verändert werden kann (z. B. **DELETE, LIST** oder **RENUMBER**). In allen Fällen wird die Fehlermeldung **"nicht verfügbar"** ausgegeben. Auch das Löschen einer Zeile durch Eingeben der Zeilennummer mit nachfolgendem <CR> geht nicht. Das Schlüsselwort kann irgendeine Konstante, Variable oder beliebige Zeichenkette sein.

Mit der Basic-Zeile **10 PRIVACY "SeCrEt"** wird in der Zeile 10 als Schlüsselwort die Zeichenkette **"SeCrET"** definiert. Allen oben ausgeführten Befehlen muß unmittelbar nach dem Befehl das Schlüsselwort in Anführungszeichen folgen, danach können weitere, durch ein Komma getrennte Befehle angegeben werden, z. B. **LIST "SeCrEt", 100-200**.

**RENUMBER**
Befehlsform: **RENUMBER Zeile, Abstand, Startzeile**
Alle Zeilennummern (Sprünge wie **GOTO, GOSUB** usw.) in einer BASIC-Zeile werden entsprechend berichtigt. Es kann auch erst ab einer bestimmten Zeile mit der Neunumerierung begonnen werden. Fehlt einer der Parameter, so wird der Wert 10 dafür angenommen. Mit **RENUMBER 200,50** werden alle Programmzeilen mit 200 beginnend mit dem Abstand 50 durchnumeriert. **RENUMBER 1000, 20, 500** ergibt neue Zeilennummern ab der alten Programmzeile 500 beginnend mit 1000 mit dem Abstand 20.

**RND** und **RANDOMIZE**
Ohne Argument liefert der **RND**-Befehl Pseudo-Zufallszahlen zwischen 0 und 1, die einem festgelegten Algorithmus entstammen. Dadurch ergeben sich nach dem Start des BASIC-Interpreters immer dieselben Zahlen.
Der Befehl **RANDOMIZE** ergibt einen zufälligen Startpunkt der Pseudo-Zufallszahlenfolge (Zahl aus dem Z80-Refresh-Register R). Wird mit **PRINT RND(0)** als Argument der Wert Null angegeben, wird erstmalig eine beliebige Zahl angezeigt und dann wiederholt. Werte größer Null entsprechen dem **RND**-Befehl ohne Argument. **PRINT RND(1)** ergibt dieselben Zahlen wie **PRINT RND**.
Negative Argumente bestimmen den Startpunkt der Zufallszahlenfolge. Die Zufallszahlensequenzen sind dann nach jedem BASIC-Start identisch.

**VARPTR**
Die dezimale Adresse der Inhalte von Variablen im Speicher wird mit diesem Befehl ermittelt und kann direkt bei den Befehlen **PEEK** und **POKE** verwendet werden. Bei Variablen für Zeichenketten (z. B. A$) wird mit **PRINT VARPTR(A)** oder **PRINT VARPTR(A$)** die Adresse des Stringzeigers angegeben. Das erste Byte an dieser Adresse enthält die Länge der Zeichenkette, das nächste Byte ist immer 0. Die folgenden beiden Bytes enthalten die Adresse der Zeichenkette, das LOW-Byte zuerst. Die letzten beiden Bytes sind nicht benutzt.

**Programmentwicklung auf dem PC**
Auf die Möglichkeit der Programmentwicklung ohne EURO-Z80 durch Programmierung in GW-Basic auf dem PC wurde bereits im Kapitel EURO-Z80 hingewiesen. Eine andere Möglichkeit besteht darin, einen Z80-CP/M-Emulator und die Disketten-Interpreterversion zu verwenden. Die Basic-Programme sind im ASCII-Format austauschbar und können von beiden Interpreten von Diskette geladen werden. Allerdings dürfen keine spezifischen Befehle nur eines Interpreters verwendet werden.
Die Programmerstellung auf dem PC kann auch mit einer Textverarbeitung erfolgen. Nur müssen die Texte als reine ASCII-Datei abspeicherbar sein, also keine speziellen Steuerzeichen besitzen. Der Basic-Interpreter erlaubt auch eine strukturierte Anordnung der Programmzeilen.

**Start des Interpreters**

Nach dem Start des Interpreters erscheint bei Bildschirmbetrieb ein Fragezeichen. Beim ersten Mal kann nur ein ‚c' (kleiner Buchstabe c) eingegeben werden. Es folgt die Frage ‚Speichergrenze', die in der Regel mit der Taste <CR> (Carriage Return) beantwortet wird. Um eine Maschinenspracheroutine im Speicher zu schützen, kann die Speicherobergrenze durch Angabe der neuen Obergrenze herabgesetzt werden. Der Interpreter meldet sich dann je nach der Adresse des oberen RAM-Endes (Label SPEND) mit der Angabe des freien Speicherplatzes und mit der Systemmeldung.

Wird bei nochmaligem Start nach einem RESET ein ‚w' (kleines w) eingegeben, so wird ein Warmstart ausgeführt und die Meldungen ‚BASIC verfuegbar' und ‚Fertig' erscheinen. Das Programm kann weiter bearbeitet werden.

Sind die RAM-Bausteine Akku-gepuffert, so muß beim Start des Interpreters zur Erhaltung des Programmes ein ‚w' eingegeben werden und der Start mit GOTO erfolgen, sonst werden die Variablenwerte gelöscht.

In der Version 2.0 enthält der Monitor einen **Z80-Debugger,** der nun auch die Entwicklung von Z80-Maschinenspracheprogrammen auf dem Meßcomputer ermöglicht. Damit ist der Euro-Z80 auch für Aus- und Weiterbildung von angehenden Maschinenspracheprogrammierern interessant.

Neben einer **komfortablen Benutzerführung** (Hilfefunktion) sind zum Beispiel Starten eines Programmes mit Vorgabe eines Breakpoints, Einzelschritt mit oder ohne CALL, Registeranzeige und Registeränderung möglich. Bedingt durch die Größe des Eproms und des BASIC-Interpreters ist die Ausgabe der Z80-Mnemonics nicht möglich.

Hierzugibt es eine eine **Debugger-Version** des Betriebssystems ohne BASIC-Interpreter, wobei nun Platz für die Ausgabe der Befehlsworte wie LD (DE),A für das Byte $12_{16}$ ist. Man kann auch Maschinensprache-Programme vom PC laden und abspeichern. Die Ein- und Ausgabe von Werten an Portbausteine wird außerdem binär angezeigt.

Wer den Euro-Z80 ohne PC als stand-alone **BASIC-Steuerrechner,** beispielsweise für eine Heizungssteuerung betreiben will, kann nun eine kleinere Interpreter-Version mit dem BASIC-Programm zusammen mit Eprom ablegen. Nach dem Einschalten der Stromversorgung des Euro-Z80 startet der Interpreter das Programm automatisch. Hierzu gibt es eine eigene Dokumentation und ein PC-geschütztes Entwicklungssystem. Ab Adresse $0_{16}$ im Eprom beginnt der Eprom-Interpreter, die BASIC-Programme befinden sich zum Beispiel ab Adresse $2500_{16}$ im Speicher. In der **Vollversion (mit Quellcode)** können die jeweiligen Adreßlagen unterschiedlichen Hardwarebedingungen angepaßt werden. Am PC ermöglicht ein spezieller Interpreter (mittels Z80-Emulatur), BASIC-Programme in den Adreßbereich des Euro-Z80 zu laden. Diese Interpreterversion beginnt erst ab Adresse $8000_{16}$, BASIC-Programme können dann bereits ab Adresse $2500_{16}$ abgelegt werden.

## Der Z80-Debugger

Allgemein können Befehle als Groß- oder Kleinbuchstaben eingegeben werden. Die Befehlsform wird jeweils am Bildschirm angezeigt, wobei Komma oder Leertaste als Trennzeichen zwischen den einzelnen Befehlsteilen erlaubt sind. Eine *falsche* Eingabe wird mit einem

Fragezeichen quittiert und muß neu eingegeben werden. ESCAPE löscht den Bildschirm und die Register. Der Stapelzeiger wird auf $7F00_{16}$ gesetzt und der Monitor neu (ohne neue Initialisierung der Portbausteine) gestartet. Adreßangaben werden gespeichert und bei nachfolgenden Befehlen entsprechend verwendet. Nach einem Reset des Z80 meldet sich der Debugger am PC mit dem Text aus ***Tabelle 1***.

**Tabelle 1: Die Startmeldung des Debuggers**

```
EURO-Z80-MONITOR (C) DR. HEHL
?->INFO / STAND 21.11.90 VERS. 2.0
>
```

Für eine korrekte Befehlseingabe gibt es eine Hilfefunktion. Sie wird durch Eingabe eines Fragezeichens aufgerufen. Die Bildschirmanzeige ist in ***Tabelle 2*** aufgeführt.
Die einzelnen Befehle des Debuggers in Kurzform:

**D für DISPLAY**
Nach Eingabe eines ‚D' erscheint die Meldung ‚D = SPEICHER anzeigen ab Adresse'. Jetzt kann die (hexadezimale)Adresse eingegeben werden, zum Beispiel 2A0 oder 02A5 oder 100. Es werden immer 256 Byte angezeigt. ***Tabelle 3*** zeigt einen Teil des HEX-Dumps. Zur Anzeige der nächsten 256 Byte genügt die zweimalige Eingabe von <CR> (RETURN-Taste). Nach der ersten Eingabe erscheint wieder der Befehl, dann wird beispielsweise der Speicherinhalt ab Adresse $03A0_{16}$ bis $049F_{16}$ angezeigt. Statt des zweiten <CR> kann man auch eine neue Dump-Adresse angeben. Eine sinnvolle Anzeige ergibt sich natürlich nur, wenn ab dieser Adresse auch ROM oder RAM vorhanden ist.

**F für FILL**
Das F-Kommando ‚F = SPEICHER fuellen von bis mit Wert' ermöglicht die Vorbelegung eines RAM-Bereiches mit einem Wert, beispielsweise ab Adresse $7000_{16}$ bis $7200_{16}$ mit dem Wert $45_{16}$. Dabei ist zu beachten, daß der Monitor ausgehend von der obersten RAM-Adresse (üblicherweise $7FFF_{16}$) nach unten Platz für zwei Stapelbereiche benötigt: erstens für sich selber ($7FFF_{16}$ ... $7F01_{16}$) und zweitens für den Einzelschrittmodus (ab $7F00_{16}$ nach unten). Die Stapelwerte darf man natürlich nicht mit dem F-Kommando überschreiben.

**G für GO**
Mit diesem Kommando ‚G = STARTEN ab Adresse, Breakpoint' kann man ein Z80-Maschinenspracheprogramm in RAM ab einer bestimmten Adresse starten (üblicherweise $6000_{16}$) und an einer sinnvollen (Break-)Adresse anhalten. Der Breakpoint muß nach einem vollständigen Z80-Befehl gesetzt werden. Soll das Programm etwa nach den Bytes 3E 23 (LADE AKKU mit $23_{16}$) ab Adresse $6000_{16}$ anhalten, ist die korrekte Breakpoint-Adresse $6002_{16}$.

Da der Abbruch über den RST-30-Befehl erreicht wird, der bei diesem Beispiel an der Adresse $6002_{16}$ gesetzt wird, funktioniert das ganze natürlich nur bei Adressen im RAM-Bereich. In ***Tabelle 4*** ist der Ablauf des Beispiels angegeben. Nach dem Sternchen werden die Break-Adresse und die nächsten 8 Bytes angezeigt. Es folgen die Register des Z80 nach der Befehlsausführung, wobei im Register A (Akku) der Wert 23 steht.

**Tabelle 2: Die PC-Bildschirmanzeige der Hilfe-Funktion**

```
? -> INFO / STAND 21.11.90 VERS. 2.0
D 100 <CR>: . . . . . = SPEICHER anzeigen ab Adresse :
F 7000 7200 45 <CR>: = SPEICHER fuellen von bis mit Wert :
G 7000,7003 <CR>: . . = STARTEN ab Adresse, Breakpoint :
T 7000 <CR>: . . . . = EINZELSCHRITT ab Adresse :
C 7000 <CR>: . . . . = EINZELSCHRITT (ohne CALL) ab Adresse :
E 7000 <CR>: . . . . = SPEICHER aendern ab Adresse :
Leertaste = vorwaerts, Minustaste = zurueck, CR beendet
Y FE 1A <CR>: . . . . = Wert1, Wert2, usw. suchen :
R = REGISTER zeigen, RB0130 = Register BC aendern,
RB = BC loeschen, RZB0130 = Zweitregister BC aendern
A=AF, B=BC, D=DE, H=HL, X=IX, Y=IY, P=PC, S=SP
X <CR>: . . . . . = Register loeschen :
BASIC starten: K = Kaltstart, W = Warmstart
>
```

**Tabelle 4: Der GO- und T-Befehl und die jeweilige Anzeige**

```
G = STARTEN ab Adresse, Breakpoint :6000,6002
* 6002 21 FF FF E5 F1 00 00 00
AF=2300 BC=0000 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=6002
AF=0000 BC=0000 DE=0000 HL=0000 I =00 F:= ........ ........

Bildschirmanzeigen nach mehrmaliger Eingabe von 'T' und <CR>:
<T = EINZELSCHRITT ab Adresse :6002
* 6005 E5 F1 00 00 00 00 00 00
AF=2300 BC=0000 DE=0000 HL=FFFF IX=0000 IY=0000 SP=7F00 PC=6005
AF=0000 BC=0000 DE=0000 HL=0000 I =00 F:= ........ ........

<T = EINZELSCHRITT ab Adresse :6005
* 6006 F1 00 00 00 00 00 00 00
AF=2300 BC=0000 DE=0000 HL=FFFF IX=0000 IY=0000 SP=7EFE PC=6006
AF=0000 BC=0000 DE=0000 HL=0000 I =00 F:= ........ ........

<T = EINZELSCHRITT ab Adresse :6006
* 6007 00 00 00 00 00 00 00 00
AF=FFFF BC=0000 DE=0000 HL=FFFF IX=0000 IY=0000 SP=7F00 PC=6007
AF=0000 BC=0000 DE=0000 HL=0000 I =00 F:=SZ.H.PNC .......
```

**Tabelle 3: So wird der Speicherinhalt angezeigt**

```
>D = SPEICHER anzeigen ab Adresse :2a0
02A0 F7 02 CD D3 06 C3 B8 04 21 E3 08 18 03 21 AE 08 w.MS.C8.!c . . .!..
02B0 CD 90 07 CD 4E 07 ED 4B 54 62 ED 43 52 62 ED 5B M..MN.mKTbmCRbmÄ
02C0 58 62 38 0A CD 11 07 C1 ED 43 52 62 18 09 C5 E1 Xb8.M..AmCRb..Ea
02D0 CD 25 07 ED 4B 52 62 0A 03 FE DD CA 7C 04 FE FD M%.mKRb . . .ÜJ . . .
```

**Tabelle 5: Suchen einer Bytefolge**

```
Bildschirmanzeige:
Y = Wert1, Wert2, usw. suchen :ff ff e5

4003 FF FF E5 F1 00
6003 FF FF E5 F1 00
```

**Tabelle 6: Registervariationen**
Bildschirmanzeigen:

```
>R
AF=0000 BC=0000 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0000 DE=0000 HL=0000 I =00 F:= ........ ........

>Rb0130
AF=0000 BC=0130 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0000 DE=0000 HL=0000 I =00 F:= ........ ........

>Rzb240
AF=0000 BC=0130 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0240 DE=0000 HL=0000 I =00 F:= ........ ........

Rb
AF=0000 BC=0000 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0240 DE=0000 HL=0000 I =00 F:= ........ ........

>Ra0023
AF=0023 BC=0000 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0240 DE=0000 HL=0000 I =00 F:= ......NC ........

>Ra2300
AF=2300 BC=0000 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0240 DE=0000 HL=0000 I =00 F:= ........ ........

>Raffff
AF=FFFF BC=0000 DE=0000 HL=0000 IX=0000 IY=0000 SP=7F00 PC=0000
AF=0000 BC=0240 DE=0000 HL=0000 I =00 F:=SZ.H.PNC ........
```

**Tabelle 7: Ausgabe der Z-80-Befehlsworte**

```
I = BEFEHLE interpretieren ab Adresse :0
0000 LD A,9B      3E 9B D3 03 3E A8 D3 07
0002 OUT 03,A      D3 03 3E A8 D3 07 3E 4E
0004 LD A,A8      3E A8 D3 07 3E 4E D3 09
0006 OUT 07,A      D3 07 3E 4E D3 09 3E 05
0008 LD A,4E      3E 4E D3 09 3E 05 D3 09
000A OUT 09,A      D3 09 3E 05 D3 09 21 FF
000C LD A,05      3E 05 D3 09 21 FF 7F 22
000E OUT 09,A      D3 09 21 FF 7F 22 03 60
0010 LD HL,7FFF   21 FF 7F 22 03 60 F9 2E
0013 LD (6003),HL 22 03 60 F9 2E 00 22 1C
0016 LD SP,HL     F9 2E 00 22 1C 60 21 00
0017 LD L,00      2E 00 22 1C 60 21 00 60
0019 LD (601C),HL 22 1C 60 21 00 60 22 05
001C LD HL,6000   21 00 60 22 05 60 AF 32
001F LD (6005),HL 22 05 60 AF 32 2C 60 CD
0022 XOR A        AF 32 2C 60 CD 02 0B C3
```

**Tabelle 8: Tabelle für Speichererweiterung beziehungsweise Adreßänderung**

1) Speicherendadresse $7FFF_{16}$: Label SPEND
Version 1.0 : vier Adressen im Eprom:
$0009_{16}$, $05AD_{16}$, $3496_{16}$, $34B2_{16}$
Version 2.0 : zwei Adressen im Eprom:
$0011_{16}$ und $003C_{16}$

2) Beginn der Basic-Programme $6277_{16}$: Label BASPRO
Version 2.0 :
$003F_{16}$ und $3EE3_{16}$

**T für TRACE**

Nun kann mit dem T-Kommando ‚T = EINZELSCHRITT ab Adresse' der nächste Z80-Befehl ausgeführt werden. Ein Breakpoint wird automatisch nach dem nächsten Befehl gesetzt. Nach dem Befehl CALL wird ebenfall im Einzelschritt vorgegangen. Möchte man den mit CALL *Adresse* aufgerufenen Programmteil in Echtzeit (ohne Einzelschritt) testen, so wird das C-Kommando verwendet (siehe dort). Man hätte genauso auch gleich am Anfang das T-Kommando ab Adresse $6000_{16}$ verwenden können. Entsprechend den Befehlen LD HL,FFFF - PUSH HL . POP AF zeigt das Flag-Register alle gesetzten Bits an.

**C für CALL-TRACE**

Der C-Befehl ‚C = EINZELSCHRITT (ohne CALL) ab Adresse' entspricht dem T-Befehl, allerdings wird nun ein Unterprogramm-Aufruf CALL *Adresse* ohne Einzelschritt direkt ausgeführt

**E für EDIT**

Mit dem E-Kommando ‚E = SPEICHER aendern ab Adresse' können Bytes ab einer Adresse im RAM-Bereich geändert beziehungsweise eingegeben werden, beispielsweise ein Programm. Die Eingabe eines Byte wird mit der Leertaste (Space) beendet und das nächste Byte angezeigt; <CR> beendet das E-Kommando, die Minustaste geht ein Byte zurück.

**Y für SUCHEN**

Mit dem Y-Kommando ‚Y = Wert 1, Wert 2, usw. suchen' werden Bytes im RAM gesucht, wobei mehrere Bytes eingegeben werden können. ***Tabelle 5*** zeigt so eine Suche nach der Bytefolge FF FF E5. Hier zeigt sich die unvollständige Adreßdecodierung der Hardware, RAM befindet sich erst ab Adresse $6000_{16}$.

**R für REGISTER**

***Tabelle 6*** erläutert die verschiedenen Möglichkeiten der Beeinflussung und Anzeige der Register. Um die Zweitregister BC, DE oder HL anzusprechen, muß ein ‚Z' vor dem Register angegeben werden. Es sind dies allerdings nicht die Registerwerte des Debuggers, sondern die eines Z80-Maschinenprogramms.

**X für REGISTER loeschen**

Dieser Befehl setzt alle Register bis auf den Stapel auf den Wert 0.

**K/W für BASIC Start**

Mit K erfolgt ein Kalt-, mit W ein Warmstart des Interpreters. Beim ersten Start des BASIC-Interpreters wird *immer* ein Kaltstart durchgeführt. Bei bereits mit Programm gefülltem und/oder akku-gepuffertem RAM muß ein Warmstart durchgeführt werden, sonst wird das Programm gelöscht.

**AUTO-Start**

Ist der Interpreter einmal gestartet, wird ein BASIC-Programm nach einem RESET der CPU oder mit der Taste <ESC> vom Monitor aus automatisch gestartet. Der BASIC-Befehl BYE führt zurück zum Monitor. Ein ‚W' springt zum Interpreter, ohne das Programm zu löschen. An RAM-Adresse $6000_{16}$ befindet sich nach dem Interpreterstart das Byte $2C_{16}$. Wird es mit dem E-Kommando auf 0 gesetzt, so ist der AUTO-Start ausgeschaltet.

## DEBUGGER pur

Die Z80-Debuggerversion ohne BASIC-Interpreter enthält weitere Befehle, die eine Maschinenspracheprogrammierung des Euro-Z80 erleichtern:

**I für Interpretieren**
Mit dem I-Befehl, I = BEFEHLE interpretieren ab Adresse' werden die Z80-Befehlswörter ausgegeben. Damit kann man auch den Debugger selber anschauen. Wie ***Tabelle 7*** zeigt, folgen auf die Mnemonics immer 8 Bytes zur Kontrolle. Das Beispiel ist übrigens der Debugger-Anfang. Mit der RETURN-Taste werden jeweils die nächsten 16 Befehle angezeigt.

**P für PORT**
Nach der Eingabe von P muß ein ‚I' für Lesen (Input) oder ‚O' für Schreiben (Output) folgen. Nach ‚I' wird die Portadresse angegeben, beispielsweise 7 für die PIO2 des Euro-Z80. Für die Ausgabe wird nach der Portadresse und einem Komma der Wert angegeben, z. B. PO4,1 für Pin 4 der 2. PIO, Port A. Mit der RETURN-Taste wird der zuvor eingegebene Befehl wiederholt. Damit kann die Hardware bequem getestet werden.

**H für PROTOKOLL-Drucker**
Die Eingabe eines ‚H' funktioniert als Wechselschalter für die Druckerschnittstelle. Jedes an den PC geschickte Zeichen wird ausgedruckt, bis zur erneuten Eingabe von ‚H'.

**S für SAVE**
Mit dem S-Befehl ‚S = Binaer Abspeichern von Adresse, bis Adresse' lassen sich Maschinensprache-Programme auf einer Diskette oder auf der Festplatte des PCs ablegen. Dazu braucht man ein Terminalprogramm im ASCII-Modus, beispielsweise Procomm. Am Bildschirm erscheint vorher die Meldung ‚BITTE „ ASCII- DOWNLOAD" des Terminalprogramms starten, dann <CR>'. Der Debugger gibt zuerst Startadresse und Länge des Programms als Programmkopf aus.

**L für LOAD**
LOAD lädt das Programm an die richtige Adresse im Speicher. Dafür ist der ASCII-UPLOAD-Modus im Terminalprogramm zuständig.

## BASIC-Erweiterungen

Die Euro-Z80-Version 2.0 enthält zusätzlich die Befehle AUTO, EDIT, FIND, REPLACE, RENUMBER und TRACE. Die neuen Befehle und die dazugehörenden Parameter sind im folgenden beschrieben:

**AUTO**
*Befehlsform: AUTO Zeilennummer + , Zeilenabstand*
Damit wird eine automatische Zeilennummerausgabe eingeschaltet. Die Startzeilennummer ist immer 10, der Abstand zwischen den Zeilennummern kann vorgegeben werden. Die Tastenkombination <CTRL-E> schaltet die AUTO-Funktion ab.

**EDIT**
*Befehlsform: EDIT Zeilennummer oder EDIT*
Mit diesem einfachen Zeileneditor kann man eine BASIC-Programmzeile ändern, ohne sie komplett neu eingeben zu müssen.

**FIND**
*Befehlsform: FIND „ Zeichenkette", Zeilenbereich*
Der Befehl sucht in einem Programm im angegebenen Zeilenbereich nach der Zeichenkette, wobei zwischen Groß- und Kleinschreibung unterschieden wird.

**RENUMBER**
*Befehlsform: RENUMBER Zeile, Abstand, Startzeile*
Alle Programmzeilen werden mit 10 beginnend mit dem angegebenen Abstand neu durchnumeriert. Selbstverständlich ändert der Interpreter dabei auch die Zeilenangaben hinter den Sprungbefehlen.

**REPLACE**
*Befehlsform: REPLACE ‚alte Zeichenkette' ‚neue Zeichenkette', Zeilennummer oder Zeilenbereich*
Eine bestimmte Zeichengruppe ersetzt eine andere, allerdings nur im angegebenen Zeilenbereich.

**TRACE**
*Befehlsform: TRACE # Kanal Wert*
Ausgabe der Zeilennummern der bearbeiteten Programmzeilen in Spitzklammern am PC-Bildschirm (beispielsweise <10><20><30><145> ...) oder auf den Drucker. Trace 0 schaltet den Befehl ab.

Das EURO-Z80-BASIC ist mittlerweile auch für IBM-kompatible XT- und AT-Computer verfügbar. Der MS/DOS-BASIC-Interpreter bietet geneu den gleichen Befehlsumfang wie die Z80-Version. Ein BASIC-Programm kann damit direkt auf dem Kompatiblen entwickelt, und anschließend einfach auf den EURO-Z80 übertragen werden.

Zum Abschluß noch einige notwendige Änderungen. Wer eine Speichererweiterung durchgeführt hat, benötigt die Adressen für die Anpassung des Debuggers. Das obere Ende des RAM-Bereichs ist zunächst bei $7FFF_{16}$. Mit dem Y-Kommando kann man die Adressen ermitteln. Aber aufpassen, nur einige Adressen sind gültig. In ***Tabelle 8*** sind die Adressen angegeben. Mit dem Quellcode kann dies einfach angepaßt werden. Mit der PAUSE-Taste kann beim AT die Ausgabe des Debuggers (beispielsweise des Interpreters) angehalten werden.

Beim Euro-Z80 gibt es Dank der universellen Eigenschaften des BASIC-Interpreters gleich mehrere Möglichkeiten, einen automatischen Betrieb durchzuführen. Zur jeweiligen Anpassung des Interpreters an die verschiedenen Hardware-Bedingungen gibt es Software für den PC, die eine einfache EPROM-Dateierstellung ermöglicht. Einen Black-Box-BASIC-Interpreter gibt es übrigens auch im 80X86-Code.

## Akku-RAM

Die einfachste Lösung für einen Dauerbetrieb ohne Terminal ist die Verwendung eines akkugepufferten RAM-Bausteins, der ein einmal gespeichertes Programm auch beim Abschalten der Versorgungsspannung konserviert. Dazu eignet sich zum Beispiel der Typ MK48Z08B-20, ein Zeropower-Ram mit integrierten Lithiumzellen. Der Baustein wird einfach anstelle des RAM-Chips IC4 (6264) auf der EPC-Platine eingesetzt.

Das Programm muß nach wie vor einmal mittels Terminalprogramm vom PC geladen werden. Nach einem manuellen Reset am Einplatinencomputer startet das BASIC-Programm mit dem erweiterten Interpreter automatisch. Die V24-Verbindung zum PC kann dann getrennt werden. Einziger Nachteil dieser Lösung: Wenn man den EPC beispielsweise tief unten im Keller als Heizungsfernsteuerung betreibt, muß man nach einem Stromausfall den Reset-Taster betätigen. Aber auch hier läßt sich Abhilfe schaffen.

Listing 1a. Rechtecksignalerzeugung mit BASIC.

```
10 REM EURO-Z80 Rechtecksignal an Pin 1/K9
15 REM Initialisierung nur bei BLACK-BOX
20 REM OUT 7,168  : REM Parallelport 2 8255 Ausgabe PA0
30 OUT 4,1        : REM high-Pegel
40 GOSUB 100
50 OUT 4,0        : REM low-Pegel
60 GOSUB 100
70 GOTO 30        : REM von vorne
75 :
80 REM Zeitschleife
100 FOR I = 1 TO 500: NEXT
110 RETURN
```

## Auto-Reset

Mit einer kleinen Hardwareänderung auf der Platine findet beim Einschalten der Stromversorgung immer ein automatischer Reset statt. Der Interpreter startet dann sofort und beginnt mit dem BASIC-Programm. Die einfachste Lösung für einen automatischen Reset ist folgender Umbau des Euro-Z80: R3 wird aus- und anstelle von R4 eingelötet, C14 durch einen 10-µF-Elko ersetzt, wobei auf die richtige Polung zu achten ist. Der Masseanschluß des Kondensators liegt neben Pin 28 des EPROMs. Der Kondensator lädt sich nach dem Einschalten über den 10-k-Vorwiderstand verzögert auf. Dadurch bleibt an Pin 4 von Gatter N5 (74LS04) zunächt ein Low-Pegel erhalten, was einem gedrückten Reset-Taster entspricht. Natürlich kann ein Reset auch weiterhin manuell ausgelöst werden.

Technisch ist diese Variante nicht optimal, in der Praxis funktioniert sie aber nachweislich fehlerfrei. Besser, aber auch aufwendiger, ist der Einsatz eines Monoflops, beispielsweise eines 74LS121, das auf dem unbestückten Sockelplatz (ZBV) neben IC88 Platz finden kann.
Das in ***Listing 1a*** enthaltene kleine BASIC-Testprogramm dient zum Test der Black-Box-Funktion. Es erzeugt an Pin 1 des Pfostensteckers K9 auf der Euro-Z80-Platine ein Rechtecksignal niedriger Frepuenz, das einfach mit Logiktester oder Zeigermeßinstrument beobachtet werden kann.

Listing 1b. Rechtecksignalerzeugung mit Z80-Maschinensprache.

```
7000 3E 1      LD  A,1        ;Akku mit 1 laden
7002 D3 4      OUT 4,A        ;an Port 4 ausgeben
7004 3E 0      LD  A,0        ;Akku mit 0 laden
7006 D3 4      OUT 4,A        ;an Port 4 ausgeben
7008 18 F6     JR  7000       ;von vorne
```

Eine höhere Frequenz kann durch Programmierung in Z80-Maschinensprache erzeugt werden. Mit dem Monitor (Version 2,0) gibt man mittels E-Kommando zum Beispiel ab Adresse $7000_{16}$ die in ***Listing 1b*** enthaltene Bytefolge ein und startet mit dem G-Kommando.
Allerdings hat diese einfache Methode doch Nachteile, vor allem bei größeren BASIC-Programmen, wenn zum Beispiel bei Robotersteuerung die trigonometrischen Funktionen benötigt werden (siehe Elektorbuch „Fahrbarer Selbstbauroboter").

Nach Murphy tritt immer dort ein Programmfehler auf, wo offenbar alles richtig ist. Eine Fehlermeldung des Interpreters kann aber beim Black-Box-Betrieb nicht ausgegeben werden. Also muß eine BASIC-Interpreterversion verwendet werden, die bei allen Fehlermeldungen zu einer eigenen Routine verzweigt. Dort kann dann zum Beispiel eine Leuchtdiode eingeschaltet werden. Änderungen sind auch bei der Verarbeitung der Programm-Variablen notwendig. Alles dies ist beim Black-Box-Interpreter (der übrigens nur mit Quellcode erhältlich ist) bereits berücksichtigt.

Wenn das Anwendungsprogramm im RAM steht, kann es durch eine Fehlfunktion möglicherweise verändert oder sogar gelöscht werden. Sicherer ist die Unterbringung von Anwendungsprogramm und Interpreter im EPROM. Ein Programmaustausch ist dann durch einen Austausch des EPROMs möglich. Alternativ kann man bei extremen Umweltbedingungen die sogenannten Scheckkarten-EPROMs verwenden, die mit passendem Adapter und Flachbandkabel mit dem Epromsockel verbunden werden.

## Programm im EPROM

Damit beim Euro-Z80 ein BASIC-Programm im EPROM kaufen kann, muß der Autostart-Interpreter im EPROM stehen, das BASIC-Programm mit den passenden Link-Adressen im internen Tokenformat vorliegen und im EPROM ab einer bestimmten Adresse vorhanden sein.

Das PC-Entwicklungssystem für den Black-Box-Interpreter enthält Software wie BASIC-Linker, Lern- und Testprogramme, Z80-Assembler und einen Z80-Emulator. BASIC-Programme beginnen im EPROM ab Adresse $2500_{16}$ und können beim normalen Speicherausbau

des Euro-Z80 bis zur Adresse $3FFF_{16}$ reichen. In der speziellen EPROM-Interpreterversion fehlen alle Befehle und Funktionen, die mit Bildschirm oder Tastatur zusammenhängen, wie beispielsweise INPUT, PRINT, RUN und so weiter.

## BASIC-Linker

Dieses Spezialprogramm für den PC kann BASIC-Programme sowohl als ASCII-Text (zum Beispiel von GW-BASIC), oder im internen Euro-Z80-Format an jede Adresse laden (natürlich außer in den eigenen Bereich) und mit geänderten Linkadressen abspeichern.
Was besagt dies? Eine BASIC-Programmzeile wie zum Beispiel 10 PRINT wird beim Euro-Z80-BASIC wie auch bei vielen anderen BASIC-Interpretern im internen Tokenformat im Speicher abgelegt. Die sogenannten Token sind Schlüssel-Bytes für Befehlswörter. So wird beispielsweise PRINT mit dem Byte $95_{16}$ verschlüsselt. Für Erweiterungen auf bis zu 200 zusätzliche BASIC-Befehlsworte stehen Zwei-Byte-Token zur Verfügung, die vor dem Schlüsselbyte mit dem Byte $880_{16}$ gekennzeichnet sind.

```
Listing 2a. Der Speicherinhalt des Euro-Z80 nach Eingabe
der BASIC-Zeile 10 PRINT ''***''.

6277  83 62 0A 00 95 20 22 2A 2A          ...G....B....''**
6280  2A 22 00 00 00                      *''....
```

```
Listing 2b. Der Speicherinhalt mit gleicher BASIC-Zeile
beim PC
219E:0100  0C 25 0A 00 95 20 22 2A-2A 2A 22 00 00 00 00
```

Das ***Listing 2a*** zeigt den dazugehörigen Speicherinhalt des EURO-Z80 ab Adresse $6277_{16}$. Die ersten beiden Bytes 83 62 sind die sogenannte Linkadresse, also $6283_{16}$. Dort beginnt normalerweise die nächste Programmzeile. Zwei Null-Bytes kennzeichnen an dieser Adresse das Programmende. Die nächsten zwei Bytes sind die Zeilennummer, also 0A 00 für die Zeile 10. In Anführungszeichen ($22_{16}$) folgen nach einem Blank ($20_{16}$) die drei Sternchen ($2A_{16}$).
Mit dem 2.0-Monitor kann man sich das mit dem D-Befehl anzeigen lassen. Nach Eingabe einer beliebigen BASIC-Programmzeile verläßt man mit dem BYE-Befehl den Interpreter und wechselt in den Monitor. Mit dem D-Kommando wird der Speicherinhalt ausgegeben.

## Linken

Wenn das BASIC-Programm im EPROM an einer anderen Adresse liegt, müssen auch die Linkadressen speziell angepaßt sein. Das ***Listing 2b*** zeigt ein Beispiel. Der BASIC-Linker berechnet die Adressen jeweils richtig, hier also $250C_{16}$ (die ersten beiden Bytes $ØC_{16}$ und $25_{16}$). Nach dem Start meldet sich der BASIC-Linker am PC mit der Abfrage nach dem BASIC-Programm-Beginn (***Listing 3***). Mit der eingabe von <CR> (Carriage Return) wird die Adresse $2500_{16}$ verwendet, es kann aber auch eine beliebige Adresse im Bereich von $11_{16}$ bis zum Beginn des Linkers eingegeben werden. Dieser befindet sich ab Adresse $8000_{16}$, damit das BASIC-Programm möglichst weit unten liegen kann. In der Quelle des Linkers können die Adressen als LABEL bequem geändert werden, falls der EPROM-Speicher erweitert wurde.

```
Listing 3. Die Bildschirmausgaben des BASIC-Linkers

VARAM86 D1 (C) Dr. Hehl 25.07.91
aendert BASIC-Linkadressen fuer EURO-Z80
BASIC-Programm laden und wieder abspeichern
RAM ............:  ab 6000h - 7FFFh
Variablen .......:  ab 6280h
Programmspeicher: bis 3FFFh
BASIC-Programm-Beginn: Adresse angeben:?
7168 Bytes Programmspeicher frei
Segmentadresse (dez/hex):7600 1DB0
BASIC-Programmadresse (dez/hex):9472 2500
Fertig
```

Mit LOAD"NAME.BAS" wird nun zum Beispiel das Basic-Testprogramm geladen. Der Dateizusatz.BAS muß angegeben werden. Mit LIST kann man sich das Programm anschauen, mit SAVE"NAME" wieder sichern. Der Dateizusatz .BIN wird automatisch angefügt. Ist eine gleichnamige Datei schon vorhanden, muß mit RESAVE"NAME" gesichert werden.

## EPROM-Erstellung

Am PC können mit einem Debugger der verkleinerte BASIC-Interpreter und das BASIC-Programm an die richtigen Adressen geladen werden. Es hat sich als sehr praktisch erwiesen, den BASIC-Interpreter mit Anfangsadresse $100_{16}$ zu assemblieren. Der im Entwicklungspaket enthaltene Z80-Emulator und -Debugger zeigt dann am PC gleich die richtigen Adressen an. Auch im Einzelschritt kann so getestet werden. Im EPROM muß sich an Adresse 0 ein Sprung zur Adresse $100_{16}$ befinden, also die Bytes C3 00 01 (Z80-Code: JP $100_{16}$). Ab $2500_{16}$ befindet sich das BASIC-Programm mit den korrekten Linkadressen ab $2500_{16}$.

Da Debugger aus CP/M-Zeiten erst ab Adresse $100_{16}$ Programme laden, muß als Ladeadresse immer ein um $100_{16}$ höherer Wert angegeben werden, also $200_{16}$ für den Interpreter. Das BASIC-Programm kommt dann an Adresse $2600_{16}$. Ab Adresse $100_{16}$ steht der Sprungbefehl mit den Bytes C3 00 01. Da der Monitor nicht benötigt wird, muß man die Initialisierung der Portbausteine im BASIC-Programm vornehmen. In ***Listing 1a*** ist in der REM-Programmzeile die Initialisierung beispielhaft angegeben.

Beachten muß man auch, daß im EPROM das Byte vor dem BASIC-Programmbeginn ($(24FF_{16})$ auf Null gesetzt wird, sonst funktioniert der Interpreter nicht. Das WRITE-Kommando des Debuggers speichert dann ab Adresse $100_{16}$ bis zur Endadresse des BASIC-Programms alles ab. Die Datei kann anschließend ins EPROM gebrannt werden.

## Z80-Assembler für den PC

Bei der Entwicklung von Z80-Maschinenspracheprogrammen auf dem PC benötigt man einen Z80-Assembler, der natürlich möglichst preiswert sein soll. Für die Euro-Z80-Vollversion wurde nun der Makro-Assembler Version 4.07 der Firma ZeitControl (Porta-Westfalica) getestet. Er ist auf einem Standard-AT sehr schnell und berücksichtigt weitgehend die allgemeinen Syntax-Regeln üblicher Assembler. Zudem kann er die Opcodes des HD64180

von Hitachi übersetzen, einem verbesserten Nachfolger der Z80-CPU. Wahlweise können Listings, Crossreferenzen und auch ein Intel-HEX-Format erzeugt werden. Eine gute Dokumentation und diverse Beispiele werden mit dem Assembler zum Preis von DM 98,- auf Diskette mitgeliefert.

## Euro-86-BASIC

Inzwischen gibt es den Euro-BASIC-Interpreter auf Diskette auch für den PC im 80X86-Code. Dieser Interpreter mit der Bezeichnung EURO86 läuft unter Debugger-Kontrolle. Nach dem Verlassen des Intepreters (BYE-Befehl) kann das BASIC-Programm im internen Tokenformat direkt mit dem WRITE-Befehl des Debuggers auf Diskette abgelegt werden. Da auch der Quellcode, ebenso wie vom Euro-Z80 erhältlich ist, ist ein Vergleich beider Interpreter auch für Umsteiger von Z80-Maschinensprache auf 80X86-Programmierung interessant. HEBAS86 heißt die Vollversion des EURO86 für den PC und besitzt viele Diskettenbefehle, so auch für sequentielle oder virtuelle (RANDOM)-Dateien. Die Quelle ist ebenfalls beim Autor erhältlich.

Dr. Hans Hehl
Lindenstr. 20
85456 Wartenberg
Tel.: 08762/3070
(16.00 - 20.00 Uhr)