Implementierung von Echtzeitbetriebssystem und PEARL-Compiler auf dem IBM-PC

von
Dr.-Ing. B. Reißenweber, Universität-GH Paderborn, Fachgebiet Prozeßautomatisierung, Pohlweg 47-49, 4790 Paderborn

Im Fachgebiet Prozeßautomatisierung der Universität Paderborn wurde ein PEARL-System, bestehend aus einem Echtzeitbetriebssystem und den zur Programmentwicklung notwendigen Komponenten wie Editor, Compiler, Lader, Bedientask auf dem IBM-PC implementiert. Die charakteristischen Eigenschaften dieses Systems werden beschrieben.
Die besonders hervorstechenden Merkmale im Vergleich mit anderen Systemen sind:

1. Programmentwicklung und Programmausführung benutzen das gleiche Echtzeitbetriebssystem.
2. Der Compiler ist in einer speziell für Compileroperationen entwickelten virtuellen Sprache geschrieben und erzeugt einen ebenfalls virtuellen Objektcode, wobei die dabei verwendete virtuelle Sprache so angelegt ist, daß man ein PEARL-Quellprogramm besonders einfach in diese Sprache übersetzen kann.

1. Ursprung und Umfang

Ende 1980, als preisgünstige PEARL-Systeme für Mikrorechner nicht verfügbar waren, wurde damit begonnen, ein PEARL-System auf einem Rechner, bestehend aus Original-INTEL-SBC-Karten mit dem 8086 als Prozessor, zu implementieren.
Der Ausgangspunkt war ein Stapel Assembler-Listings und etwas Dokumentation zu einem PEARL-System, das W. Gerth auf einem längst vergessenen Krantz-Mulby-Rechner implementiert hatte (siehe /1/).
Das davon abgeleitete PEARL-System für den 8086 läuft seit etwa drei Jahren innerhalb des Fachgebietes und wird für Lehre (Vorlesungen über PEARL mit Übungen am Gerät, Studien- und

Diplomarbeiten) und Forschung (Drittmittelprojekte, Dissertationen) eingesetzt.
Mit den verschiedenen IBM-PC's bietet sich eine Möglichkeit, das PEARL-System mit relativ geringem Aufwand auf eine wesentlich preisgünstigere (bei dem PC und PC XT allerdings langsamere) Hardware zu übertragen.
Was den Sprachumfang der Implementation anbelangt, so wurde weitgehend der Umfang von Basic-PEARL nach DIN 66253 realisiert. Es fehlen nur einige für Spezialanwendungen benötigte Elemente bei der Ein- und Ausgabe wie die absoluten Positionierungselemente COL, LINE, POS. Bei der Signal-Reaktion wird nach ON nur eine GOTO-Anweisung für einen Sprung innerhalb des Blocks, in dem man sich gerade befindet, zugelassen.
Als Erweiterung über Basic-PEARL hinaus wurden Bereiche mit dynamischen Obergrenzen eingeführt. Ihr Einsatz ist besonders vorteilhaft bei Bibiliotheks-Programmen, die je nach Anwendungsfall mit Bereichen der unterschiedlichsten Größe arbeiten müssen.

2. Hardware-Voraussetzungen

Das hier beschriebene PEARL-System läuft derzeit auf dem einfachen IBM-PC mit oder ohne Erweiterungsbox und damit auch mit oder ohne Festplatte und auf dem IBM-PC XT. Wir haben uns bisher auf Original IBM-PC's beschränkt; in wieweit das System auf den "Kompatiblen" lauffähig ist, wurde nicht untersucht.
Zentrale Voraussetzung für die Implementierung unseres PEARL-Systems ist aber die Interrupt-Struktur des Original-IBM-PC's, d.h. es muß der Interrupt-Controller-Baustein INTEL 8259 vorhanden sein und die Interrupt-Request-Leitungen müssen über den Bus geschleift sein.
Bei den Zusatzkarten bzw. Adaptern verwenden wir allerdings einige Produkte anderer Firmen wie z.B. die Herkules-Monochrome-Grafikkarte. Wegen der geringen Anzahl von Steckplätzen im PC-Gehäuse ist es sinnvoll, eine Multifunktionskarte einzusetzen, die üblicherweise mit RAM-Speicher, serieller und paralleler Schnittstelle und Uhr mit Kalender ausgerüstet ist.
IBM bietet keine Möglichkeit an, den PC mit Prozeßperipherie, d.h. mit analoger und digitaler Ein-/Ausgabe zu versehen. Wir

verwenden Karten von Data Translation; sie bestehen z.B. aus einer Trägerkarte mit drei Steckplätzen, ausgelegt für den von INTEL entwickelten SBX-Bus, und entsprechenden SBX-"Huckepack-Karten" jeweils für 16 analoge Eingänge, 8 analoge Ausgänge und 24 digitale Ein-/Ausgänge.
Bei den Geräten der Standardperipherie wird vom System die deutsche Original-IBM-Tastatur unterstützt. Die meisten Bildschirme und Drucker sind für die hier verwendeten Funktionen identisch, sodaß fast jedes Gerät verwendet werden kann. Über eine serielle Schnittstelle kann eine Rechnerkopplung hergestellt oder ein PROM-Programmierer oder ein Plotter 7475 von Hewlett-Packard angeschlossen werden.
Der gesamte Verkehr mit den Peripheriegeräten wird vom Betriebssystem selbständig erledigt, also ohne das ROM-Basic-Input-Output-System (BIOS) zu benutzen, da das BIOS für einen leistungsfähigen Echtzeitbetrieb nicht geeignet ist. Nur beim Booten tritt das ROM-BIOS in Aktion, was man nur durch Auswechseln der ROM's vermeiden könnte.

3. Start des Systems

Beim Start des PEARL-Systems sind zwei Vorgehensweisen denkbar. Will man den PC auch für Programme, die das DOS-Betriebssystem voraussetzen, wie etwa Textverarbeitungssysteme, benutzen, so ist es wohl sinnvoll, den Boot-Mechanismus des PC unverändert zu lassen, sodaß sich beim Booten das DOS-Betriebssystem meldet. Das PEARL-System kann man dann wie ein anderes Programm von Diskette oder Harddisk laden und zur Ausführung bringen. Das PEARL-System setzt dann allerdings das DOS-System außer Betrieb, indem es zunächst die Interruptvektoren umlädt. Außerdem wird der Speicherbereich, den das DOS belegt, als Anwenderspeicher genutzt, DOS also überschrieben.
Die zweite Möglichkeit beim Booten ist, daß man eine Diskette oder Disk in den dafür vorgesehenen Sektoren mit einem Programm versieht, das das PEARL-System direkt lädt.
Eine dritte Möglichkeit, die aber leider bei der derzeit für den IBM-PC lieferbaren Hardware praktisch ausscheidet, besteht darin, das PEARL-System in PROM's auf einer Adapterkarte

abzulegen, sodaß der vom BIOS ausgeführte Urladevorgang sich auf das Laden eines Programmes, das im wesentlichen nur aus einem Sprung in das PEARL-System besteht, beschränken könnte.
Beim Laden des Systems in den bis zu 640 kByte großen RAM-Speicher ist es in jedem Falle bei dem relativ kleinen Umfang des PEARL-Systems von derzeit etwa 80 kByte am einfachsten, das gesamte System auf einmal zu laden und ständig im Speicher resident zu halten.

4. Beschreibung der Systemprogramme

Die 80 kByte Systemprogramme enthalten alles, was zur Entwicklung und Ausführung von PEARL-Programmen benötigt wird. Ein Hin- und Herschalten zwischen zwei Betriebssystemen für die Programmentwicklung und die Programmausführung entfällt also.Die einzelnen System-Komponenten werden im folgenden beschrieben.

a) Betriebssystem-Kern

Er enthält zunächst die für ein Echtzeit-Betriebssystem charakteristischen Elemente:

1. Verwaltung des Prozessors, damit zu jedem Zeitpunkt die lauffähige Task mit der höchsten Priorität bearbeitet wird,
2. Verwaltung der Interrupts, die entweder vom PC selbst intern erzeugt werden oder vom Anwender von außen aufgegeben werden (zu den PC-internen Interrupts gehören z.B. der Interrupt von einem Zeitgeber, mit dem die Echtzeituhr realisiert wird),
3. Realisierung von Synchronisationsmechanismen, wie sie insbesondere durch die Semas gegeben sind,
4. Ausführung der Befehle, mit denen bei einer Task ein Zustandswechsel herbeigeführt wird (z.B. Aktivieren oder Terminieren),
5. Verwaltung des Arbeitsspeichers, wobei im vorliegenden System die Speicherzuteilung zum Laufzeitpunkt erfolgt, d.h. erst beim Start einer Task bzw. beim Aufruf einer Prozedur wird der Speicher für die lokalen Variablen zugewiesen und am Ausführungsende wieder freigegeben. Mit diesem Speicherverwaltungskonzept sind z.B. auch rekursive Prozeduren möglich.

Wenn andere Systemprogramme die Dienste des Kerns in Anspruch

nehmen wollen, so geschieht dies stets über einen Software-Interrupt, dem INT-Befehl des 8086. Da dieser Befehl automatisch mit genereller Interruptsperre abläuft, ist jeder Zugriff zum Kern exklusiv.
Zum Kern kann man auch die Programme zählen, die die Basis für die Ein- und Ausgabe mit der Standard- und der Prozeßperipherie schaffen. Während die Ein- und Ausgabe über Prozeßperipherie ohne merkliche Wartezeiten abläuft und deshalb direkt durch einen Software-Interrupt abgewickelt werden kann, wird bei der Ein- und Ausgabe über Standardperipherie nur die Übergabe des Auftrags mit Software-Interrupt erledigt. Die eigentliche Ein-/Ausgabe-Operation wird dann von einer System-Task, die diesem Gerät zugeordnet ist, angestoßen. Sobald die Gerätetask über einen Zeitraum warten müßte, der im Normalfall eine Millisekunde oder länger dauert, suspendiert sie sich und gibt damit den Prozessor für andere Aufgaben frei. Sobald die Ein-/Ausgabe-Operation fortgesetzt bzw. abgeschlossen werden kann, wird ein gerätespezifischer Interrupt ausgelöst. Die dazugehörige Interrupt-Serviceroutine bewirkt eine Fortsetzung der Gerätetask.
Es soll noch kurz auf die Gerätetask, die den Verkehr mit Floppy oder Disk ermöglicht, eingegangen werden. Diese Task beschränkt sich nicht auf den einfachen hardwaremäßigen Transfer, sondern sie verwaltet die gesamte File-Struktur des Speichermediums. Um einen Datentransfer zwischen PEARL- und DOS-System zu ermöglichen und um bei der Festplatte ohne Partitionierung auszukommen, wurde das DOS-Disketten- und Disk-Format exakt nachgebildet, einschließlich der Möglichkeit geschachtelte Directories bzw. Subdirectories zu verwenden.

b) Editor

Der Editor in seiner derzeitigen Form ist in Anlehnung an den Turbo-Pascal-Editor entstanden, der seinerseits wieder große Ähnlichkeit mit dem Textverarbeitungsprogramm Wordstar hat. Bei etwa gleichen Möglichkeiten wurde nur die Bedienung etwas gestrafft, um die Anzahl von Steuerzeichen, die man sich merken muß, zu reduzieren. Da das PEARL-System insbesondere bei der

PEARL-Ausbildung von Studenten ohne Vorkenntnisse benutzt wird, wurde bei der Bedienung immer großer Wert auf Einfachheit gelegt.

c) Compiler

Er ist entstanden aus einem Compiler, den W. Gerth für den Krantz-Mulby geschrieben hat, und er ist heute über weite Strecken nahezu identisch mit dem Gerthschen Compiler für den Motorola 68000 (siehe /2/). Diese Unabhängigkeit des Compilers vom Rechnertyp hat ihre Ursache in zwei ganz wesentlichen Eigenschaften:

1. Der Compiler ist in einer virtuellen und damit rechnerunabhängigen Sprache geschrieben. Diese virtuelle Sprache ist von W. Gerth speziell für die Programmierung von Compilern und Assemblern definiert worden und ist ganz zugeschnitten auf die dabei auftretenden Aufgaben wie etwa das Arbeiten mit Listen. Bei der Übertragung des Compilers auf einen anderen Rechner muß nur der relativ kurze Interpreter neu geschrieben werden, während der virtuelle Compilercode, der vom Interpreter zum Übersetzungszeitpunkt auszuführen ist, unverändert bleibt.
2. Der deutlich größere Teil des Compilers führt die syntaktische Prüfung des Quellprogrammes durch, zerlegt eine PEARL-Anweisung in einzelne kleine Schritte und legt dieses Ergebnis in einer Code-Liste ab. Ein recht kleiner Teil des Compilers, der Code-Generator, holt diese abstrakte Information aus der Code-Liste und bildet den für den jeweiligen Rechner geeigneten Code. Dieses Produkt des Compilers kann teilweise aus realem Code, d.h. Maschinenbefehlen des Rechners, bestehen oder es kann wiederum virtuelle Befehle enthalten. Die dabei zugrundegelegte virtuelle Sprache wird so definiert, daß es besonders einfach ist, ein PEARL-Quellprogramm in diese Sprache zu übersetzen.

Bei dem hier beschriebenen System wurde auf ein Übersetzen direkt in Maschinenbefehle und auf eine direkte Nutzung der Register des 8088 ganz verzichtet, sodaß der Compiler nur virtuellen Code erzeugt. Der 8088 besitzt nur sieben 16 Bit lange Register, von denen oft mindestens zwei Register als

Indexregister zur Speicheradressierung eingesetzt werden müssen. Es ist deshalb nicht möglich, z.B. Zwischenergebnisse beim Ausrechnen eines FLOAT-Ausdrucks (gar bei 64 Bit langen FLOAT-Größen) in Registern abzulegen, auch wenn dies ein Rechenzeit sparendes Verfahren wäre.
Große Probleme bereitet auch die spezielle Adressierungsform des 8088 mit Hilfe von Segmentregistern. Die nach außen abgegebene 20-Bit-Adresse wird gebildet, indem der 16-Bit-Inhalt des verwendeten Segmentregisters um 4 Bit nach links verschoben und zu dem 16-Bit-Offset, der z.B. der Inhalt eines Index-Registers sein kann, addiert wird. Der gesamte Bereich wird so in Segmente eingeteilt, die entsprechend dem 16-Bit-Offset maximal 64 kByte groß sein können. Vor jedem Speicherzugriff muß das verwendete Segmentregister eingestellt sein, wobei für den Datenzugriff bestenfalls zwei Segmentregister zur Verfügung stehen.
Die einfachste Möglichkeit wäre nun, den ganzen für den Anwender verfügbaren Speicher auf 64 kByte zu begrenzen; dann kann man die Segmentregister einmal beim Start des Anwenderprogramms einstellen und dann stets beibehalten. Die Adressierung könnte sich auf den 16-Bit-Offset beschränken. Bei dem beim PC hardwaremäßig verfügbaren RAM-Speicher bis maximal 640 kByte wäre dies allerdings eine harte Einschränkung. Es wurde deshalb das PEARL-System so eingerichtet, daß wohl der Code-Bereich eines Moduls für sich, die statischen Daten eines Moduls für sich, die lokalen Variablen jeder einzelnen Task oder Prozedur und jeder einzelne Bereich mit dynamischen Obergrenzen jeweils auf 64 kByte begrenzt sind, also jeweils durch eine Segmentregistereinstellung adressiert werden können, daß aber die Gesamtheit des Codes und der Daten den gesamten hardwaremäßig zur Verfügung stehenden RAM-Bereich (abzüglich des Platzes für die Systemprogramme) nutzen kann.
Da schließlich der Befehlssatz des 8088 nicht sonderlich komfortabel ist (insbesondere ist der 8088 von wenigen Ausnahmen abgesehen eine Ein-Adreß-Maschine), wird z.B. die recht einfache Addition einer 32-Bit-FIXED-Konstanten zu einer Variablen gleichen Typs schon etwas länglich:

```
MOV AX,CS               ;Segmentregister DS laden mit dem Inhalt
MOV DS,AX               ;des Codesegmentregisters CS
```

```
MOV ES,BASIS              ;Basis des lokalen Workspace in
                          ;Segmentregister ES
MOV SI,OFFSET KONSTANTE ;Offset der Konstante in Indexregister SI
MOV DI,OFFSET VARIABLE  ;Offset der Variablen in Indexregister DI
LODSW                     ;durch DS und SI adressiertes Low-Wort
                          ;der Konstante in Register AX, SI=SI+2
ADD ES:(DI),AX            ;AX zu Low-Wort der durch ES und DI
                          ;adressierten Variablen addieren
LODSW                     ;High-Wort der Konstanten nach AX
ADC ES:(DI+2),AX          ;und zu High-Wort der Variablen addieren
                          ;(mit Carry)
JNO MARKE                 ;Sprung, wenn kein Overflow aufgetreten
MOV AX,'A1'               ;Fehlerkennung, die auf Bildschirm
                          ;ausgegeben werden soll
INT 89                    ;Aufruf der Fehlerreaktion durch den
                          ;Betriebssystem-Kern
MARKE: ...                ;nächste Operation
```

Für diese eine Addition werden 30 Bytes Maschinencode benötigt, während man bei der verwendeten virtuellen Codierung mit 6 Bytes auskommt.
Ein solcher virtueller Befehl besteht aus einer Befehlsnummer, einem Adressierungsbyte und den Offsets von maximal 4 Operanden (bei der Adressierung von statischen oder globalen Variablen besteht der Operand aus Offset- und Segmentwert). Für die Addition der 32-Bit-FIXED-Größe sieht der virtuelle Befehl so aus:

Byte 0 : Befehlsnummer: 5

Byte 1 : Adressierungsbyte: 00100000B

Jeweils 2 Bits gehören zu maximal 4 Operanden; die beiden höchstwertigen gehören zum 1. Operanden, usw. Die Bits geben an, in welchem Segment die Operanden stehen; es bedeutet:

00B Operand im lokalen Workspace einer Task oder Prozedur,

01B indirekte Adressierung: unter dem angegebenen Offset im lokalen Workspace stehen Offset- und Segmentwert des Operanden,

10B Operand im Code-Bereich,

11B die Operandenangabe besteht aus Offset- und Segmentwert.

Bytes 2,3: Offset der Variablen im lokalen Workspace
Bytes 4,5: Offset der Konstanten im Code-Bereich

Die virtuelle Codierung bringt also eine deutliche Reduzierung des für den Objektcode benötigten Speicherplatzes. Die Code-Generierung durch den Compiler wird deutlich einfacher, der Compiler damit auch kürzer. Es entsteht allerdings eine Laufzeitverlängerung dadurch, daß bei der Interpretation in einem für alle Befehle gemeinsamen Programmstück entsprechend den Adressierungsbits erst der richtige Segmentwert ausgesucht und in Abhängigkeit von der Befehlsnummer zur eigentlichen Befehlsausführung gesprungen werden muß. Dieser Zeitverlust fällt aber umso weniger ins Gewicht, je komplexer der virtuelle Befehl ist.

d) Lader

Da der Compiler nur einen Durchlauf macht und damit Vorwärtsreferenzen nicht erfüllen kann, und da auf einen eigenen Linker verzichtet wurde, hat der Lader folgende Aufgaben:

1. Im Anwenderspeicher den Platz für den Modul belegen,
2. den Code und die initialisierten statischen Variablen laden,
3. an allen Stellen, an denen der Compiler die Adresse noch nicht kannte und nur eine Information an den Lader einsetzen konnte, die endgültige Adresse eintragen. Dies tritt auf bei Vorwärtsreferenzen, bei der Ansprache von statischen Variablen (erst der Lader weiß, welchen Segmentwert diese Variablen bekommen haben) und bei der Referenz von globalen Größen aus anderen Modulen.

e)Laufzeitinterpreter

Zur Ausführung des virtuellen Codes, den der Compiler erzeugt hat, muß ein Interpreter vorhanden sein. Dieser zum Laufzeitpunkt eines Anwenderprogrammes aktive Interpreter führt virtuelle Befehle aus mit den verschiedensten Aufgaben, die sich aus der Übersetzung eines PEARL-Programmes ergeben. Ein großer

Teil dieser Befehle ist natürlich notwendig, um die Anwendung der verschiedenen Operatoren auf die sechs erlaubten Datentypen und um die formatierte Ein- und Ausgabe zu realisieren.

f) Bedientask

Die Bedientask meldet sich beim Start des Systems und führt den Dialog mit dem Benutzer. Ihre erste Aufgabe ist, die für die Programmentwicklung notwendigen Systemprogramme (Editor, Compiler und Lader) aufzurufen.
Die Kommandoeingabe läuft generell im Wechselspiel ab: Der Benutzer gibt mit der Tastatur vier Zeichen ein (abgeschlossen durch die RETURN-Taste), womit des Kommando bezeichnet wird. Die Bedientask fordert dann durch Ausgabe eines Kennwortes zur Eingabe der 1. Zusatzinformation auf. Ist die Eingabe erfolgt und mit RETURN abgeschlossen, wird die nächste Zusatzinformation angefordert, usw.
Beim Compiler-Aufruf beispielsweise fragt die Bedientask nach der Eingabe COMP mit FROM nach dem File mit dem Quellprogramm. Die Eingabe F1:MESPRO.SRC bezeichnet dann das File MESPRO.SRC im Laufwerk 1 der Floppy. Mit der Ausgabe LIST fordert die Bedientask die Geräte- oder Filebezeichnung für die Ausgabe des Listings an. Die Antwort wird häufig LP: oder VT:/E sein. Im ersten Fall wird das Quellprogramm mit Zeilennummern, mit Angabe der Fehler direkt im Text und mit Zusatzinformation wie Compilierzeitpunkt, Größe des entstandenen Code- und Datenbereichs ergänzt und auf dem Drucker ausgegeben. Im zweiten Fall bewirkt der durch "/" abgetrennte Switch "E", daß nur fehlerhafte Zeilen mit Angabe des Fehlers und die Zusatzinformation auf dem mit VT angesprochenen Bildschirm ausgegeben werden. Als weiterer wichtiger Switch ist "O" zugelassen; er bewirkt eine Ausgabe des Offsets aller im Modul verwendeten Variablen, was für die Testphase wichtig ist.
Die Bedientask fragt dann noch mit OBJECT nach einer File- oder Gerätebezeichnung für den Objektcode. Wird hier nur die RETURN-Taste gedrückt, so wird kein Objektcode erzeugt. Die Eingabe D0:MESPRO.OBJ bewirkt, daß der Objektcode im File MESPRO.OBJ im Laufwerk 0 der Disk abgelegt wird. Beim Objektcode kann eben-

falls ein durch "/" abgetrennter Switch, und zwar "L", eingegeben werden, was aber nur bei ausgetesteten und fehlerfreien Programmen sinnvoll ist. Der Compiler legt nämlich beim Beginn jeder Zeile, sofern es die erste Zeile einer Anweisung ist, einen virtuellen Befehl mit der Zeilennummer als Operanden ab; dies kann durch den Switch "L" verhindert werden. Dieser Zeilennummerbefehl bietet aber einerseits die Möglichkeit, bei einer Fehlermeldung durch das System die Zeilennummer mit auszugeben und so die Lokalisierung des Fehlers möglich zu machen. Außerdem ist dieser Befehl die Basis für die Fähigkeit der Bedientask, Breakpoints, die an den Zeilennummern des Listings orientiert sind, zu setzen.
Dies ist die zweite Aufgabe der Bedientask, daß sie nach Abschluß des Ladevorganges die Möglichkeit bereitstellt, Breakpoints zu setzen und Tasks zu aktivieren. Beim Erreichen eines Breakpoints kann man eventuell neue Breakpoints setzen, Variablen anschauen und ändern (dazu braucht man die mit dem Switch "O" erzeugten Offsetlisten; die Ausgabe der Variablenwerte auf dem Bildschirm und ihre Änderung erfolgt in dem Datenformat der angesprochenen Variable). Dann kann man mit weiteren Kommandos die angehaltene Task fortsetzen oder auch terminieren.
Der dritte Komplex bei den Bedientask-Kommandos befaßt sich mit "Disk- und Disketten-Operationen". Es gibt Kommandos zum Formatieren einer Diskette, zum Anlegen oder Löschen von Subdirectories, zum Wechseln der aktuellen Directories, zum Löschen und Umbenennen von Files und vorallem zum Kopieren von Files, wobei neben dem Kopieren innerhalb des Disk-Disketten-Bereichs auch ein Kopieren von Files auf Drucker, Bildschirm und Rechnerkopplung möglich ist und umgekehrt auch von der Rechnerkopplung in ein File kopiert werden kann.
Die Kommandos, die ohne den Benutzer ausgeführt werden können, wie z.B. Compilieren und Laden können auch im Stapelbetrieb erledigt werden; dazu wird mit dem Editor ein Steuerfile erstellt und dessen Ausführung mit einem Bedientask-Kommando gestartet.

g) Bibliothek

Den Abschluß der Systemprogramme bildet eine Bibliothek, die vorallem Funktionsprozeduren für mathematische Funktionen wie Quadratwurzel, Exponentialfunktion, Logarithmus und trigonometrische Funktionen enthält sowie Prozeduren, die eine wahlweise graphische Ausgabe von Ergebnissen auf dem Bildschirm oder auf dem Plotter ermöglichen.

5. Ausblick

Da die Ausführungsgeschwindigkeit des IBM-PC und PC XT nicht berauschend ist, liegt es nahe, das System auch auf den IBM-PC AT zu übertragen. Leider sind XT und AT in vielen Punkten nicht kompatibel, sodaß die Hardwareansprache ganz neu geschrieben werden muß. Dies soll als nächstes in Angriff genommen werden.

Literatur

/1/ Gerth,W.: Ergebnisse einer Basic-PEARL-Implementierung für Kleinrechner. Fachtagung Prozeßrechner, München 1981. S. 404-412.

/2/ Gerth,W.: PEARL-Compiler, Assembler, Editor. c't 1985, H. 4, S. 62-65.