Wolfgang Mayer-Gürr

Pascal auf dem EMUF08

Entwicklungsumgebung Kat-68

Mit nur einem EPROM vom Typ 27512 (64 KByte) hat man im EMUF08 fast alles zur Verfügung, was der Programmierer braucht: Monitor, bidschirmorientierten Editor, Assembler für 68000 und einen vollständiger Pascal-Compiler. Da der EMUF über keinen eigenen Massenspeicher verfügt, ist zum Betrieb noch ein IBM-PC/XT/AT, Apple-II, Macintosh, Atari ST, Basis 108 oder ein Schneider PC nötig, der diese Aufgabe übernimmt.

Im RAM des EMUF können Programme editiert, assembliert, übersetzt und ausgeführt werden. Lediglich zur Datensicherung schreibt man Programme auf die Floppy Disk des anderen Rechners, der sonst nur als Terminal fungiert.

Die Hardware

Kleinere Programme können mit dem Standard-RAM-Ausbau des EMUF08 ohne Schwierigkeit erstellt werden, bei größeren Projekten stellen die 32 KByte aber eine wesentliche Einschränkung dar. Doch ist eine Aufrüstung des EMUF08 auf 64, 96 oder 128 KByte keine große Tat: Mit einem, zwei oder drei statischen RAMs vom Typ 43256 bzw. 62256 ist sie ganz einfach durchzuführen. Auf dem Lochrasterfeld findet sich genügend Platz für drei 28polige Sockel, die Daten- und Adreßleitungen der CPU 68008 liegen auf der 64poligen Steckerleiste direkt am Lochrasterfeld. Man verbindet einfach D0 des Steckers P mit D0 der RAMs, D1 mit D1 usw. Mit dem Adressbus (A0...A14) verfährt man in derselben Weise, R/$\overline{W}$ kommt an R/$\overline{W}$ des RAM, nur $\overline{OE}$ vom RAM wird an GND gelegt. Bleiben noch die Select-Eingänge der RAMs: Da das vorhandene RAM durch die Leitung $\overline{CSRAM}$ (siehe Schaltbild mc 2/87) in den untersten 64 KByte des EMUF08-Adressraumes zweimal eingeblendet wird, muß Pin 20 des RAM (IC1) von $\overline{CSRAM}$ getrennt und neu selektiert werden (unter IC14). In *Tabelle 1* ist die Belegung des Steckers P aufgeführt. In mc 2/87 waren hier nämlich Pin 1 mit 2, 3 mit 4 usw. vertauscht.

Das System Kat-68

Kat-68 ist ein Entwicklungssoftware-Paket für Computer mit einem Prozessor der Familie 68000. Besonders geeignet sind auch Einplatinencomputer wie der EMUF08 oder das „doppelte Lottchen", der mc-65816-Computer mit der CPU 68008, die für Steuerungs- oder Trainingszwecke eingesetzt werden sollen. Für die Programmentwicklung ist noch ein Stammrechner erforderlich, der lediglich Tastatur, Bildschirm und Massenspeicher zur Verfügung stellt. Gängige Terminalprogramme sind hierfür nicht geeignet.

Tabelle 1: Die Belegung des Steckers P

2	VCC	1	VCC
4		3	
6	GND	5	GND
8		7	
10	$\overline{BERR}$	9	$\overline{CSSYN}$
12	$\overline{VECTIRQ}$	11	CLOCK
14		13	
16		15	$\overline{IRQAKN}$
18		17	$\overline{VPA}$
20	$\overline{AS}$	19	$\overline{CS1}$
22		21	$\overline{POWERFAIL}$
24		23	$\overline{CS2}$
26		25	$\overline{CS3}$
28	$\overline{RESET}$	27	
30		29	$\overline{HALT}$
32	A18	31	A19
34	IPL1	33	IPL0/2
36	A16	35	A17
38	A14	37	A15
40	A12	39	A13
42	A10	41	A11
44	A8	43	A9
46	A6	45	A7
48	A4	47	A5
50	A2	49	A3
52	A0	51	A1
54	D6	53	D7
56	D4	55	D5
58	D2	57	D3
60	D0	59	D1
62	R/$\overline{W}$	61	
64	GND	63	GND

Um die Arbeit zu erleichtern, fordert das Kat-68 Programm den Stammrechner auf, Daten von der Tastatur oder der Diskette zu senden. Neben der Übertragung von Tastatureingaben muß der Stammrechner über eine direkte Cursoradressierung verfügen. Die Diskettenoperationen laufen grundsätzlich über Dateien mit wahlfreiem (random) Zugriff. Da zur Kommunikation eine serielle Schnittstelle dient, lassen sich fast alle Computertypen als Stammrechner verwenden. Eine Liste der Steuerbytes zeigt *Tabelle 2*.

Schließt man den Stammrechner an den Einplatinencomputer an, so sendet das Terminalprogramm so lange das Byte $0D (Carriage Return), bis Kat-68 die Baudrate des Senders erkannt und sich auf diese Übertragungsgeschwindigkeit eingestellt hat. Beim EMUF ist die Baudrate durch Jumper eingestellt, jedoch kann der Teiler des 6850 zwischen 1:16 oder 1:64 umgeschaltet werden. Ansonsten ist die Baudrate nicht einstellbar. Das Maximum liegt bei 38400 Baud.

Ist die Verbindung zustande gekommen, wird über ein Befehlsbyte die Bildschirmgröße (Zeichen pro Zeile und Zeilen pro Bildschirm) erfragt. Diese Angaben werden für den Editor und die Ausgabe von Ergebnissen gebraucht.

Erste Schritte

Die erste Hürde ist genommmen, wenn das Hauptmenü von Kat-68 auf dem Bildschirm erscheint, nun kann es an die Eingabe eines ersten Programms gehen – wie so oft soll auch hier die Ausgabe eines Satzes auf dem Bildschirm als Beispiel dienen.

Zur Eingabe des Programmtextes braucht man den Editor. Seine Bedienung erfolgt über Control-Tasten, hier hat Wordstar einen gewissen Standard gesetzt, so daß die Umgewöhnung nicht schwer fällt. Der Editor ist ein reiner Programmeditor, daher fehlen Routinen zum Formatieren des Textes. Es gibt aber Blockbefehle und selbstverständlich Such- und Austauschoperationen. Eine Liste der Befehle zeigt *Tabelle 3*.

Bei der ersten Beschäftigung mit einer Assemblersprache ist man sicher dankbar, wenn man auf einige Standardrouti-

nen zurückgreifen kann. Hierzu gehören Ein- und Ausgaberoutinen des Betriebssystems. Auch bei Kat-68 können solche Routinen benutzt werden.
Da sich die Lage dieser Unterprogramme bei späteren Versionen ändern kann, verwendet man Sprungleisten. Der Prozessor 68000 bietet als elegantere Lösung die Trapbefehle. Übergibt man einen oder mehrere Parameter per Stack oder Register, können auf einfache Art Routinen ausgewählt werden. In KAT-68 sind beide Zugriffsarten, also Sprungtabelle (*Bild 1*) und TRAP-Befehl (*Bild 2*) möglich.

Der Assembler

Die Übersetzung von Maschinensprache erfolgt in zwei Durchläufen und ist sehr schnell. Ist keine ORG-Adresse angegeben, so wird der erzeugte Maschinencode im Speicher abgelegt und kann sofort ausgeführt werden. Der Code kann auch in eine Datei auf Diskette geschrieben werden. Mit einer Option kann die Bildschirm-Ausgabe abgeschaltet oder auf einen Drucker umgeleitet werden.
Tritt beim Übersetzen ein Fehler auf, so wird eine Fehlermeldung ausgegeben und auf einen Tastendruck gewartet. Die Übersetzung kann dann abgebrochen werden. In diesem Fall wird der Editor aufgerufen, der Quelltext angezeigt und der Cursor auf das Ende der fehlerhaften Stelle positioniert.
Ist eine Datei länger als der verfügbare Speicher, lassen sich Quelltextblöcke über eine CHAIN Anweisung oder Include-Dateien verketten. Label dürfen beliebig lang sein, gültig sind aber nur die ersten neun Stellen.
Ist ein neues Programm übersetzt und gestartet, dann läßt der Erfolg oft auf sich warten, weil irgendwo ein Denkfehler vorliegt. Deshalb muß ein Entwicklungssystem Unterstützung bei der Fehlersuche bieten. Bei dem Kat-68 System ist das ganz einfach.

Debugging

Ein Programm läßt sich dazu in Einzelschritten abarbeiten: Es wird in seiner Ausführung nach jedem Maschinenbefehl angehalten. Dann werden die Adresse des Programmzählers, der disassemblierte Maschinencode und der Inhalt aller Register einschließlich des Statusregisters angezeigt.
Ist ein Programm für eine solche Einzelschrittverarbeitung zu lang, kann man im Quelltext den zu untersuchenden Teil mit TRAP #1 und TRAP #0 „einklammern". Wird der Beginn dieser

Tabelle 2: Steuerbytes, eingeleitet durch das Byte $11

$00	; Setup Sende maximale X- und Y-Koordinate des Bildschirms
$01	; UnitOut lies nächstes Byte und sende es zum Drucker.
$02	; Unitinit initialisiere Druckerschnittstelle.
$03	; BlockRead lies 1 Byte Filenummer und 2 Bytes Blocknummer lies den dadurch spezifizierten Block (512 Bytes) von der Diskette und sende ihn an Kat-Rechner
$04	; BlockWrite lies 1 Byte Filenummer und 2 Bytes Blocknummer lies 512 Bytes vom Kat-Rechner und schreibe diesen Block an die entsprechende Stelle auf der Diskette
$05	; Resetfile lies vom Kat-Rechner 1 Byte Filenummer 1 Byte Namenlänge sowie N-Bytes Filenamen Öffne File(Filename) zum Lesen
$06	; RewriteFile Lies vom Kat-Rechner 1 Byte Filenummer Öffne File(Filename) zum Schreiben
$07	; CloseFile Lies vom Kat-Rechner 1 Byte Filenummer schließe das zugehörige File
$08	; CloseDeleteFile Lies vom Kat-Rechner 1 Byte Filenummer lösche das zugehörige File
$09	; IoResult sende das IoResult-Byte der letzten Disketten- Operation an Kat-Rechner
$0A	; GotoXY lies vom Kat-Rechner das X-Koord-Byte und das Y-Koord-Byte. Setze den Cursor an diese Stelle des Bildschirms. X=0,Y=0 entspricht der oberen linken Ecke des Schirms
$0B	; Quit beende das Terminalprogramm

Tabelle 3: Befehle des Editors

Cursorbewegungen:	
CTRL S	1 Zeichen nach links
CTRL H	linkes Zeichen löschen
CTRL D	1 Zeichen nach rechts
CTRL A	1 Wort nach links
CTRL F	1 Wort nach rechts
CTRL E	1 Zeile nach oben
CTRL X	1 Zeile nach unten
CTRL W	Scroll up
CTRL Z	Scroll down
CTRL R	1 Seite nach oben
CTRL C	1 Seite nach unten
CTRL Q S	zum Zeilenanfang
CRTL Q D	zum Zeilenende
CTRL Q E	zum Seitenanfang
CTRL Q M	Seite zentrieren
CRTL Q R	zum Textanfang
CRTL Q ⟨Anz⟩	Seiten zurück
CTRL Q C	zum Textende
CTRL Q ⟨Anz⟩	Seiten vorgehen
CTRL Q B	zum Blockbeginn
CTRL Q K	zum Blockende
CTRL Q P	zur letzten Cursorposition
Einfügen und löschen:	
CTRL V	Einfügemodus an/aus
CTRL N	Zeile einfügen
CTRL Y	Zeile löschen
CTRL Q Y	Rest der Zeile löschen
CTRL T	Wort rechts löschen
Blockkommandos:	
CTRL K B	Blockanfang markieren
CTRL K K	Blockende markieren
CTRL K C	Block kopieren
CTRL K V	Block versetzen
CTRL K Y	Block löschen
CTRL K R	Block von Diskette lesen
CTRL K W	Block von Diskette schreiben
sonstige Kommandos	
⟨ESC⟩	Verlassen des Editors
CTRL Q V	Einrückmodus an/aus
CTRL I	Tab rechts
CTRL J	Tab links
CTRL Q⟨n⟩J	Tab-Distanz setzen
CRTL Q⟨n⟩F	n-tes Wort suchen
CTRL Q⟨n⟩A	n-Worte suchen und ersetzen
CTRL P	Ersatztaste f. Controlzeichen definieren
CTRL Q L	letzte Suche wiederholen
CTRL Q T	Zeilenlänge auf Bildschirmbreite reduzieren
CTRL Q Z	Zeilen und Spalten anzeige an/aus

```
; ein erstes Programm

asciout  equ       9

         lea      text,a1     ; startadresse text
         move.b   (a1),d1     ; laenge string
         subq.b   #1,d1       ; der zaehler geht bis $ff
loop     move.b   (a1)+,d0    ; buchstabe in register d0
         move     #asciout,-(A7)  ; unterprogramm buchstaben
         trap     #4          ; ausgeben
         dbra     d1,loop     ; ganzen text ausgeben
         rts
text     str       Hallo, hier bin ich
         sync
```

Bild 1. Die Ausgabe einer Zeichenkette durch ein Unterprogramm

Klammer ereicht, geht der Prozessor wieder zur Einzelschrittverarbeitung über. Zusätzlich kann man jetzt auch in den Maschinensprachemonitor schalten und alle seine Möglichkeiten nutzen. Hierzu gehören die Anzeige und Änderung von Speicher- und Registerinhalten sowie Verschiebebefehle. Mit dem Disassembler kann der Code überprüft werden.
Verläßt man jetzt den Monitor, wird das Programm wieder fortgesetzt – wenn man nicht üblerweise einige Registerinhalte so verändert hat, daß ein Absturz vorprogrammiert ist. Da bei den Einplatinenrechnern statische RAMs verwendet werden, hilft meist ein Druck auf den Resetknopf, das Programm einschließlich der Daten steht noch zur Verfügung.
Alle Bildschirm-Ausgaben kann man auch hier an einen Drucker weiterleiten.

Gleitkommazahlen

Für die Verarbeitung von reellen Zahlen in einem Assemblerprogramm steht eine Bibliothek fertiger Routinen zur Verfügung. Sie werden – mit dem entsprechenden Argument in einem Register – über einen Trap aufgerufen. Die Zahlen werden intern mit 64 Bit dargestellt, was einen Rechenumfang von 10^{-998} bis 10^{+998} ergibt. Ausgegeben werden 13 Stellen in Dezimalschreibweise. *Tabelle 4* zeigt die vorhandenen Routinen und *Bild 3* ein einfaches Beispiel zur Berechnung einer Kreisfläche.

```
strout   equ      $0d
         lea      Text,a1
         move     #strout,-(A7)
         trap     #4
         rts
text     str       So geht es auch
         sync
```

Bild 2. Mit dem Trap-Aufruf wird das Ausgabe-Programm deutlich kürzer

Da die Assemblerprogrammierung nicht jedermanns Sache ist, enthält das Kat-68-System auch noch einen Hochsprachencompiler. Weil ein Einplatinenrechner auch zu Steuerungsaufgaben eingesetzt werden soll, scheidet ein (langsamer) Interpreter aus. Pascal ist eine Computersprache, mit der viele Probleme elegant gelöst werden können. Bei der Entwicklung war eine einfache Handhabung für den Programmierer gefordert. Außerdem sollten zusätzlich die besonderen Eigenschaften eines Einplatinencomputers unterstützt werden.

Tabelle 4: Realzahlen-Verarbeitung

Umwandlung String zu Real
Umwandlung Real zu String:
Exponentialformat
Festkommaformat
Ingenieur-Format
Gleitkommaformat
Umwandlung Real zu Longinteger
Umwandlung Longinteger zu Real
Vergleichsoperationen:
=, <>, <=, <, >=, >
Grundrechenarten:
Vorzeichen kippen
Multiplizieren
Dividieren
Addieren
Subtrahieren
Quadrieren
Kehrwert bilden
Transzendente Funktionen:
Sinus
Cosinus
Tangens
Arcustangens
Arcussinus
Arcuscosinus
Wurzel
Zweierpotenz
Exponential
Zehnerpotenz
Potenz
Zweierlogarithmus
Natürlicher Logarithmus
Zehnerlogarithmus
Sonstiges:
π erzeugen
Fakultaet
Gauss-Klammer
Vorkommastellen beseitigen

Pascal an Bord

Der Pascalcompiler entspricht – abgesehen von Syntaxänderungen in der Datei-Verwaltung – voll dem von Jensen und Wirth gesetzten Standard. Darüber hinaus gibt es Funktionen und Prozeduren, die auch von anderen Compilern bekannt sind, wie eine komfortable Stringverarbeitung.
Steht genügend Speicher zur Verfügung, um den erzeugten Code zusätzlich zum Quelltext im Speicher abzulegen, erhält man sehr schnelle „turn around"-Zeiten. Der Compiler übersetzt dann pro Sekunde etwa 5500 Zeichen. Er benötigt dazu nur einen Durchlauf. Da sich die Laufzeit-Bibliothek ebenfalls im EPROM befindet, ist auch der erzeugte Code recht kompakt. Einen Vergleichswert zeigt *Tabelle 6*.
Tritt bei der Übersetzung oder Ausführung ein Fehler auf, wird sofort in den Editor verzweigt: In der obersten Zeile erscheint eine Fehlermeldung und der Cursor zeigt auf die Fehlerstelle im Text.

Sollte der Speicherplatz nicht ausreichen, kann der Code auch als Disketten-Datei erzeugt werden. Er wird dadurch nicht länger, die Laufzeit-Routinen bleiben im EPROM.
In Pascal haben Integerzahlen grundsätzlich eine Länge von 32 Bit, das entspricht dem Bereich 2147483648...+2147483647. Die Daten werden direkt adressiert, was die Ausführungszeit zwar etwas verlängert, dafür aber den Datenbereich nicht auf 32 KByte beschränkt, wie dies bei einigen Compilern für die CPU 68000 der Fall ist. Beim EMUF nützt das allerdings wenig, wenn der RAM-Ausbau gerade 32 KByte beträgt...

Tabelle 5: Compiler-Benchmark

(Ergebnisse gelten für 10 MHz 68000 Einplatinencomputer mit Kat-68 System)

Sieb des Eratosthenes (Nach BYTE)	
Länge Text	502 Bytes
Compilezeit	0,08 Sekunden
Länge Code	504 Bytes
Ausführung	7,6 Sek. (10 Durchläufe)

Debugging in Pascal

Auch bei einer Hochsprache sind Hilfsmittel zur Fehlersuche bei der Programmentwicklung sehr nützlich. Leider bieten die meisten Compiler hier keine große Hilfe. Beim Kat-68-Pascal gibt es hier zwei Besonderheiten: Mit den Prozeduren TRACEON und TRACEOFF lassen sich Programmteile klammern. Beim Erreichen von TRACEON geht der Prozessor auf Maschinenebene in den Einzelschrittmodus mit der Anzeige der Registerinhalte über. Diesen Modus kann man unterbrechen und alle Möglichkeiten des Monitors nutzen. Verläßt man den Monitor, wird das Programm normal fortgesetzt.
Setzt man beim Compilieren die Option BREAK und fügt in den Quelltext an interessanten Stellen einen Haltepunkt mit der Prozedur BREAK ein, dann hält das Programm an dieser Stelle an. Nun kann man in den Monitor schalten oder sich die Variablen anzeigen lassen. Die Variablen werden mit Namen, Adresse, Typ und Inhalt angezeigt. Ist der Inhalt nicht vernünftig darzustellen, etwa bei SETs, so wird der Inhalt als Bitmuster sichtbar gemacht. Bei Prozeduren und Funktionen werden Bezeichner, Typ und Adresse im Programmcode gezeigt. Verläßt man den Modus, wird das Programm normal fortgesetzt.

Tabelle 6: Funktionen und Prozeduren des Pascal-Compilers

Abs	Addr	Adin	Allocate
Arccos	Arcsin	Arctan	Assign
Blockread	Blockwrite	Bmem	Break
Call	Chr	Cleartimer	Close
Clrscr	Concat	Copy	Cos
Daout	Delay	Delete	Editor
Eof	Erase	Exitus	Exp
Fak	Filepos	Filesize	Firstcall
Frac	GetRegister	Gotoxy	Halt
Hex	Hi	InitCen	InitLst
InitSerA	InitSerB	Inittimer	Insert
Int	Ioresult	Keypressed	Length
Lmem	Ln	Lo	Log
Mark	Mask	Maxavail	Maxint
Memavail	New	Ord	Pi
Pos	Pot	Pred	Private
Ptr	Public	Read	Readln
Readtimer	Release	Rewrite	Reset
Round	Seek	SetRegister	Sin
Sizeof	Sqr	Sqrt	Start
Stop	Stoptimer	Str	Succ
Swap	Tan	Traceoff	Traceon
Trunc	Upcase	Val	WMem
Write	Writeln	False	True

Assembler-Unterprogramme

Reicht die Geschwindigkeit in Pascal nicht aus, so kann man auf Routinen, die in Maschinensprache geschrieben sind, zurückgreifen. Hier bieten sich drei Möglichkeiten: Die erste Variante ist die Verwendung von externen Prozeduren oder Funktionen. Man gibt nur eine feste Adresse nach der entsprechenden Deklaration an. Zur Vorbelegung von Registern des 68000 (immerhin außer dem Stackpointer 15 Register) gibt es eine entsprechende Prozedur. Nach der Rückkehr von einer mit CALL aufgerufenen Routine lassen sich die Register mit der Funktion GETREGISTER lesen.
Der Compiler hat keinen Linker. Trotzdem lassen sich fertige (Assembler)-Module während des Compilierungsvorgangs in den Programmcode einbauen. Die Programmteile müssen vorher vom Assembler zu einer Disketten-Codedatei übersetzt werden. Dazu wird ein Prozedur- bzw. Funktionskopf ganz normal deklariert, gefolgt vom Schlüsselwort CODEFILE und dem Namen dieser Codedatei. Die Assemblerroutinen müssen in sich relativ adressiert sein, was beim 68000 kein Problem ist. Parameter werden auf dem Stack übergeben.

Ungewöhnlich ist auch die Prozedur EDITOR. Hierzu wird ein Feld festgelegt, welches die Größe des Speichers bestimmt, in welchem der Editor des Systems arbeiten kann. Dem Programm stehen so sämtliche Editorfunktionen zur Verfügung, also normales Editieren, Laden oder Abspeichern von Dateien usw. Verläßt man den Editor, ist man wieder im Programm. Auf das Feld läßt sich nun byteweise zugreifen. Ein Anwen-

```
getstring       equ       $07
strout          equ       $0d
real            equ       $33

                lea       eintxt,a1          ; fragetext
                move      #strout,-(A7)      ; ausgeben
                trap      #4
                lea       buffer,a1          ; string einlesen
                move      #getstring,-(A7)
                trap      #4
                moveq     #0,d7              ; in realwert
                move      #real,-(A7)        ; wandeln
                trap      #4                 ; Real in D0/D1
                moveq     #16,d7
                move      #real,-(A7)        ; D0/D1 quadrieren
                trap      #4                 ; Ergebnis in D0/D1
                move.l    D0,D2              ; Quadrat-Wert nach D2/D3
                move.l    D1,D3
                moveq     #34,d7             ; Pi in D0/D1 erzeugen
                move      #real,-(A7)
                trap      #4
                moveq     #12,d7             ; Pi mit Quadrat multipl.
                move      #real,-(A7)
                trap      #4
                lea       Buffer,A1          ; jetzt Real D0/D1 in String
                moveq     #20,D2             ; laenge string
                moveq     #13,d3             ; nachkommastellen
                moveq     #1,D7
                move      #real,-(A7)        ; in string wandeln
                trap      #4
                move      #strout,-(A7)
                trap      #4                 ; ergebnis ausgeben
                rts

eintxt          str       Eingabe:
                sync
buffer          ds.l      20
```

Bild 3. Berechnung einer Kreisfläche: Der Radius wird eingegeben, anschließend das Ergebnis 2r * π berechnet

```
program parallel_demo;

var     time,a,b:integer;

parallel zeit;
begin
 time:=time+1
end;

parallel prozess;
begin
 if private(usr1)
 then begin
         public(usr1);
         a:=sqr(b)
      end
 else a:=111
end;

parallel zaehler;
begin
 b:=b+1;
 if b=400 then start(prozess,1);
 if ( (b > 1000) and ( b <1600))
   then begin                    (* USR1 fur alle anderen sperren *)
          if private(usr1) then ;
        end
   else public(usr1);
 if b = 2400 then stop(prozess)
end;

begin
 clrscr;
 writeln('Die Bearbeitung von 3 parallelen Prozessen');
 gotoxy(8,8);
 write('Prozess      Zaehler      Sekunden');
 a:=0; b:=0; time:=0;
 inittimer(10);
 start(zaehler,1);
 start(zeit,100);

 repeat
  gotoxy(1,10);
  writeln(a:12,b:12,time:12);
 until time >30;
end.
```

Bild 4. Ein Beispiel für Parallelverarbeitung

dungsbeispiel ist die Erstellung von Serienbriefen.
Alle Peripheribausteine lassen sich von der Hochsprachenebene initialisieren und steuern.

Parallele Prozesse

Bei Steuerungsaufgaben ist die Verarbeitung von (quasi) parallelen Prozessen oft notwendig. Kat-68-Pascal bietet hierfür ein neues Konzept: Zeitlich begrenzte Prozesse werden über eine Interrupt-Steuerung ausgeführt. Der Anwender initialisiert einen Timer, der dann in vorgewählten Abständen einen Interrupt auslöst. Ein Prozeß, der in Form einer Prozedur ohne Parameter geschrieben ist, kann nun gestartet werden. Dabei wird festgelegt, nach welcher Anzahl von Interrupts jeweils ein kompletter Durchlauf dieser Parallelprozedur erfolgen soll. Die Ausführung des Hauptprogramms wird so lange angehalten. Um Konflikte mit zeitkritischen Ausgabeleitungen zu verhindern, kann der Anwender den Zugriff für einen Prozeß privilegieren. Dies geschieht mit der Prozedur Public und der Funktion Private.
Ein Beispiel für die Parallelverarbeitung zeigt *Bild 4*: Im Hauptprogramm wird mit Inittimer alle 10 ms ein Interrupt ausgelöst. Dann werden die Parallelprozeduren Zähler und Zeit gestartet. In der Prozedur Zeit wird ein Zähler sekundenweise hochgezählt. In dem anderen Prozess wird ebenfalls gezählt. Wird er erstmalig aufgerufen, so weist first-call dem Zähler ein Startwert zu. Abhängig von diesem Zählerstand wird nun ein weiterer Prozeß gestartet, unterbunden und angehalten. Das Hauptprogramm zeigt in einer großen Schleife den Inhalt verschiedener Variablen.

Potenzen in Pascal

Da die Pascal-Syntax keinen Operator für Potenzen vorsieht, muß man zu ihrer Berechnung erst eine entsprechende Funktion deklarieren. Oft wird folgende Funktion verwendet [1]:

FUNCTION potenz (x,y: real): real;
BEGIN potenz:=exp(ln(x)*y) END;

Allerdings können damit nur Potenzen mit positiver Basis berechnet werden, was z. B. die Anwendbarkeit des Funktionsinterpreters aus [2] stark einschränkt. Um beliebige reelle Potenzen x^y zu ermitteln, sind folgende Fälle zu unterscheiden:

– Die Basis ist Null.
Für positive Exponenten ergibt sich $0^y = 0$, für $y=0$ gilt $0^0 = 1$. Für negative y ist 0^y nicht definiert.

– Die Basis ist positiv.
Der Wert der Potenz ergibt sich aus $x^y = \exp(\ln(x)*y)$.

– Die Basis ist negativ.
In diesem Fall sind nur ganzzahlige Exponenten zugelassen, weil innerhalb der reellen Zahlen keine Wurzeln aus negativen Zahlen existieren. Ist y geradzahlig, so gilt $x^y = (-x)^y$, ansonsten $x^y = -(-x)^y$.

Das *Bild* zeigt eine entsprechende Pascal-Funktion. Bei nicht definierten Potenzen wird durch den Term 1/0 ein Laufzeitfehler hervorgerufen, was natürlich an die jeweils aufrufende Routine angepaßt werden muß.

Götz Leibrock

Literatur

[1] *Plate, Jürgen:* Potenzierung in Pascal. mc 8/82, Seite 61

[2] *Röhner, Gerhard:* Funktionsinterpreter in TurboPascal. mc 5/87, Seite 78

Diese Potenzfunktion in Pascal berücksichtigt alle Fälle

```
FUNCTION potenz (x,y: real): real;

  { Berechnung beliebiger reeller Potenzen:
    potenz:=(x)^(y); (c) 1987 Götz Leibrock }

  BEGIN
  IF x=0 THEN
    IF y=0 THEN potenz:=1
    ELSE
      IF y>0 THEN potenz:=0
             ELSE potenz:=1/0    { Fehler }
  ELSE
    IF x>0 THEN potenz:=exp(y*ln(x))
    ELSE IF y=trunc(y) THEN
           IF odd(trunc(y)) THEN
             potenz:=-potenz(-x,y)
           ELSE
             potenz:= potenz(-x,y)
         ELSE potenz:=1/0    { Fehler }
  END;
```