Rolf-Dieter Klein, Jürgen Plate

Mikroelektronik im Fernsehen

Teil 7

In dieser Folge wird, wie versprochen, das Pascal-System des 68008-Computers besprochen. Mit der wohl interessantesten Routine des Grundprogramms wird dann auch die Besprechung der 68008-Software abgeschlossen. Im August beschäftigen wir uns dann noch einmal mit dem Prozessor Z80.

Es würde schon rein räumlich den Rahmen eines Zeitschriften-Artikels sprengen, eine Beschreibung von Pascal zu liefern. Außerdem gibt es genügend gute Bücher, die in die Programmierung mit Pascal einführen. Pascal eignet sich aus mehreren Gründen ganz besonders für das Erlernen des Programmierens, wobei die Zweckmäßigkeit und die Anwendbarkeit anderer Programmiersprachen garnicht angezweifelt werden soll.
Bei der Entwicklung von Pascal verfolgte Nicklaus Wirth um 1970 das Ziel, die neuesten Methoden der Software-Entwicklung wie strukturierte Programmierung und schrittweise Verfeinerung anwendbar zu machen. Es entstand eine Sprache, deren Vorzüge relativ schnell zu weitreichender Verbreitung der Sprache nicht nur in der Ausbildung, sondern auch in Technik und Wissenschaft geführt haben:
- leichte Erlernbarkeit,
- Rechnerunabhängigkeit,
- problemnahe, strukturierte und effiziente Programmierung,
- lesbare, selbstdokumentierende Programme,
- Definierung problembezogener Datenstrukturen.

Pascal in PROM

Das Pascal-S für den NDR-Klein-Computer enthält die ganzen wichtigen Elemente der Sprache Pascal, es wurde nur ein wenig „abgemagert". Als Datentypen gibt es Real, Integer, Char und Boolean; an Datenstrukturen kennt Pascal-S das Feld (Array) und den Verbund (Record). Es fehlen Mengen, Pointer und Dateien. Dagegen sind alle Anweisungsstrukturen mit Ausnahme von GOTO enthalten.
Für die Sprache Pascal auf den NDR-Klein-Computer brauchen Sie eine weitere ROA-64-Karte, vier EPROMs 2764 mit dem Pascal-Compiler und vier RAM-Bausteine. Die EPROMs belegen den Speicherbereich ab 10 000 hexadezimal. Das Pascal-System wird über die Bibliotheksfunktion aufgerufen. Das Übersetzungsprotokoll kann analog jenem des Assemblers über die Assembleroptionen gesteuert werden. Beim Franzis-Software-Service ist ein Heft erhältlich, das eine kurze Sprachbeschreibung, das Original-Listing des Pascal-S von N. Wirth, Programmbeispiele und das Assemblerlisting des 68008-Pascal-S enthält.
Das erste Programmbeispiel in *Bild 1* zeigt zwei Erweiterungen des Pascal. Einmal existiert eine genormte Schnittstelle zu Maschinenprogrammen, so daß Sie nicht nur auf alle Routinen des Grundprogramms zurückgreifen können, sondern auch Assemblerprogramme, etwa für die Grafik, einbinden können. Diese Schnittstelle wird über die Prozedur WRITELN realisiert. Durch ein spezielles ASCII-Zeichen mit dem Wert 1 wird dem Pascal-System mitgeteilt, daß keine Ausgabe gewünscht wird, sondern ein Systemaufruf erfolgen soll. Davon gibt es drei:
- E erzeugt den Aufruf eines Maschinenprogramms, das mit RTS beendet werden muß.
- P schreibt ein Byte direkt in den Speicher (P <Adresse> <Datenbyte>).
- G holt ein Speicherbyte (G <Adresse>, dann wird das Byte beim nächsten READ an die angegebene Variable übergeben).

Nach dem E des Maschinenaufrufs kommt der Name des Unterprogramms, gefolgt von den Werten der eventuell benötigten D-Register von D0 bis D7. Es müssen nur die benötigten Register, jeweils durch Leerzeichen getrennt, angegeben werden. Im Beispiel von Bild 1 wird zwischen Bildschirm- und Druckerausgabe umgeschaltet.

```
 0  program test0(input,output);
 0
 0  var a:real;
 0  i:integer;
 0  ch:char;
 0  z:array[1..5,1..5] of real;
 0
 0  procedure ausgabe;
 0   begin
 0    writeln('Ausgabe')
 2   end;
 3
 3  begin
 4  ausgabe (* auf Bildschirm *);
 6  writeln(chr(1),'E @LST');
12  ausgabe (* auf Drucker *);
14  writeln(chr(1),'E @CRT');
20  ausgabe (* wieder auf BS *)
20  end.
```

Bild 1. Ein einfaches Pascal-Programm

```
identifiers              link   obj   typ   ref   nrm   lev   adr
      29 A                  0     2     3     0   255     1     5
      30 I                 29     2     2     0   255     1     6
      31 CH                30     2     5     0   255     1     7
      32 Z                 31     2     6     1   255     1     8
      33 AUSGABE           32     4     1     3   255     1     0

blocks     last lpar psze vsze
     1       28    1    0    0
     2       33   28    5   33
     3        0   33    5    5

arrays     xtyp etyp eref  low high elsz sizw
     1        2    6    2    1    5    5   25
     2        2    3    0    1    5    1    5

code :

  0 24     7, 28     0, 63      , 32      , 18     33,
  5 19     4, 24     1,  8     5, 29     4, 24     6,
 10 28     7, 63      , 18    33, 19     4, 24     1,
 15  8     5, 29     4, 24     6, 28    13, 63      ,
 20 18    33, 19     4, 31       ,
```

Nennt man das Programm TEST0, werden auch die internen Daten- und Codetabellen zur Information ausgegeben.

Der Compiler übersetzt das Programm nicht direkt in die Maschinensprache des 68008 (wie es der Assembler macht), sondern in einen Zwischencode, der dann von einem weiteren Teil des Pascal-S, dem Interpreter, ausgeführt wird. Diese Methode erlaubt einen kurzen Compiler und sehr kompakten Code. Das Programm in *Bild 2* macht regen Gebrauch von der Maschinenschnittstelle. Es ist der Pascal-Teil des Progamms „Türme von Hanoi", dessen Assemblerteil im letzten Heft abgedruckt war. Das Ausprobieren der Funktionen G und P überlassen wir Ihrem Spieltrieb. Versuchen Sie auch einmal ein Pascal-Programm zu schreiben, das abwechselnd den Editor und dann das Pascal-S aufruft. Sie können übrigens auch von Assemblerprogrammen aus auf die Routinen des Pascal-Systems zugreifen, zum Beispiel auf die 14stellige BCD-Gleitkommaarithmetik.

Bild 2. „Türme von Hanoi", Pascal-Teil

```
  0  program hanoi(input,output);
  0
  0   const
  0    x1 = 200;   (* Lage der Tuerme in x-Richtung *)
  0
  0    y0 = 40;    (* Hoehe der Tuerme *)
  0    yoben = 100; (* Hoehe max der Tuerme *)
  0
  0
  0   var total:integer;
  0   x,y: integer;
  0
  0   h: array[1..3] of integer;  (* hoehe *)
  0
  0
  0  procedure setturm; (* Startturm *)
  0  var i:integer;
  0
  0  begin
  0   for i:=1 to total do begin
  4    writeln(chr(1),'E assplatte ',(1+total-i)*2,' ',x1,' ',y0+i*14);
 30    writeln(chr(1),'E assfest');
 36   end;
 37   writeln(chr(27),'=',chr(32+23),chr(32+31),' 1         2         3');
 55   h[1]:=total;
 60   h[2]:=0;
 65   h[3]:=0;
 70  end;
 70
 70  procedure platte(hoehe,von,nach:integer);
 71
 71  var x,y,i:integer;
 71   dx:integer;
 71   xziel:integer;
 71
 71  begin
 71   for y:=h[von]*14+y0 to yoben do
 82    writeln(chr(1),'E assplatte ',hoehe*3,' ',x1+50*(von-1),' ',y);

107   h[von]:=h[von]-1;
117   x:=x1+50*(von-1);
126   xziel:=x1+50*(nach-1);
135   if xziel < x then dx:=-1 else dx:=1;
147   repeat
147    writeln(chr(1),'E assplatte ',hoehe*3,' ',x,' ',yoben);
165    x:=x+dx;
170   until x=xziel;
174   h[nach]:=h[nach]+1;
184   for y:=yoben downto h[nach]*14+y0 do
195    writeln(chr(1),'E assplatte ',hoehe*3,' ',x1+50*(nach-1),' ',y);
220   writeln(chr(1),'E assfest');
226  end;
226
226
226
226
226
226  procedure schiebe(hoehe,von,nach,mit:integer);
227  begin
227   if hoehe > 0
229    then begin
231     schiebe(hoehe-1,von,mit,nach);
240     writeln(chr(27),'= ');
246     writeln('von ',von,' nach ',nach);
255     platte(hoehe,von,nach);
261     schiebe(hoehe-1,mit,nach,von);
270    end;
270   end;
270
270   begin
271    repeat
271     writeln;
272     write('Turmhoehe ');
274     read(total);
276     writeln(chr(1),'E assinit');
282     setturm;
284     writeln;
285     schiebe(total,1,3,2);
291    until false
291  end.
```

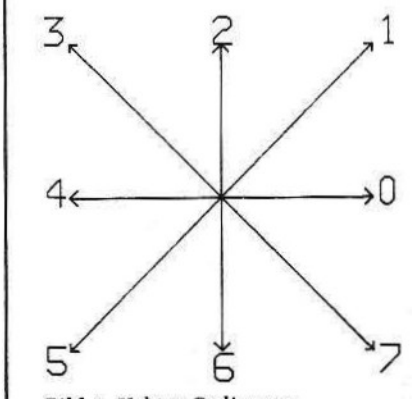

Bild 3. Vektor-Codierung

Der Figur-Befehl im 68008-Grundprogramm

Videospiele arbeiten vorwiegend mit beweglichen Figuren. Aber auch Quasi-Analoganzeigen, die einen beweglichen Zeiger haben, können aus solchen Figur-Elementen erzeugt werden.
Im Grundprogramm des 68008 im NDR-Klein-Computer ist eine Routine vorbereitet, die eine einfache Erzeugung solcher Figuren erlaubt. Der Grafik-Prozessor EF9366, der ja im Computer das zentrale Element für die Bildschirmausgabe darstellt, enthält einen eingebauten Kurzvektor-Befehl. Damit ist es möglich, sehr schnell und mit wenig Information aus Vektoren bestehende Figuren zusammenzusetzen. Man könnte daher Figuren auch direkt an den Grafik-Prozessor in Form von Kurzvektoren ausgeben, was mit dem eingebauten Befehl @CMD sehr einfach geht. Aber die Kurzvek ren müssen erst codiert werden, und zudem hat die Sache noch einen Haken: Wenn eine Figur auf dem Bildschirm

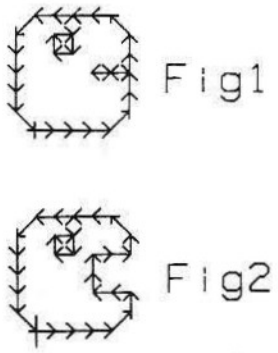

Bild 4. Packman in Vektoren zerlegt

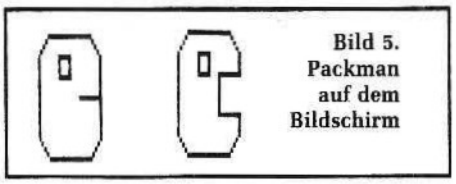

Bild 5. Packman auf dem Bildschirm

bewegt werden soll, so muß zuerst die alte Figur gelöscht werden und dann die neue auf einer neuen Koordinate ausgegeben werden. Um dem Anwender diese Mühe abzunehmen, wurde das Unterprogramm @FIGUR zur Verfügung gestellt. Man braucht ihm nur die Adresse der Figur (die in Vektorform vorliegt), die Koordinaten (x und y), sowie einen Vergrößerungsfaktor mitzuteilen. Dann kann man einfach durch Verändern von x,y oder der Größe die Figur auf dem Bildschirm bewegen. Das Löschen der alten Figur wird dabei automatisch durchgeführt.

Bevor man eine Figur auf den Bildschirm bringen kann, muß man sie in Vektoren zerlegen. Dazu zeigt *Bild 3* ein Fadenkreuz. Es ist von 0 bis 7 beschriftet, was die einzelnen Vektor-Richtungen darstellt. Es gibt noch ein paar Zusatzcodes zur Steuerung:

8 bedeutet kein Schreibvorgang bei nachfolgenden Vektoren,

9 bedeutet Schreibvorgang bei nachfolgenden Vektoren und

10 bedeutet Ende der definierten Figur.

Bild 4 zeigt zwei Figuren, die schon in Vektoren zerlegt wurden und *Bild 5* das Ergebnis auf dem Bildschirm, ein Packman-ähnliches Tierchen. *Bild 6* enthält das komplette Assemblerprogramm. Bei der Marke FIG1 wird die erste Figur definiert und bei FIG2 die zweite. Denn der Packman soll nicht nur über die Bildfläche bewegt werden, sondern sich auch selbst dabei bewegen. Das Grundprogramm ist in der Lage, auch dies durchzuführen.

Das Hauptprogramm beginnt bei START. Die Figur soll sich entlang der X-Achse von x = 0 bis x = 512 bewegen und dabei auf der Höhe y = 128, also in Bildmitte bleiben. Die Adresse der Figur wird mit LEA FIG1,A0 in das Register A0 geladen, dann folgt eine Abfrage der X-Koordinate. Immer wenn Bit 3 (AND #$8,D0) auf 1 liegt, soll FIG2 ausgewählt werden, um die Eigenbewegung durchzuführen. Bei SCHL1 wird in das Register D0 der Wert 4 geladen, dies bedeutet eine Vergrößerung um den Faktor 4, also ist jedes Vektorelement vier Punkte groß. In D1 steht die X-Koordinate, in D2 die Y-Koordinate. Dann wird durch das Unterprogramm WARTE verhindert, das die Figur sich zu schnell über das Bildfeld bewegt und außerdem ein flimmerfreies Bild erzeugt wird, da die Figur immer während der Austastlücke eingeschrieben wird. Man kann auch ein flimmerfreies Bild erreichen, wenn man zwei Bildseiten zur Ausgabe verwendet, jedoch ist der Programmaufwand dann etwas höher.

Noch ein paar Bemerkungen: Ist der Wert in Register D0 = 0, dann wird nur die alte Figur gelöscht, aber keine neue mehr geschrieben. Mit einem Unterprogramm @SETFIG kann man verhindern, das die alte Figur gelöscht wird, wenn man das einmal braucht.

Mit dem Figurenbefehl lassen sich auch einfach Fadenkreuze erzeugen, die man über den Bildschirm bewegen kann. Dann sollte man dazu eine zweite Bildebene verwenden und mit dem Autoflip-Mode beide Bildebenen quasisimultan darstellen.

```
009C00
009C00
009C00                          ********************************
009C00                          * BEWEGTE GRAPHIK MIT DEM      *
009C00                          * FIGUR-BEFEHL  840421 RDK     *
009C00                          ********************************
009C00
00A000                          ORG $A000
00A000
00A000                          FIG1:    * PACKMAN 1
00A000  00 00 00 00 01           DC.B 0,0,0,0,1,2,2,4,4,0,0,2,2
00A005  02 02 04 04 00
00A00A  00 02 02
00A00D  03 04 04 08 06           DC.B 3,4,4,8,6,9,6,4,2,0,8,2,9
00A012  09 06 04 02 00
00A017  08 02 09
00A01A  04 04 05 06 06           DC.B 4,4,5,6,6,6,6,7,10
00A01F  06 06 07 0A
00A023
00A023                          FIG2:    * PACKMAN 2
00A023  00 00 00 00 01           DC.B 0,0,0,0,1,2,4,4,2,2,0,0,2
00A028  02 04 04 02 02
00A02D  00 00 02
00A030  03 04 04 08 06           DC.B 3,4,4,8,6,9,6,4,2,0,8,2,9
00A035  09 06 04 02 00
00A03A  08 02 09
00A03D  04 04 05 06 06           DC.B 4,4,5,6,6,6,6,7,10
00A042  06 06 07 0A
00A046
00A046
00A046                          START:     * BEWEGT PACKMAN
00A046                                     * UEBER DAS BILDFELD
00A046  343C 0080                MOVE #128,D2 * Y-ACHSE
00A04A  4241                     CLR D1    * X-ACHSE
00A04C                          SCHLEIFE:
00A04C  41F9 0000A000            LEA FIG1,A0
00A052  3001                     MOVE D1,D0
00A054  0240 0008                AND #$8,D0
00A058  6706                     BEQ.S SCHL1
00A05A  41F9 0000A023            LEA FIG2,A0
00A060                          SCHL1:
00A060  303C 0004                MOVE.W #$4,D0 * VERGROESSERUNG
00A064  4EB9 00003A0C            JSR §FIGUR
00A06A  4EB9 0000A07C            JSR WARTE
00A070  0641 0002                ADD #2,D1
00A074  0C41 0200                CMP #512,D1
00A078  66D2                     BNE.S SCHLEIFE * WIEDERHOLE BIS X=511
00A07A  4E75                     RTS
00A07C
00A07C
00A07C                          WARTE:           * WARTESCHLEIFE
00A07C  383C 0001                MOVE #2-1,D4   * ANZAHL * 20MS
00A080                          WARTE1:
00A080  4EB9 00000852            JSR §SYNC
00A086  67F8                     BEQ.S WARTE1
00A088  51CC FFF6                DBRA D4,WARTE1
00A08C  4E75                     RTS
00A08E
00A08E                          END

0000    Fehler entdeckt
008A88  Ende-Symboltabelle
009456  Ende-Debug-Tabelle
```

Bild 6. Listing des einfachen Demoprogramms (§ = @)

Wie geht es weiter?

Im August werden wir über zwei neue Sprachen für den Z80 berichten und noch ein wenig auf die letzten Folgen der Fernsehserie eingehen.