VIERTE DIMENSION

2/1993

9. Jahrgang 1993 2. Quartal DM 10,--

Grafik im F-PC

OS/2-Forth

Variablen und Konstanten elegant deklarieren

DisCo-Beispiel: Die Schnecke

Interview mit Tom Zimmer

3. Folge

Die Uhr

Forth++-Anwendung:

INFOSYS 106

68HC11 Mikro Controller

FORTH MAGAZIN

Organ der Forth-Gesellschaft e.V.

IMPRESSUM

Name der Zeitschrift
VIERTE DIMENSION
FORTH MAGAZIN
Organ der Forth-Gesellschaft e.V.

Herausgeber
Forth-Gesellschaft e.V.
Postfach 1110
85701 Unterschleißheim
Tel.: 089-3173784 oder
Forth-Mailbox Tel. 089-8714548 8N1

Redaktionsleitung
Rolf Kretzschmar (rk), (verantwortlich)
Rote Gasse 7, 52499 Baesweiler
(Redaktionsadresse)
Tel/Fax: 02401-88891

Redaktion
Arndt Klingelnberg (akg), Alsdorf
Tel.: 02404-61648 Fax: 02404-63039
Klaus-Peter Schleisiek (kps), Aachen
Tel/Fax: 0241-873462

Layout, Satz, Herstellung
ORGA Sport, Rilkestr. 8, 52477 Alsdorf
Tel/Fax: 02404-61425

Grafik, Illustration, Layout
Rolf Kretzschmar (rolf)

Anzeigenverwaltung
Arndt Klingelnberg
Straßburger Str. 12
52477 Alsdorf
Tel.: 02404-61648 Fax: 02404-63039

Redaktionsschluß
Feb., Mai, Aug., Nov.

Erscheinungsweise
vierteljährlich

Auflage
1000

Preis
Einzelheft DM 10,-, Abonnementpreis DM 50,-, bei Auslandsadresse DM 55,- inklusive Versandkosten

Direktorial

Die FORTH-Gesellschaft e.V. Stand und Tendenzen 1993

von Ulrich Hoffmann, Jörg Staben, Thomas Beierlein

Ein Jahr ist seit dem Treffen in Rostock vergangen und wieder fand eine Jahreshauptversammlung der FG statt - diesmal in Nürnberg.

Ein guter Zeitpunkt, Bilanz zu ziehen und die Perspektiven auszuleuchten. Im vergangenen Jahr ist es uns gelungen, den starken Mitgliederschwund des Jahres 91/92 aufzuhalten und die Mitgliederzahlen zu stabilisieren.

Neue Aktivitäten wie der MikroController-Verleih von *Rafael Deliano* sind entstanden oder haben sich noch breiter durchgesetzt wie die Mailbox, die von Jens Wilke betreut wird.

Weiterhin hat das neue Redaktionsteam der VIERTEN DIMENSION um *Rolf Kretzschmar* aus unserer Zeitschrift ein sehr ansprechendes Blatt mit einem interessanten Inhalt gemacht. Dies war auch die einhellige Meinung der Teilnehmer in Nürnberg. Das Direktorium möchte an dieser Stelle nicht nur den Redakteuren, sondern ausdrücklich allen Aktiven danken.

Im Laufe des letzten Jahres stellte sich heraus, daß unsere Satzung den vereinsrechtlichen Bestimmungen nicht vollständig gerecht wurde. Aus diesem Grund wurden auf der diesjährigen Versammlung Satzungsänderungen vorgeschlagen und ausdiskutiert, die neben Schönheitskorrekturen noch einen wesentlichen Punkt beinhalteten: Das Stimmrecht für juristische FG-Mitglieder. Damit soll den Firmen, die in unserer FG mitarbeiten und die Arbeit der FG mit prägen, auch ein angemessenes Mitbestimmungrecht in der FG gegeben werden.

Da die Tagung leider zu diesem Zeitpunkt nicht mehr beschlußfähig war, müssen halt nun die Mitglieder über diesen Änderungsvorschlag in Schriftform abzustimmen. (Ein Formular ist diesem Heft beigegeben. rk)

Noch zu ergänzen bleibt der Beschluß der Mitgliederversammlung, die Mail-Box der FG noch weiter auszubauen und dadurch die Möglichkeiten der direkten Kommunikation zwischen den Mitglieder der FG zu verbessern.

Die FG steht allerdings auch vor einem sehr ernsthaften Problem: Es herrscht akuter Geldmangel. Hauptgrund dafür sind die vergleichsweise geringe Mitgliederzahl und vor allem die Preisentwicklung der letzten Jahre. Z.B. erhöhen sich allein durch die vor wenigen Wochen eingeführten neuen Portopreise die Ausgaben für den Versand der VD im Jahr um ca. 1800 DM(!). Auch die gute Qualität und das Layout der VD haben ihren Preis, den zu zahlen allerdings ein Beschluß der Jahreshauptversammlung 1992 in Rostsock war.

In diesem Zusammenhang gilt es, die Einnahmen und Ausgaben der FG in Einklang zu bringen. Da die Mitgliederversammlung eine Erhöhung der vollen Mitgliedsbeiträge ablehnte - die Mitgliedsbeiträge für Studenten werden allerdings auf das Niveau der Herstellungskosten der *Vierten Dimension* angehoben - bleibt nur der Weg die anfallenden Kosten zu reduzieren.

Weil unsere Zeitschrift VD den größten Posten bei den Ausgaben einnimmt, werden wir vor allem dort ansetzen müssen (Papierqualität/Druckkosten).

In Nürnberg ist sehr schön deutlich geworden, daß die Forth Gesellschaft im Laufe der letzten 10 Jahre zu einem Forum gewachsen ist, in dem die an Forth interessierten ihre Ideen, Anregungen und auch Fragen öffentlich machen können.

Wir sind davon überzeugt, daß uns dieses Jahr 1993 eine Reihe interessanter Entwicklungen bringen wird und der Kreis der FORTH-Programmierer nach wie vor aktiv bleibt.

Mailbox / DFÜ-News

von Jens Wilke
Streitberger Straße 73a, 81249 München, Tel.: 089-71 43 52

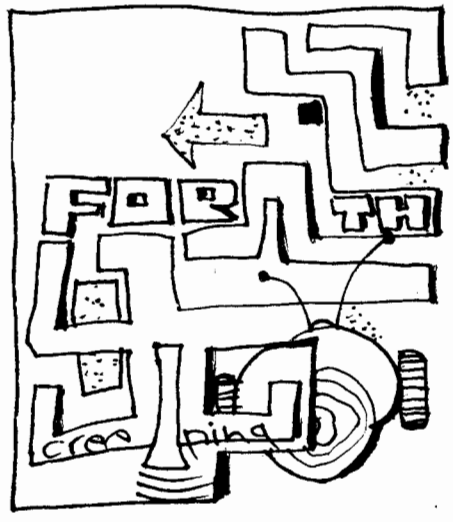
In dieser (hoffentlich) regelmäßig erscheinenden Rubrik wird Jens Wilke zukünftig über die aktuellen Neuigkeiten im Bereich DFÜ berichten und selbstverständlich auch über da, was im Mailbox-Forum zur Zeit ausgekocht wird.

Zunächst möchte ich diejenigen ermuntern, die bis jetzt noch nicht den Zugang zur DFÜ gefunden haben. Modems mit 2400 Baud sind schon unter 200 DM zu haben. Also sollte es keine Kostenfrage mehr sein, einzusteigen. Wer Probleme hat oder Tips haben möchte, kann gerne bei mir anrufen. Über den Modemmarkt gibt es auch ein paar interessante Dinge zu berichten. Sogenannte Highspeedmodems sind jetzt auch für den Hobbyanwender zu einem vernünftigen Preis erhältlich. Die höchste gängige Geschwindigkeit ist 14.400 Baud; mit zusätzlicher Datenkompression werden sogar noch höhere Übertragungsraten erzielt. Ein hübsches Feature solcher Modems ist der zusätzliche FAX-Modus. Es gibt sogar schon ein Modem, mit dem man normale Sprache (auf der Telefonleitung) samplen kann und auch wieder ausgeben. Damit kann man sich dann seinen Anrufbeantworter programmieren oder eine sog. Voice-Mailbox (Anrufbeantworter für viele Personen) aufbauen. So eine Voice-Mailbox wäre bestimmt ein interessanter und nützlicher Vereins-Service...; aber das war nur laut gedacht. Es tut sich also einiges, auch in der Mailbox (die läuft aber leider noch mit 2400 Baud). Besonders hervorzuheben ist hier das Polltool mit dem man sehr gut Telefonkosten sparen kann, da der Mailboxrechner automatisch die neusten Briefe und Nachrichten beim Anruf sendet. Somit steigt die Telefonrechnung nicht mehr als 10 bis 20 DM im Monat, wenn man regelmäßig (2-3 mal die Woche) anruft. Soviel würde die Mitgliedschaft bei einer anderen Mailbox, die Usenet (internationale EMails/News) anbietet, wie die Forthev-Mailbox, auch kosten. Bei uns ist es aber für Vereinsmitglieder kostenlos! Im Forum wurde auf die Initiative von *Rafael Deliano* über einen Embeded Standard diskutiert. Also ein Standard-Forthsystem, das auf 8-Bit Controller zugeschnitten ist. Nachdem man sich darüber klar geworden ist, ob ein neuer Standard parallel zu ANS Sinn hat, (immernoch leichte Unstimmigkeiten...) werden wohl bald ein paar ernstzunehmende Vorschläge auf dem Tisch liegen. Ich persönlich habe mich mal den Anfängern gewidmet und Anregungen für ein String-Wordset im Forum gesammelt. Für solche Dinge wurde im Filebereich der Box ein Bereich namens PROJECT eingerichtet. Darin sind Sourcen zu aktuellen Diskussionen im Forum zu finden. Als letztes ist die aktuelle Debatte zu erwähnen. Um eine zuschaueranlockende Attraktion am Messestand des Forth-eV bei der Echtzeit zu haben, wurde über Miniroboter nachgedacht die in einem vertikalen (an der Wand hängenden) Labyrinth herumfahren. Hier wird man bestimmt bald noch mehr darüber hören/lesen. c u jaw

❑

Creeping Forth II

rk



Sie erinnern sich: Raphael Deliano wirbt um Mitarbeit in einer Robotik-AG, und um die Sache in Gang zu bringen, hat er bereits viel Vorarbeit geleistet. In einem 24 Seiten umfassenden 2. Heft beschreibt er, was alles zu bedenken ist, wenn man ein Roboter-Fahrzeug (Schildkröte) entwickeln möchte. Daß die Kröte auf vorgegebenem Raster an der senkrechten Wand entlang geführt werden soll, macht die Sache nicht einfach! Doch steigert das durchaus die Attraktivität und auf Messen die Überschaubarkeit für das Publikum. (Mein Tip: Sollten die mechanischen Problem zu groß werden, kann man auf große Spiegel ausweichen und dann das Ding auf dem Tisch laufen lassen.) Sowohl Bastler als auch Forth-Programmierer sind angesprochen, in der AG mitzumachen. Raphael bietet in seinen Heften wertvolle Hinweise und interessantes Grundlagenwissen (z.B.:DC-Motore, Getriebe, Motoransteuerung) aber auch Software-Lösungen (z.B.: Vereinfachte Antriebsberechnung). Wir sind wirklich gespannt, zu erfahren, welche Resonanzen diese vorbildliche Initiative hervorrufen wird. Gerne würden wir nach der nächsten Echtzeit davon berichten, daß auf dem Forth-Stand der Bär , nein, die Schildkröte los war. Interessenten wenden sich an

Raphael Deliano
Steinbergerstr. 37
82110 Germering
Tel.: 089/8418317
MB: 089/8714548 Forth e.V.
jrd@BBS.FORTH-eV.de

Inhalt

Die Uhr (3 von 3)

von Friederich Prinz

Am Beispiel einer im Hintergrund arbeitenden Uhr wird die TSR Programmierung unter ZF-Forth ausführlich behandelt. Es wird gezeigt, wie man TSRs via Interpreter startet.

Fangen wir mit der UHR selbst an:

```
LABEL Uhr NOP NOP
AX PUSH BX PUSH
CX PUSH DX PUSH
DS PUSH ES PUSH
DI PUSH SI PUSH

CS PUSH DS POP
```

Alle beteiligten Register werden auf dem Stack gesichert. Dabei soll DOS seinen eigenen STACK benutzen !

Hier findet eine etwas unkonventionelle, aber sehr schnelle Segmentzuweisung statt. Wie bereits ausgeführt, soll die Uhr über den Vektor von INT 1Ch angesteuert werden. Bei der Übergabe der Adresse aus der Vektorentabelle wird das Register CS gel

```
Anzahl # BX MOV
```

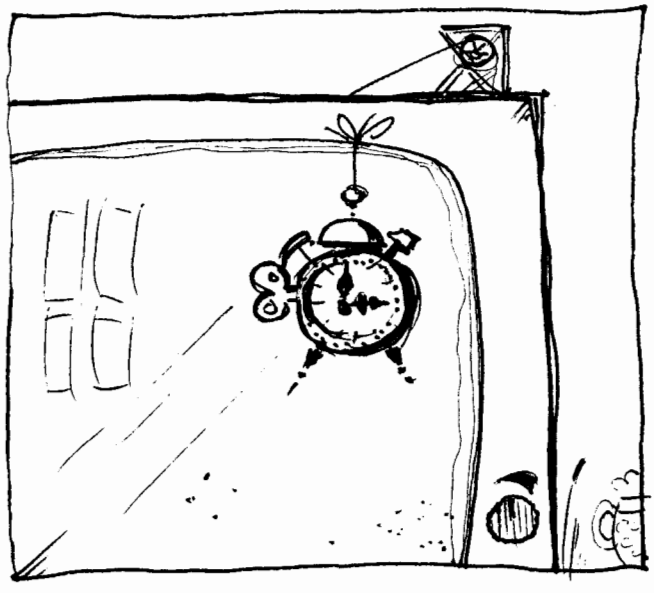
`Anzahl` ist eine der gerade angesprochenen Variablen. Hier werden die eingehenden Signale des Timers gezählt. Wenn eine bestimmte Anzahl erreicht ist, soll die Uhr aktualisiert werden. In der vorliegenden Implementierung ist dieser Wert '6'. Das heißt, daß die Uhr rund 3 mal/Sekunde aktualisiert wird.

```
UhrStart # BX SUB
         2 # BX ADD
```

An dieser Stelle kommen die bereits beschriebenen Besonderheiten eines ONE-Pass Assemblers zum ersten Male zum Tragen.

Die Adresse der 'Variablen' Anzahl kann nicht direkt übergeben werden. Der Assembler würde einfach den Offset der entsprechenden Speicherstelle auf den Anfang des aktuellen Segmentes einsetzen. Das hätte bei der späteren Auslagerung des CODE in ein separates Segment zur Folge, das diese Adresse einfach 'irgendwohin' zeigt. Die Inhalte der so adressierten Speicherzelle wären für die Uhr irrelevant. Ein Überschreiben dieser Inhalte hat in aller Regel fatale Folgen für das ganze System

Einen Ausweg aus diesem Dilemma bietet der hier eingeschlagene Weg. Wie in der Gesamtübersicht am Ende des Aufsatzes zu erkennen ist, markiert ein 'leeres' Label mit dem Namen UhrStart den Beginn des eigentlichen Codes der Uhr. Davon ausgehend, daß dieses Label als erstes in das separate Segment kopiert werden wird (siehe unten), läßt es sich als 'Bezugsmarke' zur Berechnung eines Offsets Verwenden. Jetzt wird einfach der 'Abstand' zu dieser Marke errechnet, der nach dem Umkopieren dem Abstand zum Start des dann aktuellen Segmentes entspricht.



Weil aber das WORT Label selbst noch eine 'Kennung' von 2 Byte für den FORTH-Interpreter den nachfolgenden Codes vorsetzt, wird mit `2 # BX ADD` (die errechnete Adresse steht ja in BX) der 'Zeiger' auf die gewünschten Dateninhalte noch einmal 2 Byte weiter gesetzt.

```
1 # 0 [BX] ADD
    0 [BX] AX MOV
6 # AX CMP
```

Die über den Zeiger in BX erreichbaren Daten, ein 16-Bit Feld, werden nach AX kopiert und dort mit dem Wert '6' verglichen. Die '6' entspricht sechs Signalen des Timers, und damit cirka 1/3 Sekunde. Diese Angabe kann natürlich beliebig verändert werden...

```
Aussprung JNE
```

...wenn das COMPARE mit '6' nicht Gleichheit erbracht hat, soll in die Subroutine 'Aussprung' verzweigt werden. Die Arbeit wird beendet - die Uhr wird nicht aktualisiert.

```
0 # 0 [BX] MOV
```

Die etwas ungewohnte Adressierung des Inhaltes von BX, mit dem Offset '0' entspricht der 'Primitivität' des ZF-Assemblers. Zum tiefergehenden Verständnis dieser Schreibweise, die maximale Flexibilität bei der Lösung aller denkbaren Adressprobleme garantiert, sei auf den ZF-Assemblerkurs der Moerser Forthgruppe verwiesen.

Hier wird aber einfach der Inhalt der Variable 'Anzahl' wieder auf NULL gesetzt, damit die Zählerei von vorne beginnen kann...

```
Zeit>$ #) CALL
```

Die Subroutine '`Zeit>$`' erledigt gleich mehrere Arbeiten hintereinander weg. Zunächst wird diese Routine, die hier aufgerufen wird, sich mit dem Timer selbst beschäftigen , bzw. mit dessen Signal. Danach arbeitet `Zeit>$` die aufgelaufenen 'Ticks' in eine gültige Uhrzeit um, konvertiert diese Zeit in BCD und ASCII und überträgt das alles in eine tatsächlichen Abstand der aufzurufenden Routine zur aktuellen Position im Code entsprechen. Da dieser Abstand nicht verändert werden soll, braucht `CALL` keine 'Sonderrechnung' wie im Fall der 'Variablen'.

```
.Zeit #) CALL
```

Am Ende der Arbeiten soll die Zeichenkette natürlich noch auf dem Bildschirm ausgegeben werden. Das erledigt `.ZEIT`.

```
Aussprung JMP
```

Zu guter Letzt muß die Uhr natürlich auch ihre Arbeit beenden. Das erledigt die Routine `Aussprung`, die hier mit einem unkonditionierten Sprung adressiert wird.

Der Code der UHR ist mit Blick darauf entstanden, grundsätzlich aufzuzeigen, wie 'man so etwas machen' kann. Das heißt, daß sowohl Optimierungen in Hinsicht auf das Laufzeitverhalten, als auch auf den Speicherbedarf der Uhr möglich sind. Solche Optimierungen sollten von den Lesern der VD selbst vorgenommen werden. Das will der Autor nicht leisten. Statt dessen soll die Uhr als Beispiel dafür dienen, wie man unter anderem auch 'Bildschirmschoner' als residente Hintergrundtasks implementiert.

Die Subroutine `Zeit>$` - Zeit-zur-Zeichenkette - erledigt, wie gerade angesprochen, eine ganze Reihe von Einzelarbeiten. FORTHig korrekt würde man diese Einzelarbeiten in kleine Subroutinen zerlegen. Auf Rechnern mit relativ hohem CPU-Takt dürfte das auch kaum Probleme bereiten. Trotzdem sollte man daran denken, daß jeder'`CALL`' zusätzliche Rechenzeit beansprucht und jedes '`LABEL`' Speicherplatz kostet. Aus diesen Gründen hätte ich, wenn das nicht die Verständlichkeit des CODEs beeinträchtigt hätte, die gesamte Uhr 'am liebsten am Stück' geschrieben...

```
Label Zeit>$
0 # AH MOV
1A  INT
```

Mit der Funktionsnummer '0' wird der Interrupt 1Ah aufgerufen. Dieser INT liefert zwei Zähler auf die aktuelle Uhrzeit. Nach der Rückkehr aus dem Service findet der Programmierer im Register CX die Anzahl der aktuellen Stunden. Beim Einschalten des PC wird dieser Zähler einmal auf die korrekte Stundenzahl gesetzt. Danach wird jedesmal, wenn der eigentliche Ticker überläuft (mehr als 65535 Signale = 1 Stunde), der 'Stundenzähler' um eins hochgezählt. Weil die Stundenzahl direkt ausgelesen werden kann, braucht sie nur noch eine Konvertierung nach BCD (Eine Ziffer pro Byte) und nach ASCII (Zahl in Ziffernzeichen) zu erfahren.

```
   Zeit$ # BX MOV
UhrStart # BX SUB
       2 # BX ADD
```

Zunächst muß, wie bei Anzahl, die 'wirkliche' Adresse der Zeichenkette berechnet werden.

```
CX AX MOV
```

Danach wird die Stundenzahl nach AX übertragen, weil die nachfolgenden Rechenoperatoren sich allesamt auf das Register AX beziehen...

```
0A # CL MOV
```

In CX steht jetzt dezimal 10

```
CL DIV
```

Die Division AX durch CL 'trennt' den Inhalt von AX so auf, daß in dem Teilregister AL das Ergebnis der Division steht und im Teilregister AH der 'Rest'. Damit sind Einer- und Zehnerstellen der Stundenzahl voneinander getrennt - einfach aber wirkungsvoll.

```
3030 # AX ADD
```

Wenn man auf jedes Teilregister 30h = dezimal 48 addiert, wird die in diesem Register enthaltene Zahl zu einer ASCII Ziffer...

```
AX 0 [BX] MOV
```

Da das Register BX auf den Anfang der Zeichenkette zeigt, in der die Uhrzeit festgehalten werden soll, braucht der Inhalt des konvertierten AX Registers nur noch dorthin übertragen zu werden.

```
DX AX MOV
```

Der Interrupt 1Ah liefert im Register DX die Anzahl der Signale des Timers, die in der aktuellen Stunde bereits eingelaufen sind. Diese Zahl muß natürlich möglichst exakt in Minuten und Sekunden umgerechnet werden.

```
03E8 # CX MOV
CX MUL
```

Dazu wird CX zunächst mit 1000 geladen und AX mit diesen 1000, mit CX, multipliziert. Das auf 32-Bit erweiterte Ergebnis steht nun in AX und DX !

```
471C # CX MOV
```

471Ch = 18204 dezimal - entspricht der Skalierung der Ticks/Sekunde mit 1000. Diese Zahl ist für die Arbeit der Uhr genau genug.

```
CX DIV
```

Die Division von AX:DX durch CX ergibt nun in AX die Anzahl der Sekunden, die in der aktuellen Stunde 'getickt' wurden.

```
3C # CL MOV
```

3Ch = 60 dezimal...

```
CL DIV
```

Die Division durch 60 ergibt letztlich die Anzahl der Minuten in der aktuellen Stunde, wobei die Minuten im Register AL stehen und die restlichen Sekunden in AH.

```
AH DL MOV
```

Weil die restlichen Sekunden auch noch gebraucht werden, werden sie hier sofort nach DL gesichert.

```
CBW
```

...Convert Byte to Word macht aus dem 8-Bit Wert in AL einen 16-Bit Wert in AX

```
0A # CL MOV
CL DIV
```

Auch die Minuten müssen auf die bereits beschriebene Art und Weise zunächt in BCD-Zahlen und dann nach ASCII konvertiert werden.

```
3030 # AX ADD
AX 3 [BX] MOV
```

Der Inhalt von AX wird wieder in die Zeichenkette transportiert, diesmal aber nicht an den Anfang. Die ersten beiden Zeichen sind bereits mit den Stundenangaben belegt. Das dritte Zeichen ist ein ':'. Folglich muß nun an den Offset 3 auf die Zeichenkette kopiert werden, was dem vierten Byte entspricht (Die Null ist auch eine Zahl :-))

Mit den Sekunden wird nicht anders verfahren, als bisher...

```
DL AL MOV
CBW
CL DIV
3030 # AX ADD
AX 6 [BX] MOV
RET
```

RETurn ist die Antwort auf das CALL und bewirkt einen Rücksprung auf die nächste Anweisung hinter dem CALL.

Die Ausgabe der Zeichenkette auf dem Bildschirm wurde in der UHR in eine eigene Routine gepackt. Dadurch wird der Code insgesamt etwas übersichtlicher und nachvollziehbarer. Der 'Rechenzeitverlust durch das Call und das korrespondierende Ret beträgt insgesamt 41 CPU-Takte und

wird hier einfach vernachlässigt. Wie die Tests auf verschiedenen Rechnern gezeigt haben, ist die Uhr auf allen Systemen 'schnell genug'.

```
Label .Zeit
0F # AH MOV
10 INT
```

Mit der Funktionsnummer 0Fh liefert der BIOS-Interrupt 10h verschiedene Informationen über den aktuellen Videomodus. Unter anderem gibt der Service im Register BH die Nummer der aktuellen Bildschirmseite zurück. Dies ist wichtig, weil die Uhr auch dann nachgeführt werden soll, wenn ein cleverer Programmierer diese Bildschirmseite wechselt. Die Uhr läßt sich nicht 'hereinlegen'.

Der Code der UHR ist im Blick darauf entstanden, grundsätzlich aufzuzeigen, wie man so etwas machen kann.

```
03 # AH MOV
10 INT
```

Der gleiche Interrupt 10h gibt, wenn er mit der Funktionsnummer 03h aufgerufen wird, die Cursorposition auf der aktuellen Bildschirmseite zurück. Nach der Rückkehr aus dem Service steht die Bildschirmzeile im Register DH und die Zeile in DL.

```
DX PUSH
```

Weil diese Position später restauriert werden soll, wird der Inhalt von DX jetzt auf dem Stack zwischengelagert.

```
0A # DH MOV
0A # DL MOV
 2 # AH MOV
10   INT
```

Danach wird in DH die neue Zeile eingetragen, hier 10 dezimal und in DL die neue Spalte, ab der die Uhrzeit ausgegeben werden soll. Und wieder mit dem Interrupt 10h, diesmal mit der Funktionsnummer 2, wird die neue Cursorposition gesetzt.

```
   zeit$ # SI MOV
UhrStart # SI SUB
       2 # SI ADD
```

Das kennen wir schon von den anderen Variablen... SI hält den SourceIndex - einen Zeiger auf den Start der Zeichenkette die ausgegeben werden soll. Daß hier nicht mehr BX verwendet wird hängt damit zusammen, daß der nachfolgende Operator LODS (Load String = Lade Zeichenkette) sich ausdrücklich auf das Register SI bezieht und von INTEL so vorgegeben wird.

```
8 # CX MOV
```

CX ist das Register, das automatisch bei Schleifen angesprochen wird. Insbesondere das Mnemonic LOOP decrementiert CX nach jedem Schleifendurchl auf. CX enthält den Wert 8, weil insgesamt 8 Zeichen ausgegeben werden sollen, im Format "hh:mm:ss"

```
0E # AH MOV
HERE
   AL LODS
   10 INT
LOOP
```

Eine weitere Funktion des Services von INT 10h - die Ausgabe eines Zeichens auf dem Bildschirm. Hier geschieht implizit Folgendes: Das Zeichen auf das SI 'zeigt' wird nach AL geladen und mit dem INT 10h ausgegeben. Dabei wird SI um 1 erhöht. LOOP verringert den Inhalt von CX um '1' und verzweigt wieder nach HERE. (siehe ZF-Assemblerkurs) LOOP arbeitet solange, bis der Inhalt von CX NULL ist.

```
DX POP
2 # AH MOV
10 INT
RET
```

Die auf dem Stack gesicherte 'alte' Cursorposition wird wieder nach DX geladen und restauriert. Damit ist auch die Ausgabe der Zeichenkette abgeschlossen.

Die Uhr muß ihre Arbeit noch 'ordentlich' beenden und die Kontrolle wieder an die unterbrochene Applikation zurück geben. Das besorgt die Subroutine 'Aussprung'.

```
Label Aussprung
SI POP DI POP
ES POP DS POP
DX POP CX POP
BX POP AX POP
IRET
```

Der Aussprung aus der UHR

Aussprung restauriert die Inhalte aller an den Arbeiten der UHR beteiligten Register, und nimmt die entsprechenden Werte vom Stack. Dabei ist es wichtig zu wissen, daß hier mit Stack nicht der Parameterstack des FORTH-Systems gemeint ist. Zur Erinnerung: --- die UHR wird als Anhängsel des Timer Interrupts aufgerufen. Dieser Interrupt, der zu den im Hintergrund ablaufenden DOS-Services gehört, bekommt von DOS ohnehin einen eigenen Stack zu Verfügung gestellt. Diesen benutzt die UHR einfach mit. Das ist durchaus nicht immer unkritisch. Der Programmierer kann nur schwer, oder oft gar nicht, voraussagen, wie weit der Stack des Betiebssystems bereits belastet wird. Die Uhr legt insgesamt, inklusive der Auslagerung der alten Cursorposition, `9 WORDS` auf den Stack - beansprucht also 18 Byte für sich. Das sollte der Stapel des Systems in der Regel verkraften. Wenn aber mehrere solcher Hintergrundtasks ihre Arbeit ähnlich organisieren, kann es durchaus 'eng im Keller' werden !

Von besonderer Wichtigkeit ist IRET, der Interrupt Return. IRET ist die korrespondierende Balance zum Interrupt Aufruf INT. Zu INT wurde bereits gesagt, daß dieser Befehl die Flags und die Register CS und IP auf dem Stack sichert. IRET restauriert diese Register wieder in der korrekten Reihenfolge !

Eine andere, oft von TSRs vorgezogene Version des Aussprunges ist es, die Adresse des originalen Vektors aufzurufen und dessen IRET zu nutzen. Da aber 'hinter' INT 1Ch ohnehin nur IRET steht, kann dieser Befehl auch direkt von der UHR aufgerufen werden.

Was sonst noch so zur UHR gehört

Der CODE der 'eigentlichen' Uhr, wozu nicht die initialisierende und die deinitialisierende Routinen gehören, besteht neben den bis hierher im Detail beschriebenen Routinen noch aus vier weiteren 'Labeln'. In einer Über-

sicht sieht die UHR so aus:

```
LABEL UhrStart
 LABEL Zeit$
    xxxxxxxx
 LABEL Anzahl
    xx
 LABEL Zeit>$
    Code von Zeit>$....
 LABEL .Zeit
    Code von .Zeit
 LABEL Aussprung
    Code von Aussprung
 LABEL Uhr
    Code von UHR
LABEL UhrEnde
```

Die bisher noch nicht besprochenen LABEL Zeit$ und Anzahl bedürfen kaum einer weiteren Erklärung. Sie dienen lediglich als Speicherplätze zur Aufnahme der Variablen Anzahl und der Zeichenkette Zeit$.

Die LABEL UhrStart und UhrEnde haben dagegen besondere, extrem wichtige Bedeutungen - Sie 'markieren' den Start und das Ende des CODE-Teiles der Uhr. Die UHR muß in ein eigenes, vom FORTH-System und von DOS separiertes Segment verlegt werden, wenn sie allen Anforderungen gerecht werden soll. Eine der Anforderungen an die Uhr ist aber, daß FORTH gestartet, die UHR installiert und FORTH wieder verlassen wird - wobei die UHR auch nach dem Verlassen von FORTH noch arbeiten soll. Würde die UHR im CODE-Segment des FORTH-Systems arbeiten, wäre spätestens nach dem BYE des FORTH das ganze System 'kaputt'. Der verbogene Vektor hätte nach wie vor seine Gültigkeit, aber der dafür definierte, neue Code wäre nicht mehr vorhanden. Die Ergebnisse sind undefinierbar und enden sicher in einem totalen Systemabsturz. Weitere Probleme dieser Art können auftreten, wenn FORTH zwar nicht beendet wird, aber der Anwender, wie unter ZF problemlos möglich, mit dem SYS Kommando in die DOS Ebene verzweigt. Und zumindest unter ZF macht der Editor auch noch Ärger. SED arbeitet als Overlay. Bei einem Aufruf lagert der Editor Teile des Dictionary aus und ersetzt diese zeitweilig durch Definitionen, die er für seine eigenen Arbeiten benötigt. Wird der Editor verlassen, restauriert er das Dictionary natürlich wieder - aber zwischenzeitlich gingen alle Einsprünge der UHR 'in die Wicken'- sprich: auch hier ist das Ergebnis undefinierbar und führt zu einem Systemabsturz.

Um nun aber den Code der UHR auslagern zu können, muß die entsprechende Routine exakt wissen, "von wo bis wo" dieser Code reicht. Diese Angaben entnimmt das entsprechende, noch vorzustellende Wort den Adressen von UhrStart und UhrEnde.

Der Übertragung des CODE kommt unter ZF eine Besonderheit dieses FORTH-Systems sehr entgegen. Wort Header und Bodies sind unter ZF von einander getrennt ! Das bedeutet, daß zwischen den Labeln UhrStart und UhrEnde wirklich nur der CODE der dazwischen liegenden Definition steht, sowie die für die 'Variablen' Zeit$ und Anzahl allokierten Byte. Eine kleine Einschränkung muß dieser Aussage allerdings sofort folgen. Jede Forth-Definition trägt im CODE-Segment eine 'Kennung' ein, an welcher der Interpreter 'sehen' kann, um welchen Type von Definition es sich handelt. Schließlich hat dies unterschiedliche Reaktionen zur Folge. (siehe Aufsatz ZF-Intern, Forthgruppe Moers). Diese Kennung bleibt auch bei dem Transport in das separate Segment erhalten und belastet den Code mit jeweils zwei zusätzlichen Byte. Auswirkungen auf die Rechenzeit haben diese 'Kennungen' natürlich nicht. Sie kosten halt ein wenig Platz, der sich auch noch 'wegoptimieren' ließe. Ich habe darauf verzichtet.

Installieren der UHR

Im 'Nicht-Code-Bereich' der UHR, das heißt, in dem Bereich, der nicht in das separate Segment übertragen wird, fallen sicher zunächst die Variablen auf:

- `NSegment`
  bekommt die neue Segmentadresse zugewiesen, an welcher der CODE der UHR arbeiten und wohin die Uhr ausgelagert werden soll.
- `ASegment`
  bekommt die 'alte' Segmentadresse des originalen INT 1Ch Codes (IRET) zugewiesen, so wie
- `AOffset`
  den Offset dieses Codes enthalten wird. Beide Angaben müssen notwendiger Weise festgehalten werden, wenn die UHR sich deinstallieren lassen soll.

Das bedenkenlose Experimentieren mit Interrupts zeitigt nicht selten unvorhergesehene Ereignisse

Das Auslagern der UHR in das separate Segment übernimmt die Funktion mit eben diesem Namen:

```
: Uhr_auslagern ( -- )
[']   UhrEnde   [']
UhrStart  - 16 / 1+
```

...berechnet den Speicherplatz, den die Uhr benötigt. Die Sequenz hat folgende Bedeutung: Mit den Ticks ['] auf die Label werden deren Adressen als Offsets zum CodeSegment des FORTH-Systems auf den Stack gerufen. Die Subtraktion UhrEnde minus UhrStart ergibt im Prinzip die CodeLänge der gesamten Uhr. Weil DOS im 8086/88 Modus mit dem hoffentlich bekannten Problem der Speichersegmentierung zu kämpfen hat, gibt das Betriebssystem stets nur sogenannte Praragraphen von jeweils 16 Byte Länge als Speicher frei. Mit anderen Worten, das kleinste Segment ist unter MSDOS/PCDOS immer noch 16 Byte 'groß'. Weil DOS aber stets nur mit Paragraphen als Parametern arbeitet, muß die zuvor errechnete Bytezahl durch 16 dividiert werden. Dabei kann es natürlich auf Grund der ausschließlichen Integerrechnungen von FORTH zu Fehlern kommen. Reste werden abgeschnitten und nicht von DOS angefordert - die Übertragung klappt sauber - aber die nächste Anwendung überschreibt bis zu 15 Byte der UHR. Um das grundsätzlich auszuschließen, wird am Ende der Speicherberechnung einfach noch einmal 1 zur Sicherheit aufaddiert. Damit ist in jedem Fall Platz genug im neuen Segment vorhanden.

```
ALLOC NSegment !
```

ALLOC fordert die gerade erst berechnete Anzahl von Speicherparagraphen von DOS. ALLOC gibt als Parameter die Segmentadresse zurück, die von DOS für das angeforderte Segment benannt wurde. Diese Adresse wird in der Variablen NSegment festgehalten.

Nun kann die eigentliche Übertragung erfolgen. Das Forth Wort CMOVEL überträgt einzelne Byte von einem Segment in ein anderes und hat die Syntax - *QuellSegment QuellOffset ZielSegment ZielOffset Anzahl* -. Das ForthWort ?CS: gibt die Segmentadresse des aktuellen CodeSegmentes auf den Paramterstack, so daß die Sequenz

```
?CS: [']  UhrStart
```

die Paramter QuellSegment und Quell Offset liefert.

```
NSegment @ 0
```

liefert dagegen das Zielsegment und den Offset darauf - '0'.

Die Anzahl wird noch einmal in der zuvor beschriebenen Weise berechnet (was sich sicher auch auf dem Stack regeln ließe).

```
[']UhrEnde ['] Uhrstart -
```

Listing zu: Die Uhr (Friederich Prinz)

```
\
\       UHR.SEQ
\
\       permanent im Hintergrund arbeitende Uhr
\
\       Friederich Prinz, Forthgruppe Moers,
\       Hombergerstraße 335, 4130 Moers 1
\
\       Die Uhr arbeitet als 'Anhang' an den Timer INT 8h, bzw. 1Ch.
\       Der Code der Uhr wird in ein separates Speichersegment ausgelagert.
\
\       Der Start der Uhr erfolgt aus dem laufenden Interpreter des FORTH's
\       heraus. Bei der De-Initialisierung der Uhr wird der von DOS
\       allokierte Speicher wieder frei gegeben.
\
\       ------------------------------------------------------------------
\
\       Code für ZF Forth
\
\       getestet auf 80386/33, 80286/12, 80286/16, 8086/8, 8088/6/8
\
\       letzte Änderung : 06/09.'92
\
\ ------------------------------------------------------------------------

LABEL UhrStart

LABEL Zeit$
      00 , ASCII : C,                          \ Zeichenkette mit der
      00 , ASCII : C,                          \ Uhrzeit im Format
      00 ,                                     \ "hh:mm:ss"

LABEL Anzahl 00 ,                              \ Anzahl aktuelle Ticks
                                               \ siehe Beschreibung
HEX

LABEL Zeit>$
       0 # AH MOV ( Zeit holen - BIOS )
            1A INT                             \ Zeit in CX, DX
   Zeit$ # BX MOV ( Adresse Zeit$ berechnen )
UhrStart # BX SUB                              \ berechnete Adresse von
       2 # BX ADD                              \ Zeit$ im Zielsegment
        CX AX MOV ( Stunden )                  \ CX = 'echte' Stunden
      0A # CL MOV
            CL DIV                             \ DIV 10 == Konvert to BCD
    3030 # AX ADD                              \ ADD 3030h = Konv. to ASCII
    AX 0 [BX] MOV                              \ Übertragen in Zeit$
        DX AX MOV ( Minuten )                  \ DX = 'Ticks' d. akt. Stunde
    03E8 # CX MOV
            CX MUL                             \ Skalierung mit 1000 dez.
    471C # CX MOV
            CX DIV                             \ DIV Ticks/Sek. * 1000 dez.
      3C # CL MOV
            CL DIV                             \ DIV 60 dez = Minuten
          AH DL MOV                            \ Sekunden in DL sichern
               CBW                             \ Konvert Byte to Word
      0A # CL MOV                              \ Konvertieren nach BCD und
            CL DIV                             \ ASCII und Übertragen in
    3030 # AX ADD                              \ Zeit$
    AX 3 [BX] MOV
        DL AL MOV ( Sekunden )                 \ Sekunden nach AL
               CBW
            CL DIV                             \ Konvertierung
    3030 # AX ADD                              \ s. Stunden, Minuten
    AX 6 [BX] MOV
               RET
```

```
CMOVEL
;
```

schließlich wird der Code Byte für Byte in das neue Segment kopiert.

Nun ist der Transport abgeschlossen. Das heißt, daß der Code im neuen Segment liegt. Von Arbeiten kann noch keine Rede sein. Schließlich muß in der Vektorentabelle noch eingetragen werden, daß der nächste INT 1Ch Aufruf auch in diesem Segment arbeiten muß. Dazu muß der Vektor entsprechend umgesetzt werden.

```
CODE Setz_vektor
AX PUSH BX PUSH
DX PUSH DS PUSH
```

Die beteiligten Register werden wieder gesichert, diesmal auf den Parameterstack des FORTH-Systems !

```
351C#AXMOV
21 INT
```

35h im Register AH ist die Funktionsnummer für den Interrupt 21h - den DOS Interrupt. Die Funktion 35h liest den Vektor eines Interrupts aus. Welcher Vektor ausgelesen werden soll, muß im Register AL spezifiziert sein - hier 1Ch.

```
ES ASegment #)   MOV
BX AOffset #)    MOV
```

Der Service liefert die Segmentadresse im Register ES zurück und den Offset im Register BX. Die Registerinhalte werden in die zuvor besprochenen Variablen kopiert, um bei späteren Restaurationswünschen bereit zu stehen.

```
Uhr # DX MOV
UhrStart # DX SUB
2 # DX ADD
```

Diese Rechnerei kennen wir auch schon. Die Einsprungadresse der Uhr- der Offset auf das separate Segment, beginnt dort, wo der CODE des LABELs UHR beginnt. Dieser Offset muß wieder relativ zum UhrStart errechnet werden.

```
NSegment #) DS MOV
```

Die neue Segmentadresse wird in das Register DS (DatenSegment) geladen

```
251C # AX MOV
```

Die Funktionsnummer 25h im Register AH fordert den Service 'Vektor schreiben' auf, die in den Registern DX und DS enthaltenen Werte an die durch in AL angebene Adresse zu kopieren (siehe Beschreibung Vektorentabelle)

```
21 INT
```

INT 21h führt diese Anforderung aus - ab sofort ist die UHR 'scharf'.

```
DS POP DX POP
BX POP AX POP
```

Die zuvor gesicherten Register werden restauriert....

```
NEXT END-CODE
```

... und der Vektor ist gesetzt !!!!!

Nun möchte der Eine oder der Andere Programmierer die UHR vielleicht wirklich in einer 'vernünftigen' Applikation nutzen. Bitteschön - betrachten Sie die UHR als Public Domain und machen Sie damit, was immer Sie machen wollen. - Sie wollen die UHR wieder aus dem Arbeitsspeicher Ihres PC entfernen ? Dazu fehlt allerdings noch eine kleine Routine:

Fortsetzung des Listings zu: Die Uhr (Friederich Prinz)

```
LABEL .zeit
      0F # AH MOV ( Frage nach Video Mode )
           10 INT                                  \ BH = akt. Bildsch. Seite
      03 # AH MOV ( Frage nach CursorPos )
           10 INT
           DX PUSH                                 \ Cursor auf Stack retten
      0A # DH MOV ( Zeile 10 )
      0A # DL MOV ( Spalte 10 - hier wird die gewünschte Cursorposition )
       2 # ah mov ( gesetzt )
           10 int
   zeit$ # SI MOV                                  \ Adresse der Zeichenkette
uhrstart # SI SUB                                  \ berechnen
       2 # SI ADD
       8 # CX mov
      0E # AH MOV
HERE                                               \ Schleife, Ausgabe von Zeit$
           AL LODS                                 \ Funkt. 0Eh, INT 10h
           10 INT                                  \ LODS (Byte) aus SI nach AL
LOOP                                               \ bis CX == 0
           DX POP                                  \ alte CursorPos restaurieren
       2 # AH MOV
           10 INT
              RET

LABEL Aussprung
      SI POP   DI POP   ES POP   DS POP
      DX POP   CX POP   BX POP   AX POP
      IRET

\ ------------------------------------------------------------------------

LABEL Uhr NOP NOP
      AX PUSH  BX PUSH  CX PUSH                    \ alle beteiligten Register
      DX PUSH  DS PUSH  ES PUSH                    \ werden auf dem Stack von
      DI PUSH  SI PUSH                             \ DOS gesichert !!!
      CS PUSH  DS POP          ( <---- schnelle Segmentzuweisung )
\ ==================
      Anzahl # BX MOV
    UhrStart # BX SUB
           2 # BX ADD
       1 # 0 [BX] ADD
        0 [BX] AX MOV
           6 # AX CMP
\ ------------------------------------------------------------------------
       Aussprung JNE ( AX <> 6 --> Ende )          \ 3 Mal / Sek.
\ ------------------------------------------------------------------------
       0 # 0 [BX] MOV
        Zeit>$ #) CALL                             \ Zeit$ auffüllen ...
        .zeit  #) CALL                             \ ... und ausgeben
\ ------------------------------------------------------------------------
      Aussprung #) JMP                             \ --- und Ende

Label UhrEnde

DECIMAL

\ --- Ende des CodeTeiles der 'eigentlichen' Uhr. Die folgenden Routinen
\ --- regeln das Auslagern in ein externes, zu allokierendes Segment
\ --- und die Initialisierung / Deinitialisierung der Uhr .
```

```
CODE Reset_vektor ( -- )
AX PUSH DX PUSH DS PUSH
```

kein Kommentar - das hatten wir schon.

```
AOffset #) DX MOV
ASegment #) DS MOV
```

... die gesicherten Segment- und Offsetadressen werden wieder in die Register DS und DX geladen ...

```
251C # AX MOV
```

... und der Vektor bekommt wieder seine originalen Werte.

```
21     INT
NEXT END-CODE
```

Nun sollte das von DOS allokierte SpeicherSegment allerdings ebenfalls wieder frei gegeben werden, nicht zuletzt deshalb, weil sonst bei jedem Aufruf der UHR neuer Speicherplatz angefordert wird - so lange, bis einfach kein freier Speicher mehr vorhanden ist. Und dann hieße es sofort wieder, daß die UHR unsauber programmiert sei. NEIN - Die Anweisung

```
NSegment @ DEALLOC
```

übergibt die Adresse des Speichersegmentes an DOS und sagt dem Betriebssystem ausdrücklich, daß es über diesen Speicher wieder nach eigenem Gutdünken verfügen kann. Die Anzahl der diesem Segment zugeschriebenen Paragraphen kennt DOS selbst. Darum brauchen wir uns nicht zu kümmern. Allerdings liefert DEALLOC einen Rückgabewert. '0' bedeutet, daß DOS den Speicher wieder zurückgenommen hat, eine '9' wäre die Nachricht, daß DOS die angegebene Adresse als falsch ansieht. Da wir aber die UHR sauber programmiert und ausgetestet haben, gehen wir einfach davon aus, daß auch die Rückgabe in Ordnung war - und werfen mit DROP den Rückgabewert einfach vom Stack.

Listing zu: Die Uhr (Friederich Prinz)

```
VARIABLE NSegment

VARIABLE ASegment
VARIABLE AOffset

: Uhr_auslagern ( -- )
    ['] UhrEnde   ['] UhrStart - 16 / 1+   \ benötigten Speicher berechnen,
    ALLOC NSegment !                       \ anfordern und SegAdr speichern
    ?CS:   ['] UhrStart                    \ von akt. Seg., Offs. UhrStart
    NSegment @ 0                           \ nach NeuSegm., Offs '0'
    ['] UhrEnde ['] UhrStart -             \ alle Byte zwischen den Labeln
    CMOVEL                                 \ 1:1 übertragen !!!
;

HEX

CODE Setz_vektor
     AX PUSH   BX PUSH   DX PUSH   DS PUSH
     351C # AX MOV                      \ AH = 35h, AL = 1Ch
           21 INT                       \ Vektor auslesen
ES ASegment #) MOV
 BX AOffset #) MOV                      \ Vektor sichern ( im Dictionary )
      Uhr # DX MOV
 UhrStart # DX SUB
        2 # DX ADD                      \ Einsprungadresse berechnen (OFFS)
NSegment #) DS MOV
     251C # AX MOV                      \ Fu 25h, Int 1Ch
           21 INT                       \ neuen Vektor setzen
     DS POP   DX POP   BX POP   AX POP
NEXT END-CODE

DECIMAL

: Uhr_ein ( -- )
  Uhr_auslagern
  Setz_Vektor
;

HEX

\ === Deinitialisierung ================================================

CODE Reset_Vektor ( -- )
  AX PUSH DX PUSH DS PUSH
  AOffset  #) DX MOV
  ASegment #) DS MOV
       251C # AX MOV
             21 INT                     \ alten Vektor INT 1Ch restaurieren
  DS POP DX POP AX POP
NEXT C;

DECIMAL

: Uhr_aus ( -- )
   Reset_Vektor
   NSegment @ DEALLOC                   \ Speicher wieder freigeben
   DROP
;
```

Nacharbeiten

Bei allem Spaß an dieser Exkursion durch ein komplexes Thema, bei allem Spaß an der UHR, die auf ganz unterschiedlichen Systemen einwandfrei arbeitet, bleiben doch noch einige Arbeiten offen. Diese Arbeiten möchte ich interessierten und engagierten FORTHern an's Herz legen, verbunden mit der Aufforderung, die sich sicher schnell einstellenden Ergebnisse ebenfalls in der VD vorzustellen.

Was soll der Anwender mit der Uhr anfangen, wenn sie installiert ist ? Der CODE der UHR ist im Prinzip zweimal vorhanden, einmal im CodeSegment des FORTH-Systems, und einmal in dem Segment in dem die UHR arbeitet. Es wäre nicht von Übel, wenn man den Code einfach aus dem FORTH-Segment hinauswerfen könnte, ohne dabei die Variablen und die (De)Installationsroutinen zu zerstören.

Und was fängt der Anwender an, der die UHR auf dem Bildschirm 'dirigieren' möchte ? Zeile und Spalte der Bildschirmposition sind jetzt fest vorgegeben. Natürlich läßt sich die laufende Uhr 'abschalten', neu as-

semblieren und an neuer Bildschirmposition aufrufen. Aber das ist unbequem und einfach unschön. Auch hierzu sollte ein beherzter FORTHer zeigen, daß die UHR sich die notwendigen Infos zur Bildschirmposition 'woanders' herholen kann.

Die interessanteste Aufgabe unter den Nacharbeiten habe ich für den Schluß aufgehoben.

Wenn die UHR installiert ist und FORTH vollständig verlassen wird, dann hat man alle GURUS Lügen gestraft und ein TSR aus einem laufenden Interpreter heraus installiert. Die UHR arbeitet auch unter DOS einwandfrei weiter. Aber sie läßt sich nicht mehr deinstallieren ! Ein völlig neues Hochfahren des FORTH-Systems bedeutet, daß alle Informationen über die originalen Segmentadressen und über die neue Segmentadresse verloren sind. Eine Installationsroutine müßte diese Informationen ebenfalls 'woanders' her bekommen.

Warnung vor dem Hunde

Auch auf die Gefahr hin ein wenig bissig zu wirken, möchte ich auf keinen Fall eine ernste Mahnung unterlassen. Lassen Sie mich sagen, daß die Programmierung solcher UHRen im Prinzip nicht schwierig ist. Man muß es halt nur können. Aber dieses Können ist eine Voraussetzung, die man nicht ernst genug nehmen kann. Das bedenkenlose Experimentieren mit Interrupts zeitigt nicht selten unvorhergesehene Ergebnisse. Eine falsche Funktionsnummer, ein falscher INT Aufruf - und schnell ist das CMOS des PC mit irgendwelchem Unsinn überschrieben. Das kann die unterschiedlichsten Folgen haben, von einer Systemuhr die nicht mehr 'stimmt' (eines der geringsten Übel) bis hin zu Schäden auf der Festplatte. Ich möchte gerne ausrücklich dazu ermuntern, selbst mit dem Assembler zu experimentieren (welchen Sie auch immer benutzen) und sich selbst an 'forthige' TSRs zu wagen. Die UHR ist schließlich ein hervorragendes Gerüst für die ebenfalls von *Jörg Staben* erbetene Routine eines Bildschirmschoners. Drei Viertel der dazu notwendigen Arbeiten enthält die UHR bereits. Sehr sinnvoll lassen sich TSR Programme auch in ganz anderen Bereichen einsetzen, zum Beispiel bei der Kommunikation zwischen quasiparallelen Prozessen - die erste Aufgabe der 'Nacharbeiten' zielt bereits darauf ab.

Aber bitte, machen Sie sich vorher sachkundig ! Um Ihnen dabei ein wenig zu helfen, habe ich im Anhang des Aufsatzes eine Liste von Büchern zusammengestellt, mit denen ich selbst arbeite und die meines Erachtens rundum empfehlenswert sind.

❑

Bücherliste

PC INTERN
Systemprogrammierung
Tischer - Data Becker
Ich arbeite noch mit der ersten Ausgabe dieses Buches (1987), das damals bereits auf 766 Seiten alles über MSDOS/PCDOS ansprach, was ich als wissenwert eingestuft hatte. Dieses Buch wird in der heutigen Version mit weit über 1000 Seiten ausgeliefert und behandelt bereits Eigenheiten der jüngsten DOS Versionen. Die Version, die ich benutze, wurde in der Fachwelt so begeistert aufgenommen, wie selten ein deutschsprachiges Lehrwerk auf diesem Sektor zuvor. Tischer's Buch ist in so viele Sprachen übersetzt worden, daß nur der Verlag darüber noch Auskunft geben kann. Die PC-Freaks in Frankreich nennen PC Intern sogar 'Le Bible'. Muß man mehr sagen?

8086/8088
Assemberprogrammierung
J.Hegner - KRS
Dieses Buch ist bereits von 1986, aber immer noch 'brandaktuell', zumindest wenn man 8086/88 CPUs programmieren will, oder jüngere INTEL-Prozessoren im REAL-MODE programmiert. J.Hegners Werk enthält Alles über Adressbildungen, Speichersegmentierung und Datentransfers, was es über die PC-Ur-Prozessoren zu sagen gibt. Der Buchteil, der die Befehle der CPU behandelt (selbstverständlich auch die Befehle des Coprozessors) ist ausführlich und mit Beispielen zu jedem einzelnen Mnemonic gespickt. Selbst Laufzeitangaben, bzw. Taktverbrauche der einzelnen Befehle fehlen nicht.

Die PC-Referenz für Programmierer
Thom Hogan
Microsoft Press, Systhema Verlag
Die PC-Referenz ist eigentlich kein Buch, sondern ein Tabellenwerk mit 535 Seiten. Auf diesen 535 Seiten findet sich in einzigartiger Form und dicht gedrängt einfach alles, was irgendwie mit dem PC zu tun hat. Damit wurde *Thom Hogan*'s 'Buch' zu einem unentbehrlichen Nachschlage[illegible] auf meinem Schreibtisch. Allerdings ist es [illegible]nt als 'Lehrbuch' zu gebrauchen - die Informationen sind einfach 'zu dicht'.

MSDOS für Fortgeschrittene Das Microsoft Handbuch zum Programmieren
Ray Duncan
MicroSoft Press, Vieweg Verlag
Ray Duncan's Werk ist eine wertvolle Ergänzung zu Tischer's Buch. Es behandelt einige Eigenheiten in den Tiefen von MSDOS unter einem anderen Blickwinkel als Tischer. Duncan geht im Prinzip die gleichen Themen an, aber weniger unter dem Aspekt eines 'Lehrenden'. Duncan geht davon aus, daß er Programmierer vor sich hat, die auch in MSDOS Interna im Prinzip firm sind, denen aber die eine oder andere Detailinformation fehlt. Duncan ist weniger ausführlich als Tischler, und der Buchteil in dem Duncan auf Interrupts eingeht ist nicht übersetzt worden. Das mag für einige Leser einen nicht wettmachbaren Nachteil bedeuten.

iAPX 86,88 User's Manual - intel
Dieses Buch, das leider auch nur in englischer Sprache erhältlich ist, muß über *intel*, den Hersteller der 8086/88 CPUs selbst bestellt werden. Es geht ausführlich auf alle technischen Aspekte dieser Prozessoren ein und beschreibt darüber hinaus alle peripheren Bausteine wie den Timer unserer UHR, die sich direkt durch 8086/88 CPU eines System auseinander setzen (oder anfreunden) möchte, liegt es eigentlich nahe, sich beim Hersteller selbst zu informieren. Der Informationsgehalt des Manuals ist enorm. Allerdings ist es schwer zu lesen - zum Einen wegen der englischen Sprache, zum Anderen wegen der 'sehr technischen' Texte.

Ronald Zech's Bücher zu FORTH würde ich Ihnen gerne empfehlen. Leider weiß ich vom Franzi's Verlag, daß beide Bücher aus dem Lieferprogramm genommen wurden. Das ist schade, aber unabänderlich. Wer aber guten Kontakt zu einer Buchhandlung hat, sollte versuchen über 'Fernanfrage' herauszufinden, ob nicht doch noch irgendeine Buchhandlung in der Republik ein Exemplar von Zech 'auf Halde' liegen hat. Wenn ja lassen Sie sich das kommen!
Ich leide nicht unter falscher Bescheidenheit - deshalb möchte ich allen denjenigen, die noch nicht wissen, wie 'ihr' Forth Definitionen im Speicher ablegt, den von mir verfaßten Kurs der Moerser Forthgruppe empfehlen - ZF Intern. Auch wenn Sie nicht mit ZF arbeiten (warum eigentlich nicht?) finden Sie in diesem Kurs genügend Anregungen und Hinweise, an Hand derer Sie erfolgreich versuchen können, 'ihr' Forth bis ins innerste zu verstehen.
Wer sich für den Assembler des ZF Systems im Detail interessiert, der sollte ab Anfang Dezember '92 entweder bei der Redaktion der VD oder direkt bei mir nachfragen. Zur Zeit arbeite ich an einem Assemblerkurs für die Moerser Forthgruppe, der vermutlich im kommenden Frühjahr hier anlaufen wird. Dieser Kurs basiert auf dem Assembler des ZF-Systems und zeigt auf, daß sich der einfachste Assembler zu sehr viel mehr nutzen läßt, als nur zur Definition von DUPs und DROPs.

❑

Anmerkung der Redaktion

Zum Thema TSR im F-PC siehe auch das Interview mit Tom Zimmer auf den Seiten 34/35. Tom zeigte auf der Tagung, wie man jetzt in F-PC eine TSR-Uhr programmiert. Der Quellcode wird der neuesten Version mitgegeben.

Vergleich: F-PC-Grafiktreiber

von Bernd Beuster
Carl-Benz-Str. 1a, W-6500 Mainz 1, Tel.: 06131/574333

Wer wissen will, welche gängigen Grafiktreiber es für F-PC gibt, was sie leisten und wie schnell sie arbeiten, findet hier die wichtigsten Informationen.

History

Im Zuge der Meßdatenauswertung benötigte ich einen Grafiktreiber. Der in F-PC mitgelieferte gefiel mir nicht, und so adaptierte ich einen, der ursprünglich als Unit für Turbo Pascal geschrieben wurde.

Ich benutzte diesen Treiber dann in der beruflichen Praxis, beseitigte einige Bugs, vereinfachte die Syntax und transferierte ihn dann in die Münchener Forth Mailbox (VGA-GRAPH.LZH).

Dann las ich das Editorial der VD 3/92. Ich schrieb also *Rolf Kretzschmar* und bot an, über meinen Grafiktreiber einen Artikel zu schreiben. Daraus wurde dann ein Vergleich von mehreren Grafikpaketen, wobei der Quellcode von zweien im 100kB- Bereich liegt.

Und hier ist er: Der ultimative Grafiktreibervergleich...

Beginn der Vorstellung:

Treiber für Grafik

Für die Text- und Grafikausgabe gibt es im BIOS den Videointerrupt $10. Das Register AH spezifiziert dabei dessen einzelnen Funktionen.

Für das Einstellen der Video-Modi existiert die Funktion $00 (AH=$00 AL=Mode). Die am häufigst verwendeten Grafikmodi sind in der Tabelle 1 aufgelistet.

Abweichend davon wird bei den Video7- und VEGA-VGA der Videomodus mit AX=$6F05 und BL=Modus eingestellt. Für den Zugriff auf den VESA-Modus gilt AX=$4F02 und BX=Modus.

Desweiteren gibt es noch die Grafikmodi der Super-VGAs (SVGA). Da aber hier jeder Hersteller seinen eigenen Standard kreiert hat kann hier keine allgemeingültige Tabelle angegeben werden.

Klein und schnell: VGAGRAPH

Mein Grafiktreiber sollte folgende Eigenschaften besitzen:

- *klein* damit noch genug Platz für die Applikation bleibt
- *schnell*, denn es sollten auch Oszilloskopfunktionen ein Echtzeit dargestellt werden
- *einfache Syntax*

Ursprünglich wurde der Treiber nur für den Modus $12 vorsehen, dann aber später auf den Modus $10 erweitert.

Eine niedrigere Auflösung wird meist nur für Spiele und Pixelimages verwendet. VGA-Grafikkarten dürften Stand der Technik sein, so daß auf CGA und Hercules verzichtet wurde. Ebenso gibt es keine Palettenroutinen bzw. komplexere Algorithmen, wie Flächen füllen und Vektortexte. Ebensowenig gibt es Dateioperationen mit Grafiken.

In alten c't-Ausgaben entdeckte ich einen interessanten Artikel von Jürgen Petsch [1-3], der die Grundlage für den Grafiktreiber bildete. Als Primitives, selbstverständlich in Assembler, dienen folgende Worte:

`PUTPIXEL ( x y -- )`
setzt ein Pixel

`GETPIXEL ( x y -- Color )`
ermittelt die Farbe eines Pixels

`LINE ( x1 y1 x2 y2 -- )`
zieht eine Linie

`LINEV ( x y Länge -- )`
zieht eine vertikale Linie

`LINEH ( x y Länge -- )`
zieht eine horizontale Linie

`CIRCLE ( x y Radius -- )`
zeichnet einen Kreis

Dabei sind `LINEV` und insbesondere `LINEH` optimierte Versionen, die nicht den Bresenhamalgorithmus benötigen. Bei `LINEH` werden noch zusätzlich bis zu je acht Pixel zu einem Byte zusammengefaßt, so daß die Portzugriffe auf die VGA-Karte minimiert werden. Auch nicht alltäglich: Längen kleiner/gleich Null sind erlaubt. Das vereinfacht die Programmierung und verhindert unerwünschte Effekte. Der Preis dafür ist ein kleiner Laufzeitoverhead, besonders bei `LINEH` (siehe Benchmark).

Stichworte
F-PC
Grafiktreiber
Benchmark

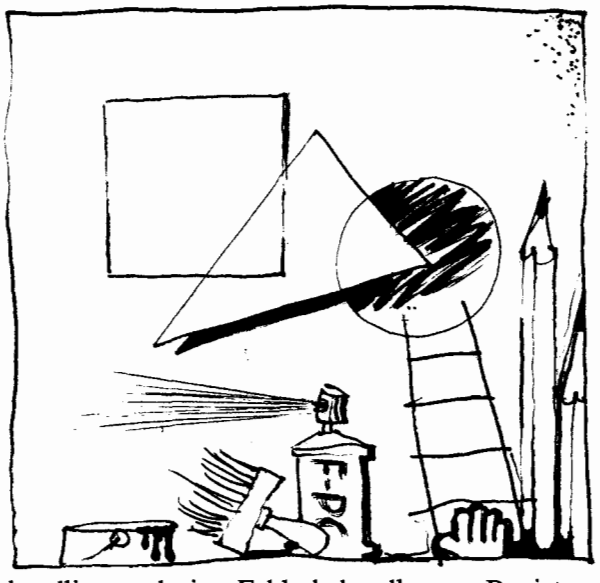
Grafik-Modus	Pixel	Farben	Speicher in Byte	ab Generation
$04	320*200	4	16000	CGA
$05	320*200	4	16000	MCGA
$06	640*200	2	16000	CGA
$0D	320*200	16	32000	EGA
$0E	640*200	16	64000	EGA
$0F	640*350	2	28000	EGA
$10	640*350	16	112000	EGA
$11	640*480	2	38400	VGA
$12	640*480	16	153600	VGA
$13	320*256	256	64000	VGA
Hercules	720*348	2	31320	

Abb. 1

Alle Primitivworte arbeiten ohne eine Fehlerbehandlung auf Überschreiten von Bildschirmgrenzen, um eine maximale Geschwindigkeit zu erhalten. Das ist besonders bei CIRCLE zu beachten: zeichnet man über die Bildschirmgrenzen hinweg, so erhält man dank der modulo-Adressierung über den Bildschirm verstreute Kreissegmente.

GETPIXEL und CIRCLE werden nur selten genutzt, so daß man sie, um Codespeicher zu sparen, auch weglassen könnte.

Eine Ebene höher definiert sind die Worte, welche ein einfacheres Stack-



handling und eine Fehlerbehandlung zulassen. Wer einen ZX- Spectrum besaß, wird sicher die Worte PLOT und DRAW wiedererkennen:

PLOT (x y --)
zeichnet einen Punkt

DRAW (x y --)
zieht eine Linie von der letzten Koordiante nach x,y

RPLOT (dx dy --)
wie PLOT, mit relativen Koordinaten

RDRAW (dx dy --)
wie DRAW, mit relativen Koordinaten

BOX (dx dy --)
zeichnet ein Rechteck mit Breite dx und Höhe dy

Diese Worte speichern ihre letzte Koordinate in die 32bit-Variable XY und führen ein Clipping auf die Bildschirmgrenzen HDOTS und VDOTS durch. Für ein Zeichnen in mathematischen Koordinaten, d.h. y-Koordinate zeigt nach oben, dienen die Worte PLOT' ... RDRAW'.

Die DRAW-Befehle sind smart, d.h. sie erkennen selbständig, ob sie die schnellen LINEH- und LINEV-Befehle nutzen sollen.

Hier noch einige Worte für den allgemeinen Gebrauch des Grafiktreibers:

HDOTS (-- x)
VALUE für maximale x-Koordinate

VDOTS (-- y)
VALUE für maximale y-Koordinate

SETCOLOR (Color --)
setzt Vordergrundfarbe

SETBKCOLOR (Color --)
setzt Hintergrundfarbe im Grafikmodus; im Textmodus wird die Rahmenfarbe gesetzt

VGA-MODE
bereitet den Modus $12 vor

EGA-MODE
bereitet den Modus $10 vor

GRAPH-MODE
schaltet in den Grafikmodus

TEXT-MODE
schaltet in den Textmodus

Logische Verknüpfungen zwischen den Daten (d.h. die 4-Bit- Farbinformation jedes Pixels) im Latch Register der VGA-Karte und den von der CPU beziehungsweise vom Set Reset Register kommenden Daten werden über das Data Rotate Register ermöglicht. Die Befehle dafür sind:

SETMOVEMODE
SETANDMODE
SETORMODE
SETXORMODE

Dabei werden im MOVE-Mode, die Daten unverändert übernommen, das ist die am häufigsten verwendete Betriebsart.

Der XOR-Mode ist ebenfalls interessant, denn damit läßt sich durch Überschreiben mit derselben Farbe auf derselben Bildschirmposition ein Pixel wieder löschen. Das funktioniert auch mit den Linienalgortithmen, so daß sich animierte Sequenzen darstellen lassen.

Dabei ist bei der Programmierung zu beachten, daß das BIOS, wenn es auf den Videocontroller zugreift (z.B. bei TYPE), immer in den MOVE-Mode zurückschaltet. Gerade im interaktivem Test ist also ein vom MOVE-Mode abweichender Modus immer auf der Kommandozeile mit einzugeben.

Wer den Treiber ANSI.SYS verwendet, kann ANSICOLR.SEQ einbinden. Dadurch werden auch im Grafikmodus die Textausgaben farbig gestaltet.

Für das Umschalten von Bildschirmseiten (page flipping), wobei man im nicht sichtbaren Bildschirm zeichnet und dann schlagartig umschaltet, gibt es auch einige Befehle:

SET-PAGE (n --)
Auswahl der aktuellen Bildschirmseite

SHOW-PAGE (n --)
Anzeige der aktuellen Bildschirmseite
WAITSYNC wartet auf den vertikalen Zeilenrücklauf

Dabei ist zu beachten, daß das page flipping nur im EGA-Modus mit Sicherheit funktioniert, sofern man nicht eine VGA-Karte mit mehr als 256kB Speicher besitzt.

Zusätzlich enthält VGA-GRAPH.LZH noch ein Spritepaket für den Modus $13, der mit dem Dirty-Rectangle-Algorithmus von Michael Abrash arbeitet [6,7]. Außerdem gibt es noch eine Integer- bzw. Floatingpointversion der AKIMA-Interpolation [8,9], mit deren Hilfe man anhand von wenigen Stützstellen einer Funktion, deren Zwischenwerte errechnen kann.

Grüße aus dem Koalaland: Bob Gunton's Simple Graphics

Dieser Grafiktreiber des Australiers Bob Gunton wurde zu einer Zeit entwickelt, als noch die XTs die meist verbreiteten Rechner waren. Deswegen legte er besonderes Augenmerk auf Geschwindigkeit und arbeitet extensiv mit Lookuptabellen, was zu Lasten des Codesegmentes geht.

Unterstützt werden EGA und VGA-Karten im Grafikmodus $10 bzw. $12:

EGA
schaltet in den Modus $10

V640
schaltet in den Modus $16

LO
schaltet zurück in den Textmodus (mit Screensave)

X-MAX, Y-MAX
VARIABLEn, enthalten die maximale Auflösung des gewählten Modus

Der LO-Befehl mußte bei mir immer zweimal aufgerufen werden, damit der Cursor wieder erscheint. Die wichtigsten Grundbefehle lauten:

WIPE
löscht den Grafikbildschirm (mit Screensave)

COLOR (n --)
setzt Vordergrundfarbe

BKGRD (n --)
setzt Hintergrundfarbe

PLOT (x y --)
zeichnet ein Pixel an x,y

LINE (x1 y1 x2 y2 --)
zeichnet eine Linie von x1,y1 nach x2,y2

ELLIPSE (x y xrad yrad--)
zeichnet eine Ellipse mit Zentrum x,y und den Radien xrad,yrad

M-O (x y --)
speichert Zentrumskoordinaten für den ARC- Befehl in die Variablen MX, MY ARC (xr yr w1 w2 --) zeichnet einen Ellipsenbogen mit Zentrum in MX, MY, den Radien xr,yr vom Anfangswinkel w1 bis zum Endwinkel w2

INIT-TT (x y --)
setzt die Anfangskoordinaten für den TT-Befehl

ANGLE (-- a)
VARIABLE, enthält den Startwinkel für den TT- Befehl

TT-IND (-- a)
VARIABLE, enthält die um Faktor 10000 skalierte Aspectratio (für EGA-Karten)

TT (w l --)
allgemeinster Turtlegrafikbefehl, zeichnet Linie der Länge l mit dem relativen Winkel w

BOX (x1 y1 x2 y2 --)
zeichnet ein Rechteck mit der Diagonalen x1,y1 x2,y2

FILLBOX (x1 y1 x2 y2 --)
zeichnet ein gefülltes Rechteck

Ebenso wie in VGAGRAPH gibt es Befehle für die Art der Verknüpfung der Farbwerte mit den schon gezeichneten Pixeln:

NLOGIC, ALOGIC, OLOGIC, XLOGIC normale, AND-, OR- bzw. XOR- Verknüpfung

Im Gegensatz zu VGAGRAPH wird hier eine Variable LOGIC gesetzt, so daß nicht die Gefahr der Überschreibung des Modus durch das BIOS gegeben ist.

Zur Erleichterung der interaktiven Erstellung von Grafiken aus Grundelementen, können die Grafikscreens in den Hauptspeicher gesichert und aus diesem wieder restauriert werden. Das funktioniert nur mit COLOR 15 (weiß) da nur eine Bitplane von 37.5kByte gesichert wird. Drei Bereiche zu je 40kByte stehen ab der Adresse $7D000 (500k) zur Verfügung, wo direkt in den Hauptspeicher ohne Memoryallocation geschrieben wird. In F-PC sollte man dagegen das POINTER-Konzept nutzen, um unerwünschte Rechnerabstürze zu vermeiden.

500K, 540K, 580K
selektiert den Screenpuffer an den Adressen 500kByte..580kByte

S-S
sichert Screen in selektierten Screenpuffer

G-S
restauriert Screen aus selektierten Screenpuffer

S-500, S-540, S-580
selektiert Screenbuffer, sichert Screen

G-500, G-540, S-580
selektiert Screenbuffer, restauriert Screen

CUT (x1 y1 x2 y2 --)
sichert einen Bildschirmausschnitt in den selektierten Screenpuffer

PASTE (x y --)
fügt den Inhalt des selektierten Screenbuffers an der Position x,y ein

Bei den CUT- und PASTE-Befehlen muß die x-Koordinate auf einer Bytegrenze liegen, d.h. sie muß ein Vielfaches von 8 betragen.

Ein objektorientierter Ansatz für die interaktive Gestaltung von Grafiken entsteht mit den Worten +LINE, +ELLIPSE und +ARC. Diese Worte erwarten auf dem Stack diesselben Paramter wie ihre gleichlautenden Pedants ohne +, speichern aber zusätzlich diese Parameter und ihren Typ in einem Kurvenarray ab.

1CV, 2CV, 3CV, 4CV
selektiert eine von (z.Z.) vier Kurven (Objekte) des Kurvenarrays

PARAMS
zeigt die Parameter und den Typ der selektierten Kurve (Objekts) an

Diese Objekte können mit folgenden Befehlen manipuliert werden:

+START (n --)
zeichnet +ARC um n Grad früher

+END (n --)
zeichnet +ARC um n Grad später

MOVR, MOVL, MOVD, MOVU (n--)
Rechts-, Links-, Abwärts- bzw. Aufwärtsverschiebung um n Pixel

WIDER (n --)
horizontale Streckung um n Pixel

SLIMMER (n --)
horizontale Stauchung um n Pixel

DEEPER (n --)
vertikale Streckung um n Pixel

THINNER (n --)
vertikale Stauchung um n Pixel

Die Manipulationen können auch automatisch mit der Geschwindigkeit, die in der VARIABLE SPEED enthalten ist erfolgen:

//
verdoppelt die Geschwindigkeit

\\
halbiert die Geschwindigkeit

RT, LT, DN, UU
verschieben das Objekt nach recht, links, unten bzw. oben, bis ein Tastendruck erfolgt

WDN, SLM, DPN, THN
horizontale bzw. vertikale Strekkung/Stauchung des Objekts bis ein Tastendruck erfolgt.

Sind die Objekte in der gewünschten Form und Position, so werden ihre Parameter mit PARAM abgefragt und in die Applikation eingebaut.

Ausgereift: Mark Smiley's Grafikpaket

Dieser Grafiktreiber unterstützt folgende Grafikkarten:

- ☐ CGA
- ☐ EGA
- ☐ Standard VGA
- ☐ Video7 SVGA

Nach dem Laden von GRAPHICS.SEQ erscheint ein Bildschirmtext, der einen Einstieg in das Grafikpaket ermöglicht. Es sind die Befehle:

CHOOSE-RES
wählt über Menü einen Grafikmodus

SET-RES
Umschalten in den gewählten Grafikmode.

DOT (x y color --)
zeichnet einen Punkt mit der gegebenen Farbe

CLIP-DOT (x y --)
zeichnet einen Punkt unter Nutzung der Variablen

COLOR LINE

`( x1 y1 x2 y2 -- )`
zeichnet eine Linie

`ELLIPSE ( xc yc a0 b0--)`
zeichnet eine Ellipse mit Zentrum xc,yc und Halbachsen a0,b0

`CLIP_FILL_ELLIPSE`
`( xc yc a0 b0 -- )`
wie ELLIPSE, aber gefüllt und mit Clipping an den Bildschirmgrenzen

`CIRCLE ( x y r -- )`
zeichnet einen Kreis mit Mittelpunkt x,y und Radius der Länge r

`CLIP_FILL_CIRCLE`
`( x y r -- )`
wie CIRCLE, aber gefüllt und mit Clipping an den Bildschirmgrenzen

`#COLORS`
VARIABLE, enthält die Anzahl der maximal Verfügbaren Farben in der gewählten Bildschirmauflösung

Zwei wichtige Befehle fehlen noch in obiger Auflistung:

`READ-DOT ( x y -- color )`
gibt die Pixelfarbe der Bildschirmposition x,y zurück

`TEXT`
schaltet zurück in den Textmodus.

Die Linienzeichenbefehle sind ebenfalls in diagonale und orthogonale Typen unterteilt, so gibt es neben `LINE` noch:

`HLINE ( x1 x2 y -- )`
zeichnet eine horizontale Linie

`VLINE ( y1 y2 x -- )`
zeichnet eine vertikale Linie

und zur Vereinfachung des Stackhandlings:

`MOVETO ( x y -- )`
speichert die Koordinaten in die globalen Variablen X2,Y2

`NLINE ( x y -- )`
zeichnet eine Linie von X2,Y2 nach x,y. X2,Y2 werden aktualisiert.

Für die Grafikalgorithmen im XOR-Modus gibt es Wörter wie `XDOT, XLINE, XHLINE, XELLIPSE` usw.

Für das Zeichnen von Rechtecken existiert eine Kombination:

`INIT.RECT (x1 y1 x2 y2--)`
speichert die Rechteckkoordinaten in Variablen

`RECTANGLE`
zeichnet das Rechteck im XOR-Modus

die ich syntaktisch aber für nicht für sehr gelungen ansehe.

Für Kompatibilitätsfanatiker wird auf Kosten der Laufzeit die Möglichkeit geboten, die Grafikoperationen über den Video-BIOS- Interrupt laufen zu lassen:

`BIOS_OR_DIRECT?`
wählt über Menü BIOS- bzw. DIRECT-Grafik

`BIOS_GRAPHICS`
kompatible, langsame BIOS-Grafik

`DIRECT_GRAPHICS`
schnellerer Modus, greift direkt auf die Hardware zu

Sehr gut unterstützt werden die Palettenoperationen. Mehrere vorgefertigte Paletten nehmen einem viel Programmierarbeit ab und erleichtern das Erstellen neuer Paletten für eigene Anwendungen.

Auch für das Speichern und Laden mühsam errechneter Fraktalgrafiken ist gesorgt:

`"BSAVE ( addr count -- )`
speichert Image in File, addr zeigt auf Filenamen

`"BRECALL ( addr count --)`
lädt Image von File, addr zeigt auf Filenamen

`BSAVE filename`
wie "BSAVE,aber interaktiv. filename kann entfallen

`BRECALL filename`
wie "BRECALL,aber interaktiv. filename kann entfallen

In `FUNCPLT.SEQ` wird ein einfacher Treiber zum Zeichnen von mathematischen Funktionen bereitgestellt:

`REGION ( FS: xl xu yl yu -- )`
definiert den Wertebereich der Funktion

`GRAPH ( functionname (--)`
zeichnet die Funktion
`functionname ( FS: x -- x y )`

`CONNECT? ( -- a )`
VARIABLE, wenn `CONNECT?` ON ist, dann werden die einzelnen Werte der Funktion grafisch miteinander verbunden, ansonsten werden sie als Einzelpunkte dargestellt.

Fast wie BGI: Ingo Mathyls Grafikdispatcher

Wie in der VD 3/92 erwähnt, schrieb *Ingo Mathyl* einen Grafiktreiber, der die unterschiedlichsten Grafikmodi des IBM-PC auf einen gemeinsamen Nenner bringt.

Der Grafikdispatcher erkennt selbständig folgende Grafikadapter:

- MDA
- CGA
- Hercules
- EGA
- MCGA
- VGA (mit Subsystem)
- CGA-Emulation auf Hercules

Von den SVGA-Grafikkarten werden unterstützt:

- Tseng ET3000, ET4000
- Trident
- Paradise
- Genoa
- ATI
- Video 7
- OAK
- Chips & Technologies
- VESA Standard

Der HiColormodus der SVGAs ist ebenfalls verfügbar. Für die Besitzer einer 8514/A wird ein Wortschatz zu Bedienung des Application-Interface bereitgestellt. Der Grafikdispatcher berücksichtigt jede Kombination der unterstützten Karten, so ist es beispielsweise möglich, eine VGA/SVGA-Karte parallel zur Herkuleskarte laufen zu lassen.

Bei der Einstellung der Grafikmodi werden auch die Seitenverhältnisse des Bildschirms (Aspect Ratio) berücksichtigt, was für die Erzeugung unverzerrter Kreise wichtig ist.

Für jeden Modus gibt es zwei Zustände: `HiResMode` und `MaxColorMode`. So ist beispielsweise für Standard-VGA der HiRes-Mode gleich dem Modus $12 und der MaxColorMode gleich dem Modus $13.

Nach dem Laden von `DISPATCH.SEQ` gelangt man in ein Menü, wo man sich die benötigte Konfiguration zusammenstellen kann. Für diejenigen Worte, welche nicht geladen werden sollen, gibt es alternativ auch Dummy-Worte, welche nur die korrekten `DROP`-Funktionen durchführen.

Der Einstieg in den Grafikdispatcher ist denkbar einfach: nach einmaligem `AUTO-INIT` und wahlweisem `MaxColorMode` bzw. `HiResMode` wird mit `GRAF-MODE` in den Grafikmodus geschaltet.

Was nach der Umschaltung in den Grafikmodus sofort angenehm ins Auge fällt, ist die Darstellung der F-

PC Statusanzeige in Farbe, sowie ein Cursor. Damit wird die interaktive Programmierung im Grafikmodus wesentlich erleichtert. Die Hilfemenüs des Grafikdispatchers wurden in die F1-Online-Hilfe von F-PC integriert.

Zur Initialisierung des Grafikmodus gibt es AUTO-INIT, welches selbständig den höchst verfügbaren Grafikmodus erkennt, bzw. die zum Grafikmodus gehörenden expliziten Wörter:

AUTO-INIT
initialisiert den höchst verfügbaren Grafikmodus

HGC-INIT
initialisiert den Herculesmodus

CGA-INIT
initialisiert den CGA-MODUS

EGA-INIT
initialisiert den EGA-MODUS

VGA-INIT
initialisiert den VGA-MODUS

SVGA-INIT
initialisiert den SVGA-MODUS

GRAF-MODE
schaltet in den Grafikmodus

TEXT-MODE
schaltet zurück in den Text-Mode

Die Grundoperationen mit Pixeln werden mit folgenden Worten durchgeführt (mit Alias-Definitionen):

SET, SetPixel, PLOT
(x y --)
zeichnet einen Punkt an x,y

XSET, XSetPixel, XPLOT
(x y --)
wie SET, aber im XOR-Modus

READ, ReadPixel, POINT
(x y -- color)
liest die Farbe eines Pixels an x,y

Die LINE-Syntax ist analog dem F-PC Grafikpaket:

LINE (x1 y1 x2 y2 --)
zeichnet eine Linie von x1,y1 nach x2,y2

LINEFROM (x y --)
speichert die Koordianten für nächste LINE-Operation

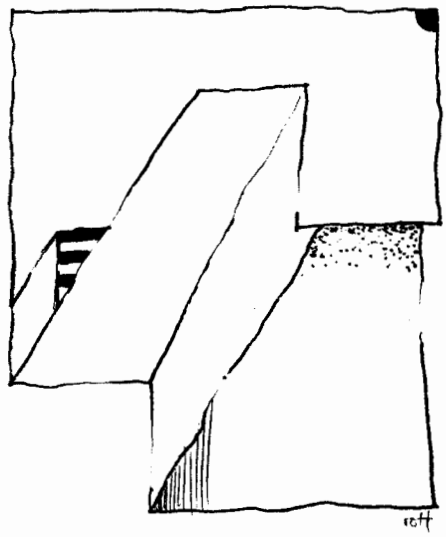
LINETO (x y --)
von letzter Cursorposition nach x,y zeichnen

LINEREL (dx dy --)
wie LINETO, aber mit relativen Koordianten

Dabei werden jeweils die Endkoordinaten gespeichert, so daß das Stackhandling vereinfacht wird.

Das Wort LINE ist smart, d.h. horizontale und vertikale Linien werden extra behandelt, so daß sich eine wesentliche Laufzeitverbesserung ergibt. Leider existiert keine LINE-Version für den XOR-Modus.

Für Ellipsen, Kreise, Rechtecke, 3D-Balken und Polygone kann man



zwischen Darstellungen mit bzw. ohne Füllmuster wählen.

FILLCOLOR (adr --)
VARIABLE, bestimmt die Farbe des Füllmusters

BGRFILLCOLOR (adr --)
VARIABLE, bestimmt die Farbe des Hintergrundfüllfarbe (nur EGA/VGA)

VOLL, FILLING-ON
Darstellung mit Füllmuster

LEER, FILLING-OFF
Darstellung ohne Füllmuster

STYLE (n --)
Wahl des Füllmusters n

STYLES (-- a)
VARIABLE, zeigt auf eine Gruppe von Füllmustern

Es sind schon 16 Füllmuster vordefiniert, sie können aber durch eine Umschaltung des Pointer STYLES beliebig erweitert werden.

Ellipsen und Kreise sind vollständig in Highlevel programmiert, was aber in den meisten Fällen kein Nachteil sein dürfte, was die Geschwindigkeit angeht:

KREIS, CIRCLE (r xm ym--)
zeichnet einen Kreis um Mittelpunkt xm, ym mit Radius r

ELLIPSE (a0 b0 xm ym --)
zeichnet eine Ellipse um xm,ym mit x-Radius a0 und y-Radius b0

Rechtecke und Polygone:

BAR (xo yo xu yu --)
zeichnet Rechteck mit den diagonalen Eckpunkten xo,yo und xu,yu

POLYGON
(x1 y1 ... xn yn n --)
zeichnet ein geschlossenes Polygon mit n-Ecken

Zum Zeichnen von 3D-Balken vewendet man:

!BAR
(Länge Höhe Tiefe x y --)
definiert die Abmessungen und die Koordinaten des zu zeichnenden 3D - Balkens

DRAWBAR (Style --)
zeichnet einen 3D-Balken, mit Füllmuster Style der Vorderfläche, der zuvor mit !BALKEN definiert wurde

3D-BAR
(Länge Höhe Tiefe x y --)
zeichnet einen 3D-Balken mit voreingestelltem Füllmuster

Textausgaben mit zoomed strings ergeben weitere Möglichkeiten. Die zoomed strings können in der Größe, Farbe und Form manipuliert werden:

ZEMIT (c --)
EMIT für zoomed character

ZOOMSTRINGL
(seg off len xzoom yzoom xout yout --)
Ausgabe eines zoomed string

ZOOMSTRING
(addr len xzoom yzoom xout yout --)
wie ZOOMSTRINGL, nicht segmientierte Version

ZGOTOXY (x y --)
Ausgabeposition für ZEMIT, .Z usw.

KURSIV-ON
kursive Zeichendarstellung einschalten

KURSIV-OFF
kursive Zeichendarstellung ausschalten

!STRINGPARA
(xzoom yzoom x y --)
definiert zoomed Stringparameter

ZOOMSTYLE (-- n)
VALUE, enthält Füllmuster für zoomed strings.

Für die Ausgabe von Zahlen gibt es Worte wie .Z, D.RZ usw.

Palettenoperationen werden in Abhängigkeit vom Grafikmodus unterstützt. Für Grafikmodi, welche keine Pallettenoperationen zulassen gibt es entsprechende Dummys.

GetAllPal (firstReg

`#Regs Segment Offset -- )`
lesen des Inhalts aller Palettenregister nach Segment Offset

`SetAllPal ( firstReg #Regs Segment Offset -- )`
schreiben aller Palettenregister

`GetPalReg (#Reg--DColor)`
Palettenregister lesen

`SetPalReg (#Reg DColor--)`
Farbe DColor in Palettenregister schreiben

Die Farbe `DColor` ist ein Doublewert, um den 24-bit RGB-Farbwert zu beschreiben.

`VSWITCH ( bank -- )` schaltet bei den SVGA zwischen den Speicherbänken um, auch gibt es einige Worte, welche die Synchronisation mit dem Zeilenrücklaufimpuls des Videocontrollers vornehmen, so daß sich ein flickerfreier Bildschirmaufbau erreichen läßt.

Zum Schluß noch einige Worte zum Schreiben und Laden von Bildschirmausschnitten im EGA/VGA 16 Farben- bzw. VGA/MCGA 256 Farbenmodi:

`Image> ( filename ( -- )`
setzt Imagehandle auf filename

`PutImage16, PutImage256 ( x1 y1 x2 y2 -- )`
speichert Bildauschnitt in die Datei filename

`GetImage16, GetImage256 ( x1 y1 -- )`
lädt den Bildauschnitt der Datei filename nach x1, y1

`Screendump ( ye ya -- )`
Hardcopy des Grafikbildschirms von Zeile ya bis ye

`Hardcopy ( -- )`
Hardcopy des gesamten Bildschirmes

Die Hardcopyroutinen sind für EPSON-kompatible 9/24-Nadeldrucker ausgelegt. Durch Ändern der Escape-Sequenzen sollte es aber möglich sein, abweichende Druckertypen anzupassen.

Die Demos

VGAGRAPH-Demos

Die Demos in `VGAGRAPH` sind relativ bescheiden. Teilweise sind sie in den Sourcen nach einem `\S` anzutreffen; sie dienten in der Entwicklungsphase einem guten Zweck und wurden daher dort belassen. Das eigentliche Demofile `VGADEMO.SEQ` enthält je ein Demo zum Zeichnen von 100 Rechtecken und Kreisen - auch im Hintergrund des EGA-Modus mit `SET-PAGE` und `SHOW-PAGE`.

Ein sich um die Y-Achse drehender Würfel demonstriert die Fähigkeiten der Animation im XOR-Modus.

Im Demo von `SPRITE.SEQ` werden eine Anzahl sich frei bewegender Sprites, die ihr Aussehen während der Animation verändern von einem "Killersprite" gelöscht.

BOB GUNTON

Die Demos sind alle in dem File `SHOWOFF.SEQ` enthalten. `4WAVE` zeigt eine animierte Sequenz mit `CUT` und `PASTE`, während die anderen Demos die Fähigkeiten des Linienalgorithmus demonstrieren.

MARK SMILEY

Am ausführlichsten von den drei Grafikpaketen sind die Demos von *Mark Smiley*. Gleich acht Files stehen zur Verfügung; bei jedem Demo erscheinen ausführliche Hinweise, so daß die Demos selbsterklärend sind:

- LAND.SEQ
  Fraktale Landschaften nach [10].
- JULIA.SEQ
  Julia-Mengen
- BARNEX.SEQ
  Fraktale Farne, Blätter und sonstige geometrische Figuren
- FUNCT.SEQ
  einige mathematische Funktionen
- A-XX.SEQ
  Fraktal
- FTURT.SEQ
  Ein Turtlegrafikpaket, wie man es von LOGO kennt. damit lassen sich sehr schön rekursive Grafiken erstellen. Für Besitzer von VGA-Karten ist zeigt TURT_DEMO die Fähigkeiten dieses Paketes.
- GIRL.SEQ
  Lädt ein VGA Bild im Modus $13. Zeigt die Möglichkeiten von Palettenoperationen.
- VKAL.SEQ
  Palettenoperationen mit dekorativen Grafikmustern.

Einige Demos verfügen über pull-down-Menüs, so daß sich vielfältige Variationen ergeben. Es empfiehlt sich vorher `GRDEMOS.DOC` zu lesen, da in vielen Demos Floatingpoint bzw. VGA benötigt wird.

GRAFIKDISPATCHER

Es gibt zwei Demos: **GRAFTEST** (in TESTS.SEQ) und `SHOWTIME.EXE`. Ruft man `GRAFTEST` auf, so erhält man ein Demo, ähnlich dem BGI-Demo von Borland. Es werden getestet:

- Pixel Schreiben und Lesen
- Grafiktextmanipulation mit gezoomten und gemusterten Strings
- gefüllte und leere Kreise
- gefüllte 3-D-Balken
- Ellipsen
- Linien (`LINE`, `LINETO`, `LINEREL`)
- gefüllte Polygone
- kompletter Farbdurchlauf der Füllmuster

Dabei wird von dem höchst auflösenden Grafikmodus ausgegangen, der mit `AUTO-INIT` erkannt wurde, und dann werden alle Grafikmodi von CGA bis SVGA in steigender Auflösung getestet.

`SHOWTIME` ermöglicht ein menügesteuertes Testen des Grafiktreibers. Der Schwerpunkt liegt hier bei der Manipulation der Farbpaletten.

Benchmarks

Speicherplatz

Die Größe des Speicherplatzes

	Code	List	Head	Total
VGAGRAPH	1784	1216	1346	4346
GUNTON	6287	2608	2478	11373
SMILEY	9648	18336	8442	36426
GRAFDISP	16298	15920	12669	44887
SPRITES	1440	608	693	2741

Abb. 2

wurde mit dem USED-Befehl von F-PC ermittelt. Dabei wurde jeweils von der Maximalkonfiguration ausgegegangen, so daß Korrekturen nach unten bei den einzelnen Grafikpaketen möglich sind. Die SPRITES vom VGAGRAPH-Paket wurden extra gezählt, da die anderen Grafikpakete keine Spriteanimation bereitstellen. Siehe Tabelle 2.

Geschwindigkeit

Für den Geschwindigkeitsbenchmark wurde ein kleines Programm geschrieben, welches die Anzahl der gezeichneten Pixel pro Sekunde ermittelt. Der verwendete Rechner besitzt einen i286/20MHz Prozessor mit einem *Norton-SI*-Faktor von 21. Beim Gegenvergleich mit VGAGRAPH und mit einem i486/33MHz (*Norton-SI* 51) wurde festgestellt, daß die Geschwindigkeit nicht wesentlich zunahm. Die Begründung liefert *Michael Abrash* in [6], wo er feststellte, daß die Zugriffe auf einen VGA-Port mit einem i486 (das ist im wesentlichen beim hochauflösenden 16 Farbenmodus der Fall) so langsam wie eine Multiplikation sind und in erster Linie von der Geschschwindigkeit des Bussystems abhängen.

Wer im Videomodus $13 noch schneller sein will, der sollte [4,5] lesen.

Die Benchmarks wurden auf den hochauflösenden 16 Farbenmodus beschränkt, da VGAGRAPH nur diesen Modus unterstützt.

VGAGRAPH1 zeigt die Ergebnisse der schnellen Wörter LINE, LINEH und LINEV, wogegen VGAGRAPH2 die Ergebnisse der DRAW-Routinen anzeigt, die noch einen kleinen Overhead haben. SMILEY1 ist die DIRECT_GRAPHICS-Version, SMILEY2 die Version mit BIOS_GRAPHICS. Man erkennt deutlich den Laufzeitnachteil der BIOS-Version. Bei GRAFDISP1 wurde (LINE) und bei GRAFDISP2 LINE verwendet, so daß sich eine kleine Laufzeitverbesserung aufgrund des Fehlens des Abspeicherns der Endkoordinaten ergibt.

❑

Literaturverzeichnis

[1] Jürgen Petsch, "Schnellfärberei", c't 8/89, S.176-184, c't 9/89, S.214-226.

[2] Jürgen Petsch, "Kopierbahnhof VGA-RAM", c't 4/90, S.438-440, c't 5/90, S.12.

[3] Jürgen Petsch, "Turbo-Circle", c't 10/91 S.172-176.

[4] Michael Abrash, "The good, the bad, and the run-sliced", Dr. Dobb's Journal, pp. 171-176, 190-191, November 1992.

[5] Michael Abrash, "Moving, faster lines, and page flipping", Dr. Dobb's Journal, pp. 143-145, 159-160, December 1992.

[6] Michael Abrash, "Yet another animation method", Dr. Dobb's Journal, pp. 127-131, 140-141, January 1993.

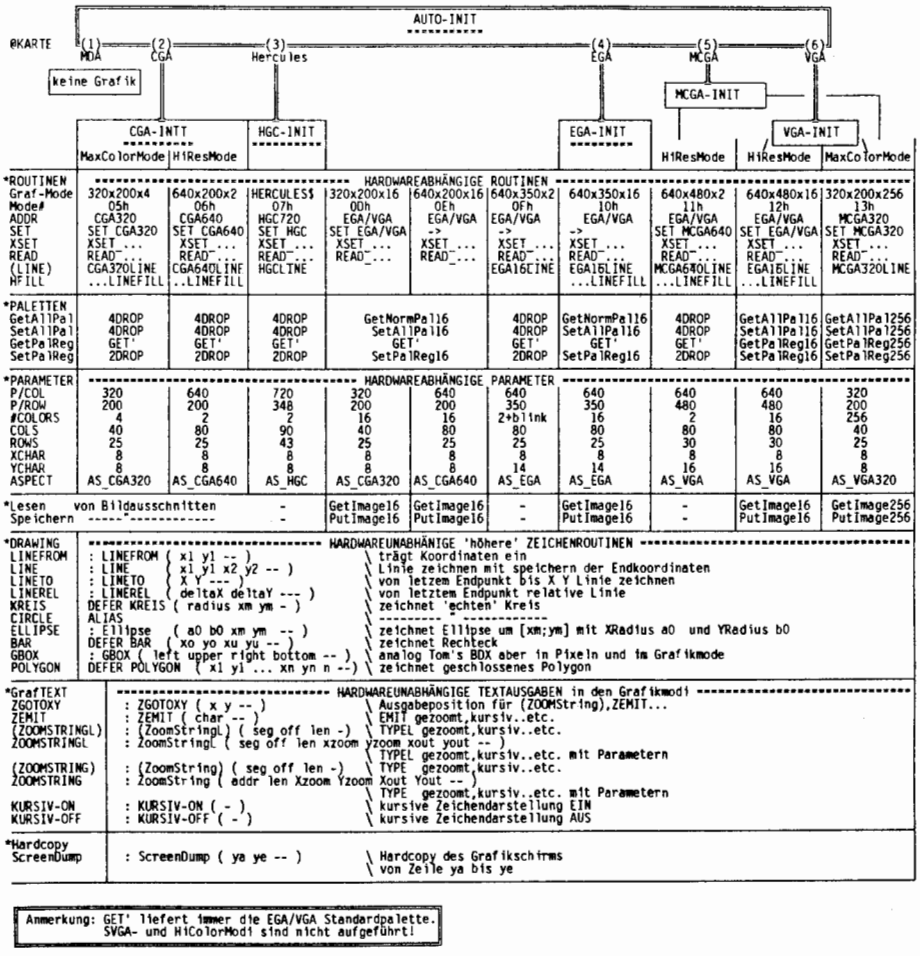
[7] Michael Abrash, "More dirty (dirtier?) rectangles", Dr. Dobb's Journal, pp. 127-129, 142-144, February 1993.

[8] Dirk Hilberg, "Akima-Interpolation", c't 6/89, S.206-214.

[9] Jorke, Lampe, Wengel, "Arithmetische Algorithmen der Mikrorechentechnik", Verlag Technik, S.270-271.

[10] Phil Koopman Jr., FORTH Dimensions, Vol. IX, No.1, pp. 12-16.

Die Struktur des Grafikdispatchers



Anmerkung: GET' liefert immer die EGA/VGA Standardpalette.
SVGA- und HiColorModi sind nicht aufgeführt!

	MOVEMODE			XORMODE		
	Horizontal	Vertikal	Diagonal	Horizontal	Vertikal	Diagonal
VGAGRAPH1	2.7M	371k	230k	2.7M	371k	230k
VGAGRAPH2	1.8M	399k	212k	1.8M	339k	219k
GUNTON	2.9M	358k	155k	182k	358k	155k
SMILEY1	2.4M	79k	79k	2.4M	57k	57k
SMILEY2	21k	21k	21k	23k	22k	22k
GRAFDISP1	3.0M	361k	177k	-	-	-
GRAFDISP2	2.6M	356k	176k	-	-	-

Abb. 3

Mengenrabatt

Variable und Konstante elegant deklariert

von Wolf-Helge Neumann
Huttenstraße 27, 7000 Stuttgart 31, Tel.: 07 11 - 8 87 26 38

Wer seine eigene (unkommentierte) Stackakrobatik schon mal verflucht hat beim Versuch, sie wieder zu ergründen, wird zugeben, wie angenehm ein paar Variablen mit aussagefähigen Namen das Leben machen können. Je zwei Wörter für Variablen und Konstanten ermöglichen einen neuen, übersichtlichen Deklarationsteil.

Bisher war ein Deklarationsteil in FORTH ziemlich lästig beim Schreiben. Nach dem dritten VARIABLE wollen die Finger Abwechslung und schreiben dann viel lieber "VARIALBE" oder sonstige Variationen.

Ein "schöner" Deklarationsteil sieht deshalb für mich wie Bild 1 aus:

Im Grunde also ähnlich wie in Pascal, nur mit dem Unterschied, daß eben der Typ am Anfang steht und bei Variablen auch gleichzeitig eine Initialisierung möglich ist.

Betrachten wir jetzt mal nur die Variablen und tasten uns an unser Wunschziel heran. Außerdem vereinfachen wir die Sache zusätzlich, indem wir von einer Syntax ausgehen, die so lauten soll:

```
VAR var1 [Variablenliste];
```

Dann muß VAR bei der Compilation nichts anderes tun, als solange VARIABLE aufzurufen, bis der Zeiger in den Quelltext (>IN) den Strichpunkt erreicht hat. Dazu müssen wir feststellen, wie der Inhalt von >IN beim Strichpunkt ist. Das erledigt FIND_; (scr 4) für uns. (Außerdem bricht es auch noch mit einer Fehlermeldung ab, wenn kein Strichpunkt mehr im Screen zu finden ist). Den Rückgabewert n bewahren wir auf dem Returnstack auf, weil wir am Ende >IN auf n+1 setzen müssen, damit ganz normal weiter compiliert werden kann. Zwischen dem Strichpunkt und dem letzten Variablennamen können jetzt auch noch eine ganze Menge Leerzeichen stehen, die wir deshalb mit -TRAILING abschneiden. Wir erhalten dadurch den Wert n'. Wenn der Zeiger >IN den neuen Wert n' überschreitet, sind wir also am Ende der Variablenliste angekommen.

Mit diesen Kenntnissen haben wir unseren Befehl VAR , der eine einfache Variablenliste compiliert (Bild 2).

Was jetzt noch fehlt, ist die Initialisierung der Variablen, wobei es freigestellt sein soll, Variablen zu initialisieren oder nicht. Wird kein Initialisierungswert angegeben, bleibt es VARIABLE überlassen, ob angelegte Variablen mit 0 initialisiert werden.

Beim Compilieren muß jetzt also nicht nur solange VARIABLE aufgerufen werden, bis der Zeiger >IN die gemerkte Position nach dem letzten Variablennamen überschreitet, sondern jetzt muß nach jedem Aufruf von VARIABLE im Quelltext vorausgeschaut werden, ob eine Zuweisung für die gerade definierte Variable vorgesehen ist, oder ob eine neue Variable definiert werden soll. Diese Arbeit übernimmt der Befehl LOOK_AHEAD (scr 3). LOOK_AHEAD benötigt dazu auch noch einen Gehilfen, der die Zuweisung ausführt. Dieser Gehilfe nennt sich := (scr 2). Fangen wir bei der Definition von := an:

Als erstes wird versucht, das nach := stehende Wort als Zahl zu interpretieren. Schlägt dies fehl, haben wir := verschaukelt, weil wir den Zuweisungsoperator hingeschrieben haben, aber keine Zahl, die zugewiesen werden könnte. Der Gehilfe := wird sauer und beschwert sich natürlich.

Findet := jedoch eine Zahl, so liegt auf dem Stack die Zahl und darüber ein Wert, der Auskunft gibt, ob es eine einfachgenaue oder eine doppeltgenaue Zahl ist. Dieser Wert wird auf dem Returnstack aufgehoben, weil er später noch gebraucht wird. Dann wird die PFA des zuletzt definierten Wortes bestimmt. Mit LAST @ haben

```
VAR     v1 := 10          \ v1: ...
        v2                \ v2: kein Preset
        v3 := 30          \ ...
        v4          ;     \ ...
2VAR    v5
        v6 := 60.         \ ...
        v7          ;
CONST   c1 := 100         \ ...
        c2 := 200   ;
2CONST  c3 := 300.        \ ...
        c4 := 400.  ;
```

Abb. 1

```
: Var ( — ) find_; dup >r
  -trailing (>in nach der letzten Variable) >r drop
  BEGIN Variable >in @ r@ > UNTIL rdrop
  r> 1+ >in ! (>in auf Zeichen nach dem ; setzen)
  ; immediate
```

Abb. 2

Stichworte

big-Forth
Atari
32-bit
Variable
Constant
Deklaration

wir die NFA dieses Wortes, NAME> wandelt diese in die CFA und >BODY wandelt die CFA in die PFA. Jetzt wird anhand des vorher auf den Returnstack geretteten Wertes entschieden, ob ein einfachgenauer Wert oder ein doppeltgenauer Wert eingetragen werden muß.

Nach der Zuweisung kann jetzt bereits wieder eine neue Variablendeklaration kommen, es könnte aber auch noch Kommentar im Quelltext stehen. Deshalb schaut LOOK_AHEAD solange weiter voraus, bis ein nicht-ausführbares Wort erscheint, das dann der Name für die nächste Variable sein muß. Nicht-ausführbar ist in diesem Zusammenhang alles außer dem \ für Kommentare und dem Zuweisungsoperator :=. Ob ein Wort ausführbar ist, entscheidet EXECUTABLE?, das `true` als Zeichen für Ausführbarkeit auf die fragliche CFA legt. In EXECUTABLE? könnte ggf. auch noch die öffnende Klammer eingetragen werden. Hat LOOK_AHEAD ein nicht-ausführbares Wort gefunden, wird der Zeiger >IN wieder auf dieses Wort gesetzt. Beim nächsten Schleifendurchlauf in VAR wird auf dieses Wort dann erneut VARIABLE losgelassen. Und das alles eben solange, bis wir am Ende der Variablenliste angekommen sind. Die neue, endgültige Version von VAR steht in Screen 5.

Entsprechend können wir jetzt 2VAR definieren und sogar CONST und 2CONST. Nur muß man dabei CONSTANT bzw. 2CONSTANT immer einen Dummy-Wert zum Fraß vorwerfen. In diesem Fall ist das die Null. Der Zuweisungsoperator überschreibt danach die Null mit dem richtigen Wert. Allerdings führt dies dazu, daß es gar nicht auffällt, wenn einer Konstanten kein Wert zugewiesen wird. Die betreffende Konstante hat dann eben den Wert Null. Man könnte natürlich noch eine Sicherung bei den Schlüsselwörtern einbauen, vielleicht in Form eines Flags, das angibt, ob eine Zuweisung optional ist oder nicht, aber das war mir etwas zu übertrieben. Eine Konstante mit dem Wert 0 macht sich beim Ablauf des Programmes bestimmt auch so bemerkbar.

Eine andere Fehlerquelle besteht darin, wenn einer Variablen ein Wert zugewiesen wird, dessen Format nicht mit dem Variablentyp übereinstimmt. Auch das ließe sich entschärfen, wenn := von VAR bzw. 2VAR ein Flag übergeben bekäme, das darüber Aufschluß gibt, ob ein einfach- oder doppeltgenauer Wert eingetragen werden muß, damit := ggf. eine Fehlermeldung bringen kann. Eine Alternative wäre auch, := deferred zu machen und VAR bzw. 2VAR würden die erforderliche Variante von := eintragen.

Programmiert wurden diese Wörter mit BigForth v1.0 auf dem Atari ST. Es funktioniert aber genauso mit dem Volksforth und dürfte also auch auf dem PC laufen.

❑

Listing zu: Mengenrabatt (Wolf-Helge Neumann

Screen #1

```
\ So kann ein Deklarationsteil jetzt aussehen:
VAR     v1 :=  10               \ v1: ...
        v2                      \ v2: kein Preset
        v3 :=  30               \ ...
        v4          ;           \ ...
2VAR    v5
        v6 :=  60.              \ ...
        v7          ;
CONST   c1 := 100               \ ...
        c2 := 200  ;
2CONST  c3 := 300.              \ ...
        c4 := 400. ;
```

Screen #2

```
\ Zuweisungsoperator := , Liste ausführbarer Wörter
: :=    ( -- )    bl word number?  dup >r 0=
  \ Trägt in die PFA des zuletzt def. Wortes einen Wert ein.
  IF abort" no number" THEN
  last @ name> >body    r> 0>  IF 2! ELSE ! THEN ;
| : executable?    ( cfa -- cfa f )    dup
  \ Hier stehen die Wörter, die nicht als Variable aufgefasst
  \ werden können, sondern ausgeführt werden müssen.
  ['] := case? IF true exit THEN
  ['] \  case? IF true exit THEN
  drop false ;
```

Screen #3

```
\ look_ahead
| : look_ahead    ( -- )
  \ Alles ausführbare nach einer Variablendefinition wird aus-
  \ geführt, bis die nächste Definition kommt.
  \ Was ausführbar ist, bestimmt executable?.
  BEGIN
    >in @    bl word find
    IF ( es hat eine CFA ) executable?
         IF     execute drop ( >in ) false
         ELSE   drop >in ! ( >in restaurieren ) true THEN
    ELSE ( es hat keine CFA ) drop ( String ) >in ! true THEN
  UNTIL ;
```

Screen #4

```
\ nach Ende-Markierung ; suchen

| : find_;    ( -- addr n )
  \ An addr steht das Endezeichen ;. Im Quelltext stehen n
  \ Zeichen vor dem Strichpunkt.
  source    >in @ - swap >in @ + swap    (Startadresse u. count )
  ascii ; scan
  0= abort" ; missing"    ( Abbruch, wenn ; am Ende fehlt! )
  source drop under -   ( >in beim ; ) ;
```

Screen #5

```
\ Variablen

: Var ( -- )    find_; dup >r
  -trailing ( >in nach der letzten Variable ) >r drop
  BEGIN    Variable look_ahead    >in @   r@ > UNTIL rdrop
  r> 1+ >in !    ( >in auf Zeichen nach dem ; setzen )
  ; immediate
: 2Var ( -- )    find_; dup >r
  -trailing ( >in nach der letzten Variable ) >r drop
  BEGIN    2Variable look_ahead    >in @   r@ > UNTIL rdrop
  r> 1+ >in !    ( >in auf Zeichen nach dem ; setzen )
  ; immediate
```

Screen #6

```
\ Konstanten

: Const ( -- )    find_; dup >r
  -trailing ( >in nach der letzten Variable ) >r drop
  BEGIN    0 Constant look_ahead    >in @   r@ > UNTIL rdrop
  r> 1+ >in !    ( >in auf Zeichen nach dem ; setzen )
  ; immediate
: 2Const ( -- )    find_; dup >r
  -trailing ( >in nach der letzten Variable ) >r drop
  BEGIN    0. 2Constant look_ahead    >in @   r@ > UNTIL rdrop
  r> 1+ >in !    ( >in auf Zeichen nach dem ; setzen )
  ; immediate
```

❑

INFOSYS-106

von Burkhard Golf, Rolf Schönlau,
Franz Angenendt, Klaus W. Pleßmann

pdw, RWTH Aachen, Beverstr. 46, Tel. 0241-50 07 49

Im ersten Teil dieser Veröffentlichung wurde mit Forth++ eine Objekt-orientierte Spracherweiterung für Forth vorgestellt. Diese Spracherweiterung wurde im Rahmen des Sonderforschungsbereichs 106 "Korrelation von Fertigung und Bauteileigenschaften bei Kunststoffen" [Pleß92] entwickelt und wird als Programmiersprache für das Informationssystem INFOSYS-106 eingesetzt.

Der zweite und abschließende Teil der Veröffentlichung soll dem Leser eine Einführung in den strukturellen Aufbau des INFOSYS-106 geben, insbesondere in dessen Objekt-orientierten Sprachinterpreter. Zunächst wird der Anforderungskatalog an das Informationssystem vorgestellt, der letztlich zu der Entwicklung des hier vorgestellten INFOSYS-106 führte. Nach einer anschließenden Motivation für die Benutzung von Objekt-orientierten Programmiermethoden folgt eine grobe strukturelle Beschreibung des INFOSYS-106. Am Beispiel der Mechanismen, die die Kommunikation zwischen Benutzer und Informationssystem bereitstellen, wird abschließend noch auf die Programmierung des Objekt-orientierten Forth-Sprachinterpreters eingegangen. Eine ausfürlichere Darstellung ist im Rahmen einer Veröffentlichung leider nicht möglich, aber für detailliertere Informationen möchten wir den interessierten Leser auf [Ange92] verweisen.

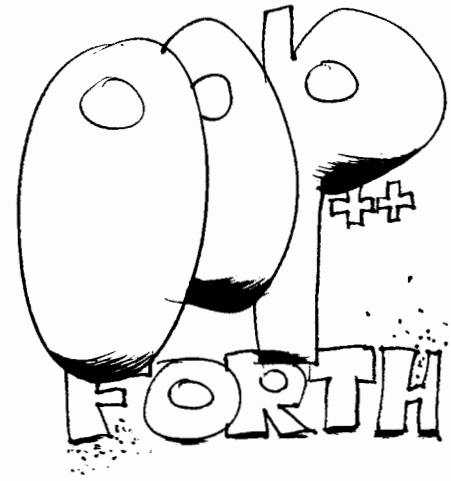
1. Ein Informationssystem für den Sonderforschungsbereich 106

Schon bald stellten sich die im Sonderforschungsbereich 106 verwendeten Methoden für die Aufbereitung der erzielten Meßergebnisse als nicht ausreichend heraus. Bislang wurden die Ergebnisse nach der Erfassung in einer zentralen Datenbank mit Hilfe eines speziellen Korrelationsprogramms ausgewertet und die gefundenen Korrelationen gespeichert. Sonstige darüber hinausgehende Informationen, die einen nicht unwesentlichen Bestandteil der Forschung darstellen, konnten nicht zentral gespeichert werden.

Diese zusätzlichen Notizen der Projektbearbeiter reichen über strukturierende Aussagen bis zur Dokumentation und Begründung von Ergebnissen und knüpfen praktische Tips, Erfahrungen und Einschätzungen als inhaltliche Ergänzung an die einzelnen Ergebnisse. Zusätzlich zu den bisher verwendeten Werkzeugen mußten also neue für die Erfassung und Strukturierung dieser Informationen gefunden werden. Damit ein Projektbearbeiter die Begründung von Aussagen im zugehörigen Datenmaterial sofort eintragen (bzw. auch finden) kann, sollten die zusätzlichen Informationen direkt an die Ergebnisse in der Datenbank geknüpft werden.

Aus der Aufgabe der Erfassung und Aufbereitung dieser verschiedenartigsten Informationstypen ließ sich zunächst die Frage nach einer geeigneten Strategie ableiten, wie die verschiedenen Informationstypen zu erfassen und anzuordnen sind. Die Anordnung der Informationen mußte darüber hinaus jederzeit für Erweiterungen oder Umstrukturierungen änderbar bleiben, um ein inkrementelles Wachstum des Informationsgehaltes im System zu ermöglichen.

Aus diesen Gründen konnte nur eine nicht lineare Anordnungsform der Informationen in Frage kommen. Die zusätzlichen Informationen werden zunächst als Textstücke aufgefaßt und dann über Referenzen an direkte Ergebnisse oder andere Informationen geknüpft. Für den Benutzer besteht die Menge an abrufbaren bzw. editierbaren Informationen, die im folgenden auch als Informationsbasis bezeichnet wird, also aus Textstücken



und dazugehörigen Referenzen, die in die Datenbank oder auf andere Textstücke verweisen. Hinzu kommen systemintern noch für den Rechner verwertbare Informationen. Im einfachsten Fall sind dies Autorkennungen, Sicherheits- oder Zuverlässigkeitsangaben oder eine erweiterte Liste von

Stichworte

OOP
Informationssystem
Hyper-Text

Die Autoren:
Dipl.-Inform. Burkhard Golf,
Dipl.-Inform. Rolf Schönlau,
Dr. rer. nat. Franz Angenendt ,
Prof. Dr.-Ing. Klaus W. Pleßmann;
Verfahren der Prozeßdatenverarbeitung und Prozeßführung, RWTH Aachen.
Die Autoren waren von 1990 bis 1992 gemeinsam am Lehr- und Forschungsgebiet für Verfahren der Prozeßdatenverarbeitung und Prozeßführung (pdv) der RWTH Aachen tätig. Das pdv stellte zu dieser Zeit das Teilprojekt "Datenhaltung, Datenkorrelation und Datenauswertung" im Sonderforschungsbereich 106.

Referenzen. In komplexeren Fällen können auch prozedurähnliche Codesequenzen enthalten sein, die in bestimmten Situationen ausgeführt werden. Die Gesamtheit von Text, Referenzen und Codesequenzen wird unter einer Informationseinheit zusammengefaßt.

Die Zuordnung von Codesequenzen zu den Informationseinheiten schafft die Möglichkeit, bei Aktivierung einer Informationseinheit hierzu eingetragene Codesequenzen auszuführen, um bestimmte Einstellungen durchführen zu können. Beispielsweise kann die Datenbank, die standardmäßig mit voreingestellten Parametern aufgerufen wird, durch die Wahl ausgezeichneter Informationseinheiten mit einer veränderten Parameterbelegung aktiviert werden. Gleiches gilt im Fall des Korrelationsprogramms. Die erfaßten Informationseinheiten können demnach einen passiven oder aktiven Charakter besitzen, der durch die Existenz von Codesequenzen bestimmt wird.

Der Aufbau der Informationsbasis soll direkt durch die jeweiligen Projektbearbeiter möglich sein, um Informationsverluste klassischer Akquisitionsmethoden zu vermeiden. Die Ergebnisse werden als Texte erfaßt, da sie normalerweise auch in sprachlicher Form dargestellt werden. Während der Erfassung werden die strukturellen Informationen (Referenzen, Codesequenzen, etc.) von natürlichsprachigen Sequenzen getrennt.

Ein direkter Aufbau der Informationsbasis durch den Experten erfordert ein hohes Maß an Interaktivität sowie eine gezielte Unterstützung vom Informationssystem. Diese Unterstützung wird in der Oberfläche des INFOSYS-106 durch die Integration von Informationseinheiten und Referenzen in ein Hypertextsystem bereitgestellt. Die Technik Textbereiche mit integrierten Referenzen zu strukturieren, die vom Rechner verarbeitet werden können, kann gut für die Ergebniserfassung und Textverknüpfung im Informationssystem eingesetzt werden. Interessanterweise wurde diese in jeder Enzyklopädie verwendete Struktur schon 1945 vorgeschlagen, um ein weltumfassendes Informationssystem MEMEX zu entwickeln [Bush45].

Wie in Bild 1 ersichtlich, stellen sich die Informationseinheiten dem Benutzer in Form von Karten dar, die er auswählen und ansehen kann. Der Inhalt einer Karte wird im zentralen Fenster angezeigt. Im Bild wird gerade die Eröffnungskarte "Start" angezeigt, die eine Dialogbox öffnet und den Benutzernamen abfragt. Anschließend bietet diese Karte dem Benutzer eine Themenauswahl an, mit der er das Thema auswählt, mit dem er die Sitzung beginnen möchte. Mit dem Systemkommando Thema kann diese Themenauswahl jederzeit wiederholt werden.

Die Eröffnungskarte besitzt die vier Referenzen "Systemkommandos", "Textreferenzen", "Info-Stapel" und "Eine Einführung". Diese Referenzen können vom Benutzer aktiviert werden und liefern weitere Informationen. Auf dem Info-Stapel sieht der Benutzer den Stapel der bislang ausgewählten Karten, im Moment also nur eine Karte.

Die Informationseinheiten definieren somit nicht weiter teilbare Einheiten in einer Hypertextbasis und bilden die Knoten der Hypertextstruktur, die durch Referenzen miteinander verknüpft sind. Die Referenzen sind die Kanten zwischen den Knoten und erzeugen die nicht lineare Struktur eines Hypertextes. Informationselemente enthalten also durch Referenzen Be-

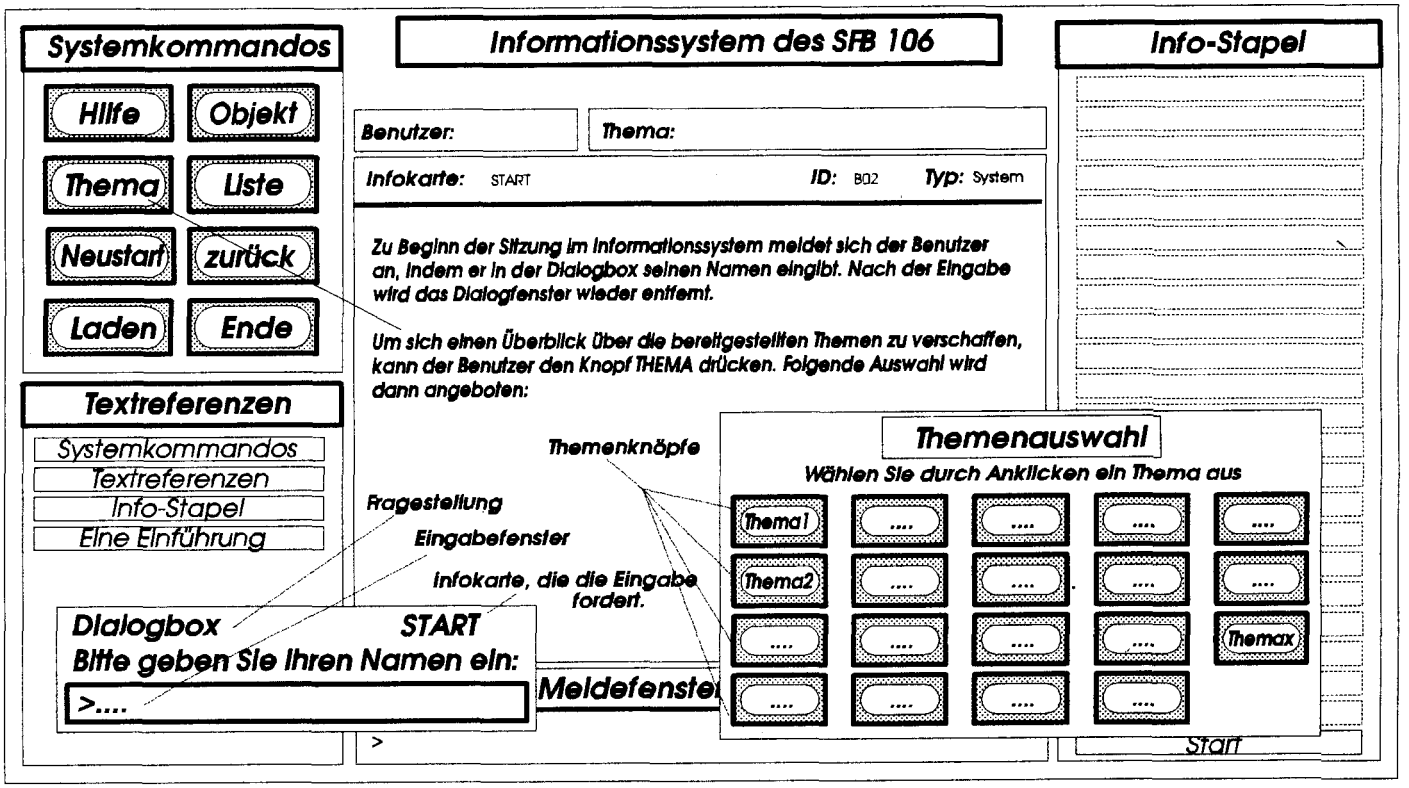


Abb. 1: Oberfläche des INFOSYS-106

züge zu anderen Informationselementen, die wie in einem Hypertextsystem durch Interaktion des Benutzers aktiviert werden können. Auf diese Weise erhält jeder Benutzer auch einen leichten Zugang zu Eintragungen anderer Benutzer und kann Kommentare zu beliebigen Informationseinheiten einfügen.

Da die Informationsbasis schrittweise durch die Benutzer bzw. Experten aufgebaut werden sollte, konnte nur ein Interpreter im Kern des Informationssystems die geforderte Interaktivität bereitstellen. Ein Interpreter kann Änderungen zur Laufzeit innerhalb eines Informationselements berücksichtigen und den gefundenen Code eines Informationselements direkt ausgeführen. Diese Suche nach einem Interpreter führte dann zum Einsatz von Forth, wobei für die Realisierung der Informationsbasis im Institut ein Forth83-System zur Verfügung stand.

2. Objekt-orientierte Strukturen des Informationssystems

Die einzelnen Informationseinheiten lassen sich als Objekte im Sinne der Objekt-orientierten Programmierung vorteilhaft realisieren und verwalten. Denn mit Hilfe der Objekt-orientierten Programmiertechnik läßt sich auf einfache Weise eine Kommunikationsschnittstelle für alle Informationseinheiten festlegen, die die typkorrekte Nutzung von Daten wie Adressen, Aufzählungstypen, reellen Zahlen oder Objekten kontrolliert. Mit dem Typkonzept von Forth++ existiert in Form der Zusicherungen eine Unterstützung dieser Schnittstelle direkt auf dem Datenstack des Laufzeitsystems. Zudem stellt die Vererbung - in Verbindung mit der Möglichkeit Methoden deferred zu deklarieren - ein komfortables Instrument dar (siehe Kapitel 5).

Die Informationseinheiten werden als lose gekoppelte Objekte realisiert, so daß dynamisch weitere Informationseinheiten ergänzt werden können. Durch die Interaktivität des Interpreters wird diese lose Kopplung der Objekte auch zur Laufzeit unterstützt. Alle Interaktionen mit Objekten, bei denen der Benutzer beteiligt ist, laufen über die interaktive Schnittstelle als Nachrichten an Objekte ab. Dies ist zum Beispiel der Fall, wenn der Benutzer des Informationssystems eine Referenz auswählt und damit dem referenzierten Objekt die Nachricht zugesendet wird seinen Inhalt darzustellen. Für Objekte ist daher die Kommunikation mit dem Benutzer identisch mit der Kommunikation zu anderen Objekten.

Das Informationssystem stellt die für die Erzeugung von neuen Informationseinheiten notwendigen Klassen und Verwaltungsobjekte bereit, wenn ein Benutzer die Informationsbasis ergänzen möchte. Dann werden automatisch leere Objekte vom Typ Informationseinheit erzeugt, die der Benutzer mit individuellen Daten besetzen kann. Diese Objekte besitzen durch die Vererbung der Methoden bereits alle Funktionen zum Einsatz in der Informationsbasis.

Weil sich die Informationsbasis aufgrund des inkrementellen Aufbaus permanent im Wachstum befinden wird, können Referenzen auf noch nicht angelegte Informationselemente nicht ausgeschlossen werden. Die so ins Leere weisende Referenzen führen bei der Aktivierung nicht zu Fehlfunktionen, da der gewählte Interpreter und die Versendung von Nachrichten in Objekt-orientierten Sprachen als Kommunikation zwischen Objekten eine einfache und effiziente Möglichkeit bereitstellen, diese Fehlaufrufe abzufangen und zu verwalten. Hier wird - in Analogie zum dynamischen Binden - bei der Nachrichtenübergabe zwischen Objekten die Existenz des Wortes geprüft und anschließend die Aktivierung durchgeführt oder im Fehlerfall eine Meldung an den Benutzer ausgegeben.

Um nicht auf die Mittel der Objekt-orientierten Programmierung verzichten zu müssen, wurde das im ersten Teil dieser Veröffentlichung vorgestellte Forth++ entwickelt. Der Forth-Interpreter hat durch seine interne Struktur und Kompaktheit gute Laufzeiteigenschaften und stellt durch die Erweiterung eine objektorientierte Programmiersprache bereit, mit der die Programmierung des INFOSYS-106 Objekt-orientiert erfolgen konnte.

Die im Informationssystem genutzten Vorteile betreffen vor allem die Interaktivität. Zudem wird die Struktur des Wörterbuchsystems herangezogen, um eine Klassenhierarchie zu realisieren und die Methoden nur lokal in einer Klasse bekannt zu machen. Über die Suchstrategie des Forth-Laufzeitsystems ist auch die Vererbung von Methoden an Unterklassen realisiert worden.

3. Konzeptionelle Struktur des Informationssystems

Entsprechend den aus Hypertextsystemen abgeleiteten Möglichkeiten, kann in einer Konsultation die Informationsbasis durch sukzessive Auswahl von Referenzen durchwandert werden. An jeder Position im Informationssystem können zusätzlich Texteingaben in Form von neuen Informationseinheiten erstellt und durch Referenzen mit anderen Informationseinheiten verknüpft werden.

Die Informationseinheiten sind zunächst passive Elemente, die ihren Inhalt nach Aktivierung darstellen und Referenzen für eine Weiterführung des Dialoges anbieten. Informationseinheiten können aber auch eine aktive Rolle spielen, um Eingaben vom Benutzer zu erfragen und in ihre inhaltliche Darstellung zu integrieren. Der Typ einer Informationseinheit legt ihre Eigenschaften fest, wie die Informationseinheit sich verhält und welche Aktivitäten von der Informationseinheit vor, während oder nach ihrer Benutzung ausgeführt werden.

Jede Informationseinheit besitzt einen Namen und trägt diesen bei jeder Aktivierung in die Dialoghistorie ein. Hiermit kann der Benutzer seinen zurückgelegten Weg durch die Informationsbasis beobachten und gegebenenfalls zurückgehen.

Wird ein Informationsmodul oder eine gesamte Informationsbasis geladen, so lassen sich alle geladenen Informationseinheiten in einem Systemregister registrieren. Ausgezeichnete Informationseinheiten, die Einstiegpunkte in Einzelthemen des Problemraumes sind, tragen sich zusätzlich in das Themenregister ein, um dem Benutzer einen thematischen Zugang in den Problemraum zu ermöglichen.

Die für das Informationssystem definierten Referenzen sind mit Nachrichten zwischen Objekten gleichzusetzen. Die Typisierung besteht daher im Aufruf verschiedener Methoden der Klassendefinitionen. Die Referenzen werden dem Benutzer in Form von aktivierbaren Knöpfen angeboten und besitzen eine Beschriftung, die dem Benutzer die durch die Referenz ausgelöste Aktion vermittelt.

4. Die Systemebene des INFOSYS-106

Die Konzeption des Informationssystems stellt in der Systemebene umfassende Funktionen bereit. Über ein Kommando kann mit Hilfe der Systemebene ein Prozeß auf einem im Netz befindlichen Rechner aktiviert und ein Datenaustausch ermöglicht werden. Die Systemebene umfaßt die folgenden Komponenten:

- Dialogkontrolle *(FDA, Flexibler Dialog-Assistent)*
- Grafikkomponente *(MiMo, Mini Motif)*
- Kommunikationsmechanismus *(IPC, Inter Process Communication)*
- Kontrollprozeß *(Monitor, Überwachung aller Prozesse)*
- interner Koppelprozeß *(Prozeßadapter)*
- externer Koppelprozeß *(Netzadapter)*
- Sprachinterpreter *(Forth++, objektorientierter Forth-Sprachinterpreter)*

Die Komponenten der Systemebene operieren als parallel existierende Prozesse und nutzen den Kommunikationsmechanismus zum Austausch von Informationen und Aufträgen sowie zur Abstimmung ihrer Aktivitäten. Innerhalb der Komponenten ist der Flexible Dialog-Assistent FDA die zentrale Schaltstelle für die Koordinierung des Systems. Zur Systemüberwachung wird vom Monitor jeder gemeldete Prozeß zyklisch kontrolliert. Über den FDA werden alle Datenströme koordiniert. Die Grafikkomponente MiMo hat allein Kenntnis über das verwendete Grafikpaket.

Mit diesen konzeptionellen Entscheidungen ist die Systemebene eine offene, modulare Struktur mit eindeutiger Verteilung der Aufgaben. Der Interpreter mit der Objekt-orientierten Spracherweiterung Forth++ gehört zum Umfang der Systemebene und kann aus Sicht des Flexiblen Dialog-Assistenten als normale Komponente aufgefaßt werden, die über den Kommunikationsmechanismus Ein-/Ausgabeaufträge ausgibt.

5. Die Kommunikation der Informationsbasis

Als Abschluß soll ein einfaches Beispiel für die Programmierung des Interpreters dienen. Alle Objekte der Informationsbasis nutzen einen Dialogmanager, der die einheitliche Schnittstelle für die Kommunikation nach außen darstellt. Die Realisierung der Eigenschaften des Dialogmanagers in den Methoden der Klassendefinition zeigt die einfach gehaltene Struktur der Kommunikation. Mit der Methode `ioreques` kann eine Informationseinheit einen Dialogwunsch anmelden. Die Daten werden mit den Methoden `get_number` bzw. `get_string` an das beauftragende Objekt zurückgesendet. Die Methode `release` führt zur Freigabe des Dialogmanagers.

Als Objektdaten besitzt ein Dialogmanager eine Flagge zur Anzeige eines aktiven Dialoges und Variablen zur Speicherung des zu übermittelnden Datentyps sowie der Objektadresse des Empfängers, die bei der Anmeldung eines Dialogwunsches übergeben werden. Die nach außen hin sichtbaren Methoden der Dialogmanagerklasse sind in Bild 2 in Forth++-Notation zusammengestellt.

Mit diesen vier Methoden ist die Kommunikation mit der Benutzeroberfläche für alle Objekte der Informationsbasis festgelegt. Im Dialogmanager müssen die Methoden `iorequest` und `release` realisiert sein, während die beauftragenden Objekte die Übernahmemethoden `get_number` bzw. `get_string` bereitstellen müssen. Um durch Vererbung die Methoden in ihrer Aufrufstruktur für alle Unterklassen bekannt zu machen, ihre Realisierung jedoch den Unterklassen zu überlassen, werden sie in der Vaterklasse deferred deklariert (Bild 2). In allen weiteren Unterklassen dieser Basisklasse stehen diese Methodenköpfe bereit, um im Falle der Datenübernahmemethoden immer wieder individuell auspro-

Sichtbare Methoden der Dialogmanagerklasse

```
Methode iorequest /* zur Anmeldung eines Dialogwunsches */
      : iorequest assert (int16:datatype glob_object_prt:object_addr --> boolean)
      /* mit   datatype      : 1=Zahlen; 2=Zeichenfolgen
               object_addr    : Adresse des Empfängerobjektes
               Mit der logischen Flagge wird dem beauftragenden Objekt mitgeteilt, ob der
               Dialogwunsch angenommen wurde (true) oder der Dialogmanager zur Zeit keine
               weiteren Dialogwünsche annimmt (false).
      */
Methode release     /* zur Abmeldung des Dialogwunsches */
      : release assert (-->)
Methode get_number  /* zur Übersendung von Zahleneingaben an ein Objekt */
      : get_number assert (int16 -->)
Methode get_string  /* zur Übersendung von Zeicheneingaben an ein Objekt */
      : get_string assert (string -->)
```

Abb. 2

grammiert werden zu können.

Mit den hier eingesetzten Mitteln objektorientierter Programmierung, die in der Spracherweiterung Forth++ zur Verfügung steht, kann für jede Objektklasse eine eigene, den Erfordernissen angepaßte Übernahmemethode programmiert werden, ohne den diese Methoden nutzenden Dialogmanager von der Redefinition in Kenntnis setzen zu müssen. Bild 3 stellt für eine Objektunterklasse den Zusammenhang zwischen Vaterklasse, deferred deklarierten Methoden und Realisierung der Methoden dar.

Die beiden Methoden `get_number` und `get_string` müssen nicht beide für den Dialogwunsch notwendig sein. Erfragt eine Informationseinheit ein Datum aus der Datenbank, z. B. die Temperatur eines Verarbeitungsschritts, so kann die nicht benötigte Methode `get_string` als Methode ohne Codekörper angelegt werden. Die Bestimmung einer deferred geerbten Methode ist aus formalen Gründen zwingend, da nur dann Objekte dieser Klasse realisiert werden können, wenn alle benannten Methoden auch ausprogrammiert sind. Nicht benötigte, deferred deklarierte Methoden müssen demzufolge zumindest mit einer leeren Methodendefinition versehen werden.

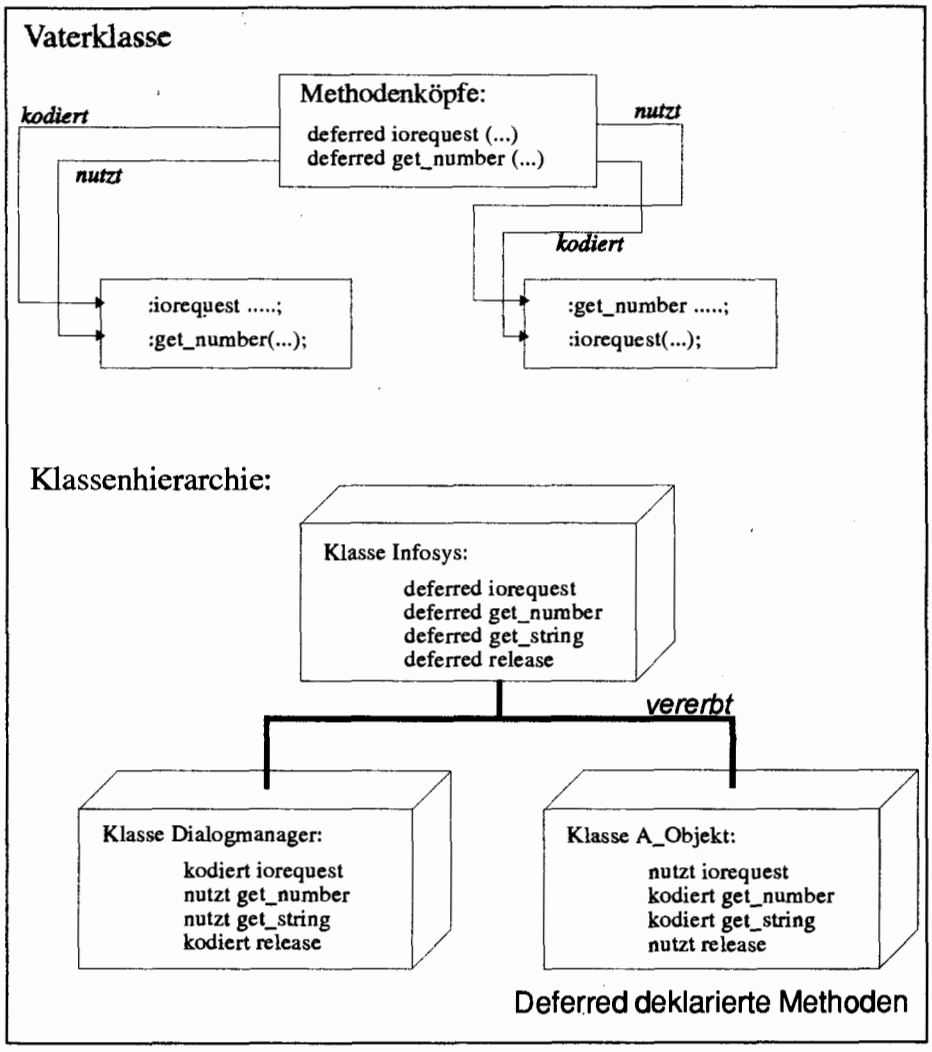
❑

Literatur:

[Ange92]
Angenendt, F.; Ein Beitrag zur Konzeption integrativer Informationssysteme, Dissertation an der RWTH Aachen, 1992

[Bush45]
Bush, V.; As We May Think, Atlantik Monthly 176, Juli 1945,
(Pruduktion in: Software Engeneering Environments, Hrsg. K.H. Bennet, Chichester 1989)

[Pleß92]
Pleßmann, K. W. (Hrsg); Abschlußbericht 1991, Sonderforschungsbereich 106, RWTH Aachen



Deferred deklarierte Methoden

Fragebogenaktion rk

1. Auswertung der VD-Fragebogenaktion (aus VD 4/92)

42 Fragebögen trafen bis zum 20. Januar in der Redaktion ein.
50 waren es schließlich am 30. März.

21 Jahre ist der jüngste,
72 der älteste Einsender.

36 Jahre ist das Durchschnittsalter aller Einsender.

13 gaben als Beruf "Ingenieur" (mit oder ohne Dipl.) an.
Es folgten
8 Studenten
4 Lehrer
2 Rentner.
Die anderen Berufe waren nicht eindeutig einzuordnen.

26 nutzen den Computer professionell.
7 verdienen Teile ihrer Brötchen mit Forth.

37 Einsender arbeiten
mit IBM-workalikes,
11 mit Atari,
7 nutzen Mikro-Controller,
2 arbeiten mit MAC's.
Der Rest machte keine Angaben oder die Hardware wurde nur einmal genannt.

30 programmieren hauptsächlich in Forth,
18 in C,
13 in Assembler,
10 in Pascal,
5 in BASIC.

11 benutzen F-PC, wenn sie fortheln,
7 ZF
7 F83
2 Volks-Forth
3 UR-Forth
3 F68K
2 LMI.
Der Rest nutzt andere Forth-Systeme.

36 Einsender wollen in der VD mehr über Mikro-Controller lesen,
36 sind an Forth als Lehr- und Lerninstrument interessiert,
38 möchten gerne Grundlagenthemen auch mehrmals behandelt sehen,
28 möchten auch forth-fremden Themen Raum einräumen,
27 möchten über preiswerte Hardware-Erweiterungen und deren Programmierung in der VD lesen.

Die Auswertung der "Smily-Fragen" zeigt deutlich, daß die Mehrzahl der VD-Leser mit dem Layout und dem Heft allgemein recht zufrieden sind. Im nächsten Heft folgt die ausführliche Auswertung dazu.

Die Schnecke

oder:
...neulich (vor 2 Jahren) bei Rolf...

von Michael Prümm
Maastrichter Str. 65, 52074 Aachen, Tel.: 0241-8 16 67

Den Anfang einer hoffentlich nicht endenden Reihe von Programmierbeispielen im DisCo-Lampenfeld macht Michael mit einer elegant programmierten Schnecke. Das DisCo-Lampenfeld wurde speziell als Spielwiese und Lehrgegenstand für den Programmiersprachenunterricht entwickelt und in den zwei vorigen VD-Ausgaben vorgestellt.

Vor ca. zwei Jahren zeigte mir *Rolf Kretzschmar* sein Leuchtdiodenfeld. Zu dieser Zeit war er immer auf der Suche nach ansprechenden Beispielen für seinen Programmierkurs. Spontan kam mir die Idee zu einer Spirale, die sich von außen nach innen um das Feld windet. Ebenso spontan meinte Rolf, das sei viel zu aufwendig zu programmieren. Das war eine Herausforderung!

Was braucht man nun, um die Spirale zum Vorschein zu bringen? Eine nähere Betrachtung des Problems zeigt, daß man zunächst einmal (1) eine Strecke am Rand entlang gehen und dabei hinter sich einzelne LEDs einschalten muß, dann (2) im rechten Winkel abbiegt und wieder mit Schritt 1 anfängt, bis man (3) in der Mitte angekommen ist. Dabei wird der Weg, den man geradeaus zurücklegt, immer kürzer. Dies schreit geradezu nach den Worten `GEH` für Schritt 1 und `DREH` für Schritt 2. Unser code sieht an diesem Punkt so aus:

```
: Schnecke
  begin geh dreh
  mitte? until ;
```

Die Worte `GEH` und `DREH` haben dabei mit der Schnecke an sich nichts zu tun, sondern sind auch für andere Experimente mit dem LED-Feld zu gebrauchen. Diese Art von Grafik hat einen Namen: Schildkröten-Grafik (englisch: turtle graphics) und wurde von *S. Papert* für die Programmiersprache LOGO erfunden. Die Vorstellung dabei ist, daß eine Schildkröte einen Stift an ihren Stummelschwanz gebunden hat und einfache Kommandos wie "vorwärts 10 Schritte, rechts 45 Grad, vorwärts 20 Schritte" versteht, um so eine Grafik zu Papier zu bringen.

Stichworte

DisCo
Programmierbeispiel
Turtle-Graphics

Die Spirale soll in der unteren linken Ecke beginnen, läuft dann den linken Rand hoch, biegt nach rechts ab ... bis sie dann schließlich in der Mitte des LED-Feldes endet. Eine kleine Skizze auf einem Stück karierten Papiers zeigt, daß der Spirale eine ganz einfache Regel zugrunde liegt:

1. 7 Schritte vorwärts, abbiegen
2. 7 Schritte vorwärts, abbiegen
3. 7 Schritte vorwärts, abbiegen
4. 6 Schritte vorwärts, abbiegen
5. 6 Schritte vorwärts, abbiegen
6. 5 Schritte vorwärts, abbiegen...

Bis auf Punkt 1 wird jede Schrittweite offensichtlich zweimal verwendet. Ebenso sieht man, daß die Schrittweite ein Parameter für das Wort `GEH` sein muß. Außerdem wird `GEH` immer von einem `DREH` gefolgt, deshalb können wir sie zusammenfassen. Der code besteht nun also aus:

```
: geh&dreh ( n)
  vorwaerts haken ;
: Schnecke 7 dup geh&dreh
  begin ( schrittweite)
    dup geh&dreh
    dup geh&dreh
  mitte? until ( n) drop ;
```

Wann ist nun die Mitte erreicht? Führt man die Betrachtung oben bis zum Ende der Spirale durch, sieht man, daß die Mitte genau dann erreicht ist, wenn die Schrittweite 0 wird. Außerdem fehlt dem code oben noch das Verkleinern der Schrittweite. Damit ergibt sich das fertige Programm:

```
: geh&dreh ( n)
  vorwaerts haken ;
: Schnecke anfang klecks
 7 dup geh&dreh
 begin ( schrittweite)
   dup geh&dreh
   dup geh&dreh 1-
 dup 0= until ( 0) drop ;
```

Damit ist der Beweis erbracht, daß das Zeichnen einer Spirale doch sehr einfach ist. Übrigens gibt es auch noch andere einfache Methoden, eine Spirale auf das LED-Feld zu zaubern.

Jetzt fehlt uns nur noch die Schildkröten-Grafik, um das Resultat betrachten zu können.

Schildkröten-Grafik

Die verwendeten Grafikworte wurden bewußt sehr einfach gehalten und umfassen nur das, was für die Spriale gebraucht wird. So ist es z. B. nur möglich, um Winkel von 90° zu drehen. Trotzdem sind sie allgemein genug, um eine ganze Reihe interessanter Grafiken damit zu erzeugen. Doch davon später mehr.

Da wir das Bild in einem kartesischen Koordinatensystem abbilden, die Schildkröten-Grafik jedoch mit einer Schrittweite und Richtung vom letzten Punkt aus, also mit Polarkoordinaten arbeitet, müssen wir uns die aktuelle Position und eben die Richtung merken.

```
2variable stift
2variable richtung
```

`stift 2@` hinterläßt die x- und y-Koordinaten des aktuellen Punktes auf dem Stack und richtung enthält nicht den Winkel (α) sondern cos α und sin α. Da wir hier keine Fixpunk-

Fortsetzung auf Seite 33

FORTH/2

ein 32-Bit System für OS/2 2.x

von Michael Major und Friederich Prinz
c/o F. Prinz, Homberger Straße 335, 4130 Moers 1

Sie arbeiten mit OS/2 und haben noch kein passendes Forth? Dann sollten Sie diese brandaktuelle Besprechung unbedingt lesen!

Nachdem Mitglieder der Moerser Forthgruppe in ihrer 'Haus-Mailbox' MHB in der MailArea /FORTHEV/NEWS erstmalig zu Beginn des Frühjahres '93 Hinweise auf ein Forth gefunden hatten, das unter OS/2 laufen und alle Vorteile dieses modernen Betriebssystems nutzen sollte, war in Moers der Wunsch geweckt, dieses Forth- System möglichst bald zu sehen. Zwar arbeitet das in Moers favorisierte ZF einwandfrei unter OS/2 2.0 in 'DOS-Boxen', was mehrere Moerser intensiv nutzen, aber ZF bleibt eben immer ein 16-bittiges Forth für DOS-Maschinen, was alle zum Thema DOS bekannten Restriktionen beinhaltet.

Diverse Nachforschungen und Versuche das 'OS/2-Forth' aufzutreiben, verliefen zunächst ergebnislos. Erst auf einem Rechner der Uni Münster wurde eines unserer Mitglieder fündig, so daß uns das nachstehend kurz zu beschreibende Forth-System erst zu Anfang Mai '93 erreichte.

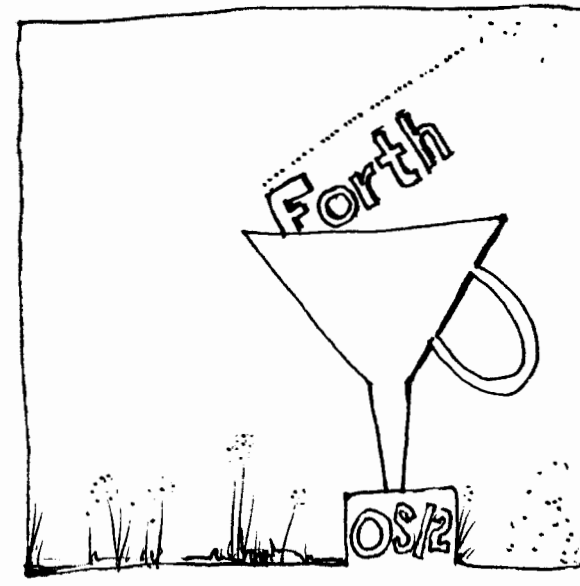
Wir hatten zunächst einmal mehr mit einem 'echten Saurier' gerechnet, als uns aus Münster via DFÜ die Nachricht erreichte, daß man uns ein Forth für OS/2 zu Verfügung stellen könnte. Programmentwicklungssysteme für 32-Bit Betriebssysteme sind bereits von mehreren Herstellern auf dem Markt, und wer Borland's 32-Bit Pascal und das C++ aus gleichem Hause schon gesehen hat, wer Microsoft's C 7.0 oder den GNU C 32-Bit Compiler kennt, der kann vermutlich nachvollziehen, daß man nach solchen Erfahrungen einem 32-Bit Forth mit sehr gemischten Gefühlen entgegensieht. Um so überraschter waren wir, als wir dieses System endlich 'zu Hause' hatten...

Stichworte

OS/2-Forth
32-Bit-Forth

Ein gerade einmal 32.793 Byte großes File mit dem Namen FORTH.EXE ist die Basis dieses bisher für OS/2 einmaligen Systems mit dem Namen FORTH/2. Der von der Firma *"BLUE STAR SYSTEMS"* in Indiana entwickelte Forthkern ist mit dem MASM 6.0 assembliert worden und enthält dementsprechend 'dichten' und optimierten Code. FORTH.EXE erwartet als Betriebssystem OS/2 2.x und setzt damit Rechner voraus, die mindestens Intels 80386 CPU beherbergen. Bei Erfüllung dieser relativ hohen (?) Anforderungen steht dem Forther ein System zur Verfügung, das erstmals alle Resourcen dieser Prozessoren ausnutzt. Endlich wird auch 'die linke CPU-Hälfte' genutzt, sprich: FORTH/2 arbeitet mit echten 32-Bit Zahlen. Die konsequente Nutzung des 'Flat-Memory-Model' macht der gelegentlich als zumindest störend empfundenen Speichersegmentierung der DOS-Systeme eine Ende.

FORTH/2 präsentiert sich in der vorliegenden Betaversion mit der Revisionsnummer 0.025 als ein reines Textsystem ohne Anbindung an die WPS (WorkPlaceShell) oder den PM (Presentation Manager) des Betriebssystems. Das heißt, daß FORTH/2 auf der Kommandozeilenobefläche von OS/2 arbeitet. Hier bleiben dem Anwender natürlich sowohl der 'Vollbildschirm' als auch das 'Textfenster' als Alternativen. Der erste Aufruf von FORTH.EXE läßt das System sich mit dem allseits bekannten OK: melden und stellt dem Forther die gewohnte Interpreteroberfläche zu Verfügung. Einen kurzen, leider nicht selektierbaren Überblick über die implementierten Definitionen vermittelt `WORDS`, dessen Ausgabe sich nur mit der 'Pause'-Taste anhalten läßt. Dabei fällt auf, daß zwar alle 'Grundfunktionen' eines herkömmlichen Forth-Systems enthalten zu sein scheinen (`DROP` und `SWAP` as usual ...) , daß aber sowohl ein Editor, als auch ein expliziter Compiler wie zum Beispiel SED im ZF-System fehlen. Insgesamt gibt sich FORTH/2 noch etwas rudimentär und läßt den von ZF verwöhnten Forther zunächst ein wenig zurückschrecken - allerdings völlig zu Unrecht ! Tatsächlich benötigt ein unter OS/2 arbeitendes System keinen eigenen Editor. Unter einem Multitasking System bereitet es keinerlei Probleme, einen beliebigen (ASCII !) Editor in einer separaten Task aufzurufen und standby zu halten. Hier bot sich zuallererst der betriebsystemeigene Editor E.EXE erfolgreich an. Aber auch das Einbinden eines Editors in das System bereitet



keine Schwierigkeiten.

Bei jedem Aufruf von FORTH.EXE wird ein File mit dem Namen FORTH.INI geladen und kompiliert. In diesem File befinden sich diverse Definitionen herkömmlicher Forth-Systeme, die technisch nicht Bestandteil eines Systemkerns für ein 32-Bit System mit preemptiven Multitasking sein müssen. FORTH.INI ist eine ASCII Datei, die sich dementsprechend einfach manipulieren läßt. Eine zusätzliche Definition 'EDITOR', die den jeweils favorisierten Editor des Anwenders aufruft, kann in FORTH.INI problemlos aufgenommen werden. Neben dieser grundsätzlichen Funktion von FORTH.INI dient dieser File gleichzeitig als ein erstes Beispiel zur Arbeit mit FORTH/2, ebenso wie fünf weitere Source-Files. Diesen Beispielen lassen sich Antworten zu weiteren Fragen entnehmen, wie zu der Frage nach dem Compiler. So zeigt sich, daß mit dem Editor erstellte Quelltexte jederzeit mit dem Aufruf von `INCLUDE` 'nachgeladen' werden können. Sicher wünscht man sich hier die gewohnte Interaktion zwischen Interpreter, Compiler und Editor. Sicher wäre es zumindest sehr bequem, wenn der Abbruch einer Kompilation den Editor aufrufen und den Text-Cursor an die fehlerhafte Stelle des Quelltextes setzen würde. Aber hier muß auf solche Bequemlichkeiten einstweilen noch verzichtet werden.

FORTH/2 bietet noch keinen 'eigenen' Assembler, wartet dafür aber mit einem positionellen `CASE`-Construkt, mit FIX-Point Zahlen von 0,0001 bis 200.000,0000 und mit lokalen Variablen auf, wobei vor allem Letzters immer wieder Inhalt verschiedenster Diskussionen war. 'SYSCALL's ermöglichen eine direkte Anbindung der eigenen Applikationen an die besonderen Fähigkeiten des OS/2, wobei an dieser Stelle gesagt werden muß, daß die hier potentiell schlummernden Möglichkeiten tiefergehendes Wissen um OS/2 und seine Interna voraussetzen. Alles in allem ist FORTH/2, dessen Entwicklungsbeginn erst knapp 1 Jahr zurück liegt, ein 'Fuß in der Tür' der OS/2 Welt, in welcher FORTH nur einem Bruchteil der Konkurrenz ausgesetzt ist, gegen die sich 'unsere Sprache' auf der DOS-Ebene seit 20 Jahren nur schwer behaupten kann. Wenn schon innovative Systeme wie NAXOS in der Forthgemeinde keinen Wiederhall erzeugen konnten, sollte FORTH/2 durchaus breitere Beachtung finden (sofern die notwendigen Hard- und Softwarevoraussetzungen erfüllt werden), umsomehr, als FORTH/2 ein 'echtes' Forth ist.

Forth/2 ist ein Fuß in der Tür der OS/2-Welt

Den immer etwas 'hakeligen' Einstieg erleichtern nicht nur die dem System beiliegenden Quellfiles und die Datei FORTH.INI, sondern auch drei *.DOC Files, in denen neben der bisherigen Entwicklungsgeschichte von FORTH/2 auch die Funktionen der implementierten Definitionen inklusive Stackbilanzen beschrieben sind. FORTH/2 ist allerdings kein Public Domain Erzeugnis, sondern gehört in die Kategorie der Shareware. Der Autor stellt dem Anwender sein Produkt für 30 Tage zur Probe zur Verfügung und erbittet danach eine Registrierung, die für eine einzelne Maschine gegen Einsendung von 20.- $ vorgenommen wird. Weiterhin bietet *"BLUE STAR SYSTEMS"* zusätzliche Lizenzen gegen jeweils 10,- $ Dollar an. Als Clou kann an der nachstehenden Adresse sogar der Assembler Quellcode für den Kernel (FORTH.EXE) gegen Einsendung von 50,- $ bezogen werden. Vor allem den an den Interna des Systems und an den Schnittstellen zum Betriebssystem interessierten Forther wird letzteres Angebot sicher besonders angenehm auffallen.

Zu beziehen sind sowohl die angesprochenen Lizenenzen und Quellen, als auch das System selbst bei:

BLUE STAR SYSTEMS,
PO BOX 4043,
Hammond, Indiana 46324

Das FORTH/2 in der in Moers vorliegenden Version gibt die Moerser Forthgruppe selbstverständlich an jeden Interessenten gerne weiter.

❑

Gehaltvolles

zusammengestellt von Klaus-Peter Schleisiek

Inhaltsübersicht: Forth Dimensions der Forth Interest Group, USA

6, Vol.14 März/April 1993

1, Vol.15 Mai/Juni 1993

68HC11

und noch mehr Forth ...

von Jan Michaels & Holger Dyja
H. Dyja, Naumannstr. 13, W-1000 Berlin 62, Tel.: 030-784 12 57

Die geringe Größe, die ein Forth-System benötigt, macht es gerade für Mikro-Controller mit geringem Speicherplatz zur idealen Entwicklungsumgebung. Interaktive Testmöglichkeiten machen Hardwaretests zum Kinderspiel. Warum gerade der 68HC11 so forth-tauglich ist, wird hier beschrieben.

1991 begannen wir, damals noch getrennt voneinander, unsere Ideen zu verwirklichen, Forth auf einen Microcontroller zu portieren. Dabei sind wir verschiedene Wege gegangen, die sich kreuzten und zusammenführten. Zwei Tatsachen machten uns den gemeinsamen Weg einfach. Erstens hatten wir uns den selben Microcontroller, den Motorola 68HC11, vorgenommen. Und zweitens implementierten wir das gleiche Speichermodell. Dadurch konnten die zwei Wege, Forth-System und Target- Compiler, zusammenlaufen. Heute bieten beide mit zusätzlichen Tools eine komplette, in sich geschlossene Entwicklungsumgebung, die wir um ein Controllerboard erweitert haben.

In dieser kleinen Serie wollen wir in der Vierten Dimension die Forth-Tauglichkeit des 68HC11 beschreiben, und Möglichkeiten mit Forth eine Controller- Anwendung zu realisieren die einem simplen Beispiel demonstrieren.

Die Tools

Die Tool-box, die wir vorstellen, besteht im wesentlichen aus vier Teilen.

- einem interaktiven Forth-System, das auf dem Microcontroller läuft und direkt die Eingabe von Forth-Worten erlaubt. Damit ist insbesondere das interaktive Testen von Programmteilen und dem Zusammenspiel mit der Hardware möglich.
- einem Terminal-Programm mit integriertem Editor. Hiermit lassen sich Forth-Quelltexte editieren und sofort zum Target übertragen. Dies ist die PC-Oberfläche zum Forth-System.
- einem optimierenden Forth-Target Compiler. Quelltexte können hiermit direkt in Motorola S-Records umgewandelt werden und sind auf dem 68HC11 ausführbar.
- einem Monitor und Debugger. Dies ist die Oberfläche zum Arbeiten mit dem Target-Compiler. Alle vom Target-Compiler erzeugten Symbole stehen zur Verfügung und können für komfortables symbolisches Debugging benutzt werden.

Stichworte
Mikro-Controller
68HC11
Hardware

Familienchronik

Das heutige Angebot und die Vielfalt der sich auf dem Markt befindlichen Microcontroller machen die Auswahl recht schwer. Unter den vielen Microcontrollern sind die 8051-Familie von Intel und die 68HC11-Familie von Motorola besonders interessant. Der CPU-Kern der 68HC11-Familie macht diese Controller besonders Forth-tauglich. Dies und die auf dem Chip enthaltene Peripherie machte uns die Entscheidung für diesen Controller besonders leicht. Ein kleiner Überblick über die 68HC11-Familie soll zum einen zeigen, was man heute von Microcon-

Auf dem Markt befindliche 68HC11 Derivate:

Typ	wesentliche Eigenschaften
68HC11 A0...A8	Daten-/Adressbus multiplexed, 3IC, 5OC, 8ADC, 256byte RAM, 512byte EEPROM, z.Zt. 12MHz max., on-chip ROM maskenprogrammierbar
68HC11 D3	Daten-/Adressbus multiplexed, 4IC, 5OC, 0ADC, 196byte RAM,0byte EEPROM, 4kbyte on-chip ROM maskenprogrammierbar
68HC11 E0...E9	Daten-/Adressbus multiplexed, 3/4IC, 4/5OC, 8ADC, 512byte RAM, 512byte EEPROM, z.Zt. 12MHz max., on-chip ROM maskenprogrammierbar
68HC11 F1	Daten-/Adressbus nonmultiplexed,3/4IC,4/5OC,8ADC, 1024byte RAM 512byte EEPROM relocatibel, 4 CS programmierbar, 16 MHz

IC= InputCapture, OC= OutputCompare, ADC= AD-Konverter, CS= Chipselects

Highlights zukünftiger Entwicklungen:

68HC711G5	8 10bit ADC, 4PWM, 3/6IC, 4/7OC, 2.Timer, 16kByte EPROM
68HC11K4	1 Mbyte Adressraum, 4 PWM, 4 CS, 16MHz
68HC711N4	12 ADC, 2 DAC, 6 PWM, 16bit Math Coprocessor, 24 kByte EPROM

PWM= Pulsweitenmodulator, DAC= DA-Konverter

Abb. 1

trollern erwarten kann, und zum anderen, was sie besonders Forth-tauglich macht.

Die 68HC11 Microcontroller-Familie ist eine Weiterentwicklung der 6800/01/02/03 8-Bit Microprozessoren, die einerseits im Befehlssatz und andererseits mit Controllerperipherie "on-chip" erweitert wurden. Sämtliche Mitglieder dieser Familie sind in stromsparender HCMOS-Technologie gefertigt. Bei allen 68HC11 Versionen wird eine Quarzfrequenz von 16MHz angestrebt. Diese Frequenz wird durch einen Teiler von 4 auf den Systemtakt heruntergeteilt.

Der 68HC11F1

Am Beispiel vom 68HC11F1 soll der Aufbau und die Funktion des 68HC11 beschrieben werden.

Blockschaltbild 68HC11F1

Der 68HC11F1 ist für Microcontrollersysteme mit externem RAM, externem EPROM und externer Peripherie besonders geeignet, wie es bei Prototypen und Unikaten häufig notwendig ist. Diese Eigenschaften werden durch die frei programmierbaren Chipselects unterstützt, so daß einfache Anwendungen ohne großen Hardwareaufwand realisierbar sind.

Der CPU-Kern ist dem 6800/02 ähnlich aufgebaut und um ein Indexregister Y, 16-Bit-Befehle, 16-Bit-Division, 8*8 Bit Multiplikation und Bit-Manipulationsbefehle erweitert worden. Die 68HC11 CPU besitzt zwei 8 bit Akkumulatoren die zu einem 16 bit Akkumulator zusammengefaßt werden können, zwei 16 bit Indexregister, Conditioncode-Register, Programmcounter und Stachpointer. Gerade die 16 Bit breiten Indexregister und die 16 Bit Befehle machen den 68HC11 besondes geeignet für Forth-Systeme. Zum Beispiel braucht der 68HC11 nur 11 Zyklen (bei 16MHz 2.75µs) und 4 Byte um Parameter auf/von dem Parameter-Stack zu bewegen:

```
  08 INX       (3)
ED00 STD 0,X   (5)
  08 INX       (3)
  09 DEX       (3)
EC00 LDD 0,X   (5)
  09 DEX       (3)
```

Dagegen sind die Bit-Manipulations-Befehle nur schwer auf Forth-Ebene ausnutzbar.

Im direkten Vergleich mit der 8051-Familie fällt beim 68HC11 auf, daß alle Hardware-Register memory-mapped sind. Sie lassen sich also direkt über die Forth- Worte `C@` und `C!` ansprechen. Spezielle Operatoren, um auf Codespace, "on-chip" RAM oder auf externes RAM zuzugreifen, sind nicht notwendig. Auch unterscheiden sich Zugriffe auf externes RAM oder "onchip" RAM nicht. Solche unterschiedlichen Zugriffe sind damit auch in Hochsprache gleich schnell und über die Indexregister möglich.

Neben der Forth-Tauglichkeit ist 68HC11 mit seiner "on-chip" Peripherie auch jeder Controlleranwendung gewachsen. Auf diese Peripherie (Ports, AD-Wandler, Timer, Chip-selects und serielle Schnittstellen) kann über einen 96 Byte großen Registersatz zugegriffen werden, mit dem die Datenrichtungen der Portleitungen bestimmt und den Ports folgende Sonderfunktionen zugeteilt werden:

- PortE
  8-kanaliger 8-Bit AD-Konverter
- PortA
  4 InputCapture, 5 OutputCompare und 1 Pulsakkumulator
- PortD
  1 RS232 Schnittstelle
  und 1 synchrone ser. Schnittstelle
- PortG 4
  programmierbare Chip-selects

Die Input-Capture-Eingänge können dazu verwendet werden, Zeiten zu messen und Ereignisse zu zählen. Mit den Output- Compare Ausgängen lassen sich dagegen komplexe Ausgangssignale erzeugen, wie z.B. Pulsweitenmodulatoren, wobei anzumerken ist, daß die Input-Capture Eingänge auch als zusätzliche Interrupteingänge benutzt werden können. Selbst wenn die Output-Compare Ausgänge als I/O-Port belegt sind, stehen die Output- Compare-Counter als zusätzliche Timer weiterhin zur Verfügung. Gerade in dem ausgefeilten Timersystem besteht die Stärke des 68HC11 für Controlleranwendungen, weshalb auch konsequent jedem Timer ein eigener Interruptvektor gegeben wurde.

Insgesamt verfügt der 68HC11 über 20 Interruptvektoren und einen Resetvektor. Dabei dienen die Funktionen und Interrupts "Watchdog", "Illegal-Opcode" und "Clockmonitor" besonders der Sicherheit einer Microcontoller-Anwendung.

Das größte on-chip-RAM stellt der 68HC11F1 mit 1kByte zur Verfügung. Es befindet sich normalerweise ab der Adresse $0000, kann aber wie der Registersatz und das eingebaute EEPROM auf jede beliebige Adresse im 4kByte-Raster verschoben werden. Damit ist es möglich im Gegensatz zu den A und E Versionen 56kByte ununterbrochenen Adreßbereich zu erhalten, ohne das eingebaute EEPROM abzuschalten. Die Adresslage des EEPROM ist ab Werk bei $FE00-$FFFF, die für einfache Singlechip Anwendungen am günstigsten ist. Auch für Testzwecke ist diese Konfiguration interessant da dort die Vektoren schnell geändern werden können. Eine weitere interessante Eigenschaft ist die Möglichkeit den Registersatz in die "Zero-Page" (ab Adresse $0000) zu verlegen. Auf diese Adressen kann der 68HC11 um ca. 30% schneller zugreifen, so daß beschleunigte I/O- Operationen mit den Ports durchführbar sind.

Ein 68HC11F1-Microcontrollerboard wird Mitgliedern der Forth-Gesell schaft incl. Software, RS232-Kabel und Netzteil zum Ausleihen zur Verfügung gestellt.
Holger Dyja,
Tel. 030/784 12 57
Naumannstr.13 ,1000 Berlin 62

Literatur

- MC68HC11F1
  Technical Data, Motorola
- MC68HC11F1
  Programming Reference Guide,
  Motorola, MC68HC11F1RG/AD
- MC68HC11
  Programmer's Reference Manual,
  Motorola, MC68HC11PM/AD
- MC68HC11
  Reference Manual,
  Motorola MC68HC11RM/AD

Besondere Beachtung verdienen auch die Stromspar-Modes, in die sich der 68HC11 über die Befehle STOP und WAI versetzen läßt, wobei nach STOP sogar alle Zähler und der Quarzoszillator angehalten werden, um den Energieverbrauch zu minimieren. Aufgeweckt wird der 68HC11 nach WAI durch jeden Interrupt. Aus dem STOP-Mode kommt man nur durch einen Reset oder einen XIRQ (pseudo nonmaskable Interrupt) heraus.

Alle 68HC11 sind in 4 Betriebsarten betreibbar. Der "Singlechip"-Mode ist eigentlich nur sinnvoll bei 68HC11- Versionen mit eingebautem ROM, EPROM oder EEPROM. Die Hauptbetriebsart des 68HC11F1 wird der "Expanded-nonmultiplexed"-Mode sein, der die Daten-, Adreß- und Steuersignale des CPU-Kerns an den Port's B, C und F herausführt. Zusätzlich gibt es noch den "BOOT"- und "Test"-Mode, die der Programmentwicklung auf dem 68HC11 dienen.

(wird fortgesetzt).

❑

Kunst und Forth kps

Neulich war ich in Aachen in einem neuen Museum für moderne Kunst wo auch die Elektronik einen Platz hat. Eines der Objekte besteht aus zwei vertikalen Leuchtschrift-Bändern, jedes 2 Stockwerke hoch. Die Texte stammen von der New Yorker Künstlerin *Jenny Holzer*, die ihre Ansichten und Einsichten oder vielleicht auch Provokationen in englischer Sprache produziert; alle vier Stunden von neuem, aber wer bleibt schon so lange? Die Block-Buchstaben sind mit zahlreichen Attributen abgewandelt: rot, gelb, grün, mit Rändern, mit Flammen und so weiter.

Der laufende Text läßt keinen Widerspruch zu. Dagegen wehre ich mich mit der Überlegung, wie in Hard- und Software's Namen wohl diese Massen von roten und günen Leuchtdioden der Leuchtbänder gesteuert werden. Da sich die Künstlerin mit Datenstrukturen und Pixelarithmetik herumschlägt, das mag ich nicht glauben. Sie definiert vielleicht den Zeichensatz und schreibt ansonsten einfach ihren Text, gemischt mit Attributs-Anweisungen wie "geflammt" statt etwa ESC-f. Auch Compilieren und Linken geben nicht gerade das nötige schöpferische Feedback. Was sie braucht, das ist ein programmierbarer Interpreter. Wer schon einmal mit dem kleinen 8 mal 8 Leuchtdiodenfeld KlaRo und der DisCo-Software gespielt hat, der hat schon eine Ahnung, womit hier wohl programmiert worden ist.

Ich wollte es genau wissen und habe mich mit der dankenswerten Unterstützung des Ludwig-Forums und der Kölner Galerie Monika Sprüth bei der Elektronik-Firma erkundigt, die die "Holzer Signs" programmiert hat. Fa. SUNRISE SYSTEMS, INC. in Pembroke, MA, USA antwortete wie folgt:

Ihre Vermutung über die Architektur trifft im Ganzen zu. Die untere Ebene der LED-Steuerung besteht aus verteilten Prozessoren in den Anzeige-Türmen, die in Assembler programmiert sind. Jeder Turm ist mit einem PC verbunden, auf dem die höheren Programm-Ebenen mit HS/FORTH realisiert sind, einem ausgezeichneten Produkt. Das System ist praktisch beliebig erweiterbar; nachdem es zuerst im Guggenheim in New York City gelaufen ist, haben wir es in mehreren Variationen auch kommerziell eingesetzt.

Na bitte: dahinter steckt wieder mal ein kluger Kopf; und der nimmt natürlich in solch einem Fall Forth! Life zu besichtigen im Ludwig-Forum zu Aachen.

Gemischtes rk

Verwöhnt...

...wird die Redaktion normalerweise nicht, wenn Sie um Reaktionen der Leser bittet. Nur ein kleines Beispiel: In der VD 3/1993 S.30 bat ich um Informationen zu Forth in der Meßtechnik, weil an einer Schule entsprechende Soft- und Hardware angeschafft werden sollte. Resonanz: Null-Komma-Null! Doch nun die Überraschung: Sechs (!) 'Aufnehmer' fanden den Weg zur VD-Redaktion! Das war die Antwort auf den Artikel "Guter Aufnehmer gesucht" (VD 1/1993, S.37). Vielen Dank für die Zuschriften. Es sind einfache, aber auch eigenwillige Lösungen dabei. Für Anfängern kommen allerdings nur zwei Lösungen als Musterlösung in die engere Wahl. In eine der nächsten VD's wird dann eine Version vorgestellt werden.

Zuwachs...

...hat die ModuNORM Produkt-Familie der Firma *Forth-Systeme GmbH* erhalten. Die Redaktion erhielt schon vor einiger Zeit die Geburtsanzeige des hoffnungsvollen Sprößlings mit dem schönen Namen CPU 68332. Das 62x95mm² kleine Modul soll hervorragend für Anwendungen im Echtzeitbereich geeignet sein, da das Interruptverhalten sowie die gute Peripherieunterstützung auf der CPU sehr leistungsfähige Systeme erlaubt. Es bietet hochflexible I/O-Ports, eine serielle Schnittstelle, Echtzeituhr und 16 MHz Systemfrequenz. Neben dem MPU (132 Pin Quad-Flat-Pack) ist auf dem Modul 256kByte SRAM, RTC, RESET- und Batterie-Backup-Logik mit Spannungsüberwachung für RAM und RTC, Bus-Pufferung, Adressdekodierung, sowie ein Boot-Stecksockel für maximal 128kByte EPROM sowie 2 Flash-Eproms mit maximal 512kByte untergebracht. Ebenfalls erwähnenswert ist, daß der Motorola-Prozessor auf die schnelle Abarbeitung von Hochsprachen ausgelegt ist. Zur ModuNORM-TM Familie gehören auch Module mit Siemens SAB 80c166, Hitachi 64180, NEC V25, Harris RTX2000/RTX2001, Hitachi 6303 sowie Drucker-Controller für Fujitsu und Seiko Thermodruckwerke und ein LCD-Controller für fast alle gängigen Grafik LCD's.

Gewonnen...

...haben *Ulrich Hoffmann* und *Andreas Dobbertin* den Programmierwettbewerb auf der Kongressmesse *Echtzeit '93* (Karlsruhe, 16.Juni). Programmiergegenstand war ein zweirädriger, zweiäugiger (daher der Name KuckKuck) Roboter. Dieser mußte entlang einer großen, schwarzen Acht, die auf eine Holzplatte gemalt war, bewegt werden: zweimal oberer, zweimal unterer Kreis, und so forth. Das Team, dessen Tierchen mindestens dreimal auf diese Art die Acht durchlaufen konnte, hatte gewonnen. Neun Teams nahmen am Wettbewerb mit sehr unterschiedlicher Soft- und Hardware teil (siehe dazu: Die Sieger von akg) Ulli siegte nach 2,5 Stunden mit Forth (F-PC) und einem 386sx-SchleppTop mit 8255 IO Karte. Keines der anderen Teams konnte in der vorgegebenen Zeit die Aufgabe vollständig lösen. Nun kommt auf Ulli die Aufgabe zu, das Programmierproblem für den Wettbewerb im nächsten Jahr bereitzustellen. Lieber Ulli, mach's nicht zu schwer, damit auch andere Programmiersprachen eine Chance haben. Sonst sind demnächst die Forther bei dem Wettbewerb unter sich...! (Einen ausführlicheren Bericht finden Sie unter de.comp.lang.misc)

Gelohnt...

...hat sich nach Ansicht von *Ulrich Hoffmann* die Präsenz der Forth-Gesellschaft auf der Echtzeit-Messe. Die FG war dort wieder mit einem eigenen Stand vertreten. Optisch ansprechend gestaltet und ausgestattet mit einigen Lockvögeln (Großes und kleines Lampenfeld mit *Kretzschmar/Schleisiek*'scher DisCo-Software; forth- gesteuerter Schnelldrucker von GFC/Düsseldorf; Die 'Bunte Box' vom vorjährigen Programmierwettbewerb mit *Arndt Klingelnberg* an den Keyboards; uvm.) Forth auf der Echtzeit bedeutet: zeigen, daß Forth als alternatives System zur Lösung von Echtzeitprogrammieraufgaben durchaus seine Existenzberechtigung hat. *Ulrich Hoffmann* dankt all denen, die dazu beigetragen haben, daß der diesjährige Stand zu den erfolgreichen Aktionen der Forth-Gesellschaft gezählt werden kann.

Ulrich Hoffmann: Der Gewinner des Programmierwettbewerbs

Termine

ATARI
1993aug20--22 Düsseldorf
Internationale Funkausstellung
1993aug27--sep05 Berlin
MessComp '93
7. KongressMesse für industrielle Meßtechnik
1993sep07--09 Wiesbaden
EMO
Europäische WerkzeugMaschinen Ausstellung
1993sep14--22 Hannover
euroFORTH '93
1993oct15--18 Marienbad
Tschechische Republik
SYSTEMS
(Computer und Kommunikation)
1993oct18--22 München
PRODUCTRONA
(Electronic-Fertigung)
1993nov09--13 München

(bitte geben Sie uns Ihren Input zu für Forther sinnvollen Veranstaltungen: Forth-Kurse, Forth-Anwendungen, Forth-Kunden, Forth-Konkurrenten und Hardware für Forther)

Programmierwettbewerb auf der Echtzeit '93 — akg

Die gestellte Aufgabe wurde nach einer Programmierzeit von zweieinhalb Stunden vom Team Hoffmann als erstem und einzigen vollständig gelöst. Der Wettbewerb wurde nach etwa 3 Stunden beendet und die weiteren Plätze nach dem erreichten Lösungsstand vergeben.

1. Platz:
U. Hoffmann, A. Dobertin, Kiel
386SX-Laptop mit 8255 I/O-Karte, Forth (F-PC)

2. Platz:
H. Heinle, H. Wagner, R. Steinhauser, Legau
486PC, Borland C++

3. Platz:
G. Kleinhaus, M. Kleinhaus, A. Schasse, Herne
68000 Einplatinenrechner (Kat'ce 1.3), Kat'ce Betriebssystem

4. Platz:
D. Peter, Wuppertal
PIC16C54, PICALC-Assembler

5. Platz:
K. Heinz, Dr. W. Wejgaard, Synics AG, CH-Regensdorf
PC+68HC11, Forth und Holon

6. Platz:
U. Keinki, H. Pippig, H. Krause, Ing.-Büro Keinki, Schwalbach
Single-Board-Computer mit 80C537, Cross-Assembler

7. Platz:
K. Fr. Gebhard, H. Vogt, Berufsakademie Stuttgart
HP9000/425e + CPU30-ZBE(Force) + A201-M11(mem),
VX Works + HP-UX mit X-Windows

8. Platz:
Dr. Ing. Woitzel, U. Schütz, FORTecH Software, Rostok
68HC11-SBC (Elrad_Mops), comFORTH

9. Platz:
A. Klingelnberg, Klingelnberg Consultants, Alsdorf
386SX ohne Zusatzhardware, Forth83 F-PC Version AK

FORTH ganz GROSS

Das z.Z. größe Forth-System wird auf dem Flughafen von Riad, der Hauptstadt Saudi Arabiens, betrieben. Das enorme Kontrollsystem besteht aus einem Netzwerk von 700 poly-FORTH Computern, die über Hochgeschwindigkeits-Hardware verbunden sind. Eine ClusterForth-Ebene erlaubt eine einfache Programmierung und Softwareübertragung an jeden angeschlossenen Computer. Das Kontrollsystem überwacht und steuert über 26.000 Sensoren/Aktoren und 800 Kartenleser für ca. 12.000 Angestellte.

Quelle: FD, Vol.15, Heft 1, 1993

FORTH ganz klein

...Im Gegensatz zu den vorher genannten Sprachen kommt FORTH mit etwa 16 KByte Speicher aus und verbindet die Vorteile eines Interpreters mit der Geschwindigkeit eines Compilers. Die Sprache ist sehr stark strukturiert und kennt keine GOTO-Anweisung. Sie läuft auf praktisch jedem bekannten Computer (...). Wenn Sie an kleinen (siehe oben ! rk) Anwendungen interessiert sind, die sich mit der Steuerung bestimmter Vorgänge befassen, und eine Hochsprache bevorzugen, sollten Sie FORTH ernsthaft in Erwägung ziehen. Nachdem Sie dieses Buch gelesen haben, sollten Sie den Stapelspeicher (stack) schätzen gelernt haben, der auch die zentrale Datenstruktur in FORTH ist. Die Datenverarbeitungsfähigkeiten von FORTH müssen allerdings vom Programmierer organisiert werden, und die umgekehrte polnische Notation bedarf einiger Übung. Außerdem lebt man in FORTH in einer eigenen Welt und kann nicht ohne weiters den Code aus anderen Sprachen übernehmen. Da eingefleischte FORTH-Benutzer zu Fanatismus neigen, weisen wir lieber gleich darauf hin, daß ein FORTH-Benutzer unsere Kritik sang- und klanglos zurückweisen wird.

Aus: Aho, Alfred V., Compilerbau, Teil 1, 1992-2, Addison-Wesley (Deutschland), S.483 zugefaxt von: W. Schemmert

Fortsetzung von Seite 26 (Schnecke)

tarithmethik verwenden, sondern mit ganzen Zahlen rechnen, sind wir so auf die Werte 0 und 1 eingeschränkt und somit auf die Winkel 0°, 90°, 180° und 270°. Um einen so gegebenen Winkel um 90° zu verändern, werden die folgenden beiden Worte verwendet:

```
: 90+ ( x y - x' y')
  negate swap ;
: 90- ( x y - x' y')
  swap negate ;
defer rechter-winkel
```

Sie sind die beiden möglichen Implementationen des Vektors rechterwinkel

Da wir nur um rechte Winkel drehen können, was mehr einem Haken entspricht, wollen wir das Wort auch lieber `haken` nennen.

```
: haken richtung 2@
  rechter-winkel
  richtung 2! ;
```

Nun brauchen wir nur noch das Wort `vorwaerts` zu definieren.

```
: vector+
  ( x1 y1 x2 y2 - x y)
  >r rot + swap r> + ;
defer (klecks) ( x y)
: klecks
  stift 2@ (klecks) ;
: schritt stift 2@
  richtung 2@ vector+
  stift 2! ;
: vorwaerts ( n)
  0 ?do schritt klecks
  loop ;
```

`(klecks)` ist die einzige Schnittstelle zur Grafikdarstellung. Seine Aufgabe ist es, an der Position x, y einen Punkt anzuzeigen.

Hier ist noch ein Wort zur Initialisierung des Grafikpakets. Es stellt den Stift in den Koordinatenursprung und setzt die Richtung auf 0°, d. h. gerade nach rechts gehen.

```
: anfang 0 0 stift 2!
  1 0 richtung 2! ;
```

Aufgaben:

- Ändere Schnecke so, daß nur ein 4x4 großes Quadrat gefüllt wird.
- Was muß dann geändert werden, um einen 4x4 großen Rhombus zu füllen, so daß die Spirale um 45° gedreht erscheint?

❑

Anmerkung von R.Kretzschmar an seine Schüler: Liebe Schüler, die Ihr von mir in Klausuren Punkte abgezogen bekommt, wenn Ihr keine oder unvollständige Stack-Kommentare schreibt: Ich konnte den Autor nicht von seiner Ansicht abbringen, daß Stack-Kommentare dann unnötig sind, wenn nichts auf dem Stapel passiert und daß es reicht, nur die Eingabe zu benennen, wenn kein Wert hinterlegt wird. Beruft Euch nicht auf diesen Artikel! Ich hinterlasse gnadenlos weiter rote Tinte!

Hier funkt's

rk

Unter den Forthern gibt es einige Funkamateure. Zwei unserer drei Direktoren (Ulrich Hoffmann, Thomas Beierlein) und einer der VD-Redakteure (rk) sind in diesem Hobby mehr oder weniger aktiv. Paket-Radio ist eine Betriebsart, die Computertechnik und Funktechnik miteinander verbindet. Ich bin unter dem Rufzeichen DF6KR über die Mail-Box DK0MWX für andere Funkamateure erreichbar. Mitteilungen, die Amateurfunk und Forth betreffen, können dort für mich hinterlegt werden. Wenn es zu speziell ist, werde ich eine Mitteilung an die Redaktion eines Amateufunkerfachblattes senden. Vielleicht kann auf diesem Weg die Frohe Botschaft Von Forth auch in die Gemeinde der Funker getragen werden. Doch ist zu befürchten, daß die Funker unter den Forthern zu beschäftigt sind. Denn mein erster Aufruf, es mögen sich die forthelnden Funker bei mir melden, blieb ohne Echo! Schade!

Outing

Es sollte ein Sonderpreis für die Firmen/Personen gestiftet werden, die sich nicht scheuen, Forth im Firmennamen und/oder auf der Visitenkarte zu nennen. Ohne größere Recherche fallen mir nur zwei europäische ein: FORTecH und Forth-Systeme GmbH. Besonders hat mir der Briefkopf von Herrn Wejgaard gefallen:

: Wolf Wejgaard
Dr.sc.nat.dipl.Phys.ETH
Neuhöflirain 10
CH-6045 Meggen

Forth Engineering
Anwendung, Entwicklung, Kurse
041 • 37 37 74 ;

Jetzt bitte nicht mosern, sondern umgehend Ihre Firmen-Karte mit Forth-Hinweis an die Redaktion senden! Die ersten zehn werden abgedruckt! (Nicht schummeln! Einzelstücke werden nicht berücksichtigt!)

Überhaupt...

...bin ich immer wieder überrascht, wer alles MikroController mit Forth als Vatersprache anbietet. Wie gerne würde ich eine vollständige Liste veröffentlichen; einträchtig nebeneinander auf ein oder zwei Seiten; zum kopieren und weitergeben! Derart hätte man doch gerne die Qual der Wahl! Oder...?

Forth in der Elrad

von Winfried Wendler

Es kommt nicht oft vor, daß Forth in den größeren Zeitschriften behandelt wird. Um so bemerkenswerter ist eine Artikelserie ab Heft 5/93 in der Zeitschrift Elrad. Berichtet wird dort ausführlich über ein komplettes Forthpacket aus den USA. Dieses Paket (Payne Forth) enthält je einen Compiler für den PC und für die 8051 Prozessor-Familie. Zu diesen Compilern sind auch die passenden Editoren und Assembler beigefügt. Betrieben wird das 51er Forth mit Hilfe eines speziellen Terminalprogramms. Über dieses Terminalprogramm kann das 8051 Forth auf den Massenspeicher des PC's zugreifen. Das Sahnestückchen dieses Packetes ist ohne Zweifel der Methakompiler. Mit ihm ist es möglich, nach Abschluß der Programmierphase ein EPROM zu erstellen. Der Quelltext braucht zur Methakompilation nicht mehr verändert zu werden. Von allen Programmen des Paketes sind die Quelltexte beigefügt. Die mitgelieferte Dokumentation ist trotz ihres Umfangs recht lückenhaft. Besonders der Methakompiler wird nur dürftig beschrieben. Von den Autoren ist ein ein Buch erhältlich, das im wesentlichen die Quelltexte und die Hardwarebeschreibung enthält. Zum Betrieb des Forth ist ein 8051 Bord nötig, das eine zusätzliche Schnittstelle mit einem 8250 besitzt. Das Payne Forth benutzt die prozessoreigene Schnittstelle nicht. In der Elrad 5/93 ist nun ein Rechner beschrieben, der speziell für diese Forth entwickelt wurde. Eine Diskette mit dem Forth ist bei mir gegen Einsendung von DM 20 erhältlich.

Winfried Wendler
Aachener Strasse 43
10713 Berlin
(030) 822 93 43

Interview mit Tom Zimmer

Tom Zimmer nahm an der Forth Tagung 1993apr23--25 in Nürnberg teil. Danach schlossen sich zwei Tage Sightseeing an. Während eines Autobahntransfers zwischen den F-PC-Hochburgen Alsdorf und Hilden, fand sich noch etwas Ruhe, Tom Zimmer zu bitten, den Lesern der VD über sich und Forth etwas mitzuteilen:

VD: Tom, Du bist bekannt für's F-PC, dem in Deutschland wohl am meisten genutzen PD-Forth-System auf dem PC und den Target Compiler TCOM. Tom, Du warst von *Dirk Brühl* eingeladen worden. Ich finde es toll, daß das geklappt hat. Vortrag und Diskussionsrunden auf den Forth Tagen sind nun vorüber, aber die Leser der *Vierten Dimension* (ich hatte Tom bereits vorher in USA mit einigen Probehefen eingedeckt) möchten mehr über Dich und Deine Arbeit mit Forth wissen. Wann startestest Du mit Forth und warum eben Forth und nicht -C- oder Pascal oder ... ?

Tom: So, ersteinmal möchte ich meinen Dank zum Ausdruck bringen für die Gelegenheit, hier herüber zu kommen und auf der Konferenz sprechen zu können. Es was sehr interessant und es hat mir viel Freude bereitet.

F-PC erzeugte bei den kommerziellen Forth-Vertreibern Druck, zu ihrem Produkt eine komplette Werkzeugkiste zum Debuggen mitzuliefern

Ich begann mit Forth wirklich zu arbeiten Anfang der 80er Jahre, ich suchte nach einer Sprache, die gleichzeitig mächtig und schnell sein sollte. Ich hatte Erfahrung mit Basic aber noch keine Kenntnisse über -C- . Basic emfand ich zwar als einfach, aber als zu langsam. Den ersten Kontakt mit Forth bekam ich durch *microforth*, das war von PolyForth, für den 8080. Ich tippte die Quelltexte in ein Intel µP-Development System und assemblierte das dann. Anfangs war ich mir nicht ganz sicher, was das alles sollte, aber es erschien recht interessant. Danach versuchte ich mich an eigenen Implementationen von Forth bis die fig Version kam, die von der Forth Interest Group, so ungefähr 1978.

Ich bezog eine Version für den 6502 , ein Freund tippte es ein und ich fing damit an zu arbeiten. Ab 1986 war ich bei Maxtor in der 'Test engineering group' beschäftigt, es ging um Software für die Diagnose der Festplatten, die Maxtor bekanntlich herstellte. Zu diesem Zeitpunkt hatte ich eine genügend fundierte Forth Erfahrung, um mit Forth an dieser Diagnose-Software zu arbeiten. Ich hatte mit dem kommerziellen LMI Forth gearbeitet; bei Maxtor nutzen wir dann aber F83. Speziell nutzten wird die Version F83Y, die die Header getrennt hält. Ich nahm strukturelle Änderungen an F83Y vor, damit wir mit größeren Programmen arbeiten konnten und mit sequentiellen TextDateien.

VD: Üblich lesbare, eben sequentielle Forth Quelltexte waren zumal damals ja unüblich. Wieso ändertest Du gerade das? Das kostet doch einiges an Aufwand!

Tom: Ich glaube, daß TextDateien viel mehr Akzeptanz für Forth bringen, zumal für die traditionellen Computer-Wissenschaftler, und ich meinte, daß das die Adaption zu normalen TextDateien hin ganz wichtig sei, damit Forth auch weiterhin genutzt werden würde.

So von 1986 bis 1989 , da wurde wohl F-PC veröffentlicht , arbeitete ich an verschiedensten Forth Versionen mit vielen Namen und unterschiedlichen Eigenschaften. Von DF , das war ein 'Direct Threaded', zu ZF , das war wohl das letzte bevor es dann F-PC hieß.

Ich erstellte einen TextDatei-Editor dazu, und das war dann wohl PF , das stand für Prefix Assembler Forth. Und das gab ich -- unglücklicherweise -- so mit diesem Namen heraus, wohl auf der FORMEL Konferenz in 1987. Unglücklicherweise war das aber eben auch die Abkürzung für PolyForth. Also wurde das dann umgetauft in ZF, und, wie ich gehört habe, wird das wohl noch von einigen in Deutschland genutzt und wohl auch in den USA. ZF, ja das war recht nett, es war eben klein gegenüber F-PC . Der Raum war begrenzt, aber es war eben kürzer und schnell; eben gut nutzbar zum Experimentieren.

VD: Aber was war denn der Grund für's größere F-PC?

Tom: Nachdem nun ZF veröffentlicht war, hatte ich einige Bedenken wegen des vorhandenen Platzes für die eigentliche Anwendung. ZF nutzt zwei Segmente, das eine für die Header und das andere für Daten, Code- und Colon-Definitionen. Und eben diese drei in einem Segment zusammen begrenzen halt deutlich den Programmumfang der compiliert und gestartet werden kann *(red.: Für einfache Adressierung ist ein 8086 System auf 64 kB große 'Segmente' begrenzt.)*.

Ich hätte übrigens schon vorher erwähnen sollen, daß *Robert Smith* und *Jerry Modrow* mit mir bei Maxtor arbeiteten. *Robert Smith* war wichtig für strukturelle Korrekturen im KERNEL und hat einiges in CODE umgeschrieben. *Jerry Modrow* wiederum machte sich nützlich, indem er das alles testete und Applicationen schrieb. Von ihm kam die Rückkopplung, damit hier und da Verbesserungen durchgeführt wurden. Und indem wir drei nun Appliaktionen für Maxtor erstellten, entwickelten wir die Idee, ein drittes Segment nur für Colon-Definitionen zu ergänzen. Unsere Tests zeigten, daß etwa die Hälfte einer Anwendung für Colon-Definitionen und die andere Hälfte für Daten verbraucht werden.

(red.: Wenn hier von Colon-Definitionen gesprochen wird, so sind die CFA-'Listen' gemeint (CodeFieldAdressen, ANS würde von Execution Token sprechen), die innerhalb einer Colon Definition compiliert sind, und eben auch dem 'Thread' folgend abgearbeitet werden.)

So implementierten wir ein Forth mit dem Namen F88 und das nutzte eben ein drittes Segment und zwar für die Colon Definitionen und ."-strings. Die mögliche Programmgröße war nun deutlich größer und das fast ganz ohne Einbuße an Schnelligkeit. Es war direct threaded, auf dem Retun Stack lag ein Element und zwar der Offset zu der Definition, die gerade interpretiert wurde. Es war leistungsfähig und ließ nun sinnvoll große Programme zu. Letztendlich aber hatten wir TestProgramme, die auch dafür zu groß waren.

VD: War das dann der Start zu F-PC mit jeweils zwei Elementen auf dem Retum Stack und einem beliebig großem List-Segment?

Tom: Nun ja, ich entschloß mich die Architektur nochmals zu ändern, eben zu dem was dann später einmal F-PC genannt werden sollte. Colon-Definitionen starten nun an einer Paragraphen -Grenze im List-Segment, was diesen Bereich um 25% vergrößerte und das GesamtProgramm um 10 bis 15 %. *

Wichtig war aber die maximale Programmgröße, die konnte nun auf das doppelte oder sogar eher das dreifache angehoben werden. Damit wird dann der Code-space zum limitierenden Faktor die Programmgröße.

VD: Tom, Danke Dir, wir haben nun eine Vorstellung von der vielgestaltigen Entwicklung von F-PC aus Laxen und Perry's F83 heraus. Aber wieso ist ein Forth, in dem so umfangreiche Applikationen möglich sind und zu dem es dazu so viele Hilfen und Beispiele gibt, Public Domain?

Tom: Nun, es gab verschiedene Gründe F-PC als Public Domain zu veröffentlichen. Einmal meine ich, daß ich keinen Support geben kann, zumindest nicht den, der für ein kommerzielles Produkt erforderlich wäre. Ein anderer Grund war mein Wunsch, daß auch ein solches Forth allgemein verfügbar sein sollte, bei dem den Leuten viele Tools zur Verfügung stehen. Und -- wie ich sagte -- ich verwendete vorher LMI-Forth, und das hatte keinen Debugger und Decomiler gehabt. Ich mußte meinen eigenen schreiben. Das sind eben Dinge, die ich in jedem Fall in F-PC integriert haben wollte. Und dann eben auch den sequentiellen TextEditor, das halte ich für ungemein wichtig, damit das auch von Programmierern allgemein akzeptiert werden kann.

Aber ich wollte auch den üblichen Standard vorwärtspuschen, wenn so was denn überhaupt möglich ist. So erzeugt F-PC doch Druck bei den kommerziellen Forth Vertreibern, eine komplette Werkzeugkiste zum Debuggen mitzuliefern. Und das wollte ich nun unbedingt in F-PC integriert haben. Einige -- warum auch immer - nehmen an, daß ein Debugger bei Forth nicht wirklich notwendig ist. Ich persönlich halte das für sehr sinnvoll.

Und wie sich F-PC so entwickelte kamen einige weitere Werkzeuge hinzu, und ich entwickelte auch

*) Anm. d. Red.: Bei den 8086 Processoren gibt es eine 20.087 breite Addressierung die physikalisch aber nur zu 20 bit genutzt werden kann. Sie setzt sich aus einer 16bit breiten 'normal'-Adresse und einer um 4bit verschobenen, aber auch nur 16bit breiten 'Segment'-Adresse zusammen. Die normale Adresse kann auch als Offset innerhalb dieses Segmentes verstanden werden. Der Adressbereich der Normaladdresse ist 64kB, der der Segmentadresse 1 MB, allerdings nicht in einer Stufung in Byte sondern eben in Paragraphen. 1 Paragraph enthält 16 Byte. Zur Addressierung innerhalb eines Paragraphen muß die Normaladdresse genutzt werden. Wird bzw. kann nur auf eine ParagraphenGrenze addressiert werden, so kann 1 MB trotz eines nur-16bit-Pointers abgedeckt werden.

```
para $0001 ist equivalent to addr $0010

 long            long     flat
seg:off  ist equivalent to  seg:off   20-bit
3FE8:0010    ----->     3FE9:0000   $03'FE90
3FE8:0011    ----->     3FE9:0001   $03'FE91
3FE8:0080    ----->     3FF0:0000   $03'FF00
3FE8:0180    ----->     4000:0000   $04'0000
FFFF:000F    ----->     FFFF:000F   $0F'FFFF  ( 20 bit )
FFFF:FFFF    ----->     FFFF:FFFF   $10'EEEF  ( 20.087 bit )
```

Beispiele, und so nahm es an Umfang stetig und stetig zu.

VD: Ja, Tom ich glaube schon, daß es interessant ist, die vielen Hilfsmittel zu haben und eben auch den kompletten Quelltext bis hinunter zum Meta Compiler. Das vermisse ich bei kommerziellen Systemen. Nun aber zu TCOM, dem Target Compiler; der ist ja aus F-PC hervorgegangen.

Tom: Nun, dazu wollte ich gerade kommen. Was mich nämlich zunehmend störte war, daß es immer schwieriger wurde, kleine Programme mit F-PC zu erzeugen. So wurde zwar F-PC letztendlich ein ausgefeiltes ProgrammEnwicklungsSystem für umfangreiche Applikationenen, aber eben nicht für kleine Sachen. So fing ich dann mit dem TCOM Compiler an. Ich wollte eben einen Compiler haben, der nur das erzeugte, was zum Ablauf des Programmes notwendig war. TCOM optimiert so Ihre Applikation. Darüberhinaus gibt es noch andere Optimierungen, die den Code verkleinern. TCOM nutzt einige klassische Optimierweisen, ich glaube das nennt man PeepHole Optimierung.

TCOM hat heute über 60 Beispielprogramme, Utilities für alle Bereiche, dazu Graphik und dann eben auch so einiges, das ich selbst gerne nutze. Und mitlerweise gibt es auch TSR , also terminate and stay resident. Du kannst ein Programm im Speicher belassen, und auf einen Knopfdruck hin kommt es hoch, egal was im Vordergrund gerade lief. Das hat etwas gebraucht, bis es nun da ist, ich glaube, nun aber mit einem gut nutzbaren Funktionsumfang.

VD: Tom, Du bist ja nicht mehr bei Maxtor, kannst aber auch bei dem jetzigen Arbeitgeber mit Forth arbeiten.

Tom: Ja, augenblicklich arbeite ich bei *Samsung Halbleiter* in San Jose, Kalifomien. Ich bin dort der Software-Manager. Wir erstellen Tools, die von AnwendungsIngenieuren genutzt werden, die Programme fürs ROM schreiben; für Samsung 4- und 8-bit µPs. Das wird genutzt für weiße und braune Ware, also Farbfernseher, VideoRecorder, Fernsteuerungen und anderes mehr, das so kleine µPs verwendet. Wir versorgen sie mit Forth Compilern und Entwicklungs-Debuggern. Es ist eine integrierte EntwicklungsUmgebung, in Forth geschrieben, mit einem wirklich ansprechenden Funktionsumfang und entsprechender Bedienung. Das ist recht beliebt bei unseren Kunden.

VD: Du sprichst auch von 4bit Controllern, geht das denn mit Forth? Wohl nicht mit 16bit breiten Zellen ?!

Tom: Das interessante, was ich bei *Samsung* entdeckte, ist, daß Forth eben nicht unbedingt für 16bit implementiert sein muß. Es kann eben auch als 8bit-System implementiert sein. So können sehr kleine und effiziente Programme generiert werden, die sehr schnell laufen. Selbst größere Sachen können so auf diesen kleinen µPs realisiert werden. Und dabei ergibt sich ein Programmiersprachen-Interface, das einfach zu nutzen ist; eben einfacher als ein Assembler auf diesem gleichen Controller.

VD: Nun ja, aber wie wird es weitergehen; auch in Zukunft Forth? Einmal persönlich und auch allgemein.

Tom: Und wenn ich in die Zukunft blicke ... . Ich persönlich -- wie wohl auch ohne Zweifel einige der Leser der *Vierten Dimension* -- kennen die Probleme die Forth nun mal hat, akzeptiert zu werden. Zumal wenn Kunden meinen, -C- sei der Weisheit letzter Schluß für alle Aufgaben. Ich habe da natürlich offensichtlich eine andere Meinung. Und ich glaube, zumal wenn noch einige Zeit verstreicht, daß doch mehr Leute, inklusive denen aus den Mangementbereich, begreifen werden, daß -C- so einige Probleme hat. Was wir tun können, um Forth weiterzubringen? Nun, wir sollten es nutzen, wann immer wir das können und dabei die Programme lesbar gestalten, leicht zu warten und gut dokumentiert. Gegen ein gutes Programm anzukommen, ist in jeder Programmiersprache hart. Es ist halt wichtig, daß wir unseren Job gut machen, daß unsere Arbeit bei den Managern gut ankommt, eben besonders wenn es Forth ist. Und immer sollte natürlich das Projekt fertiggestellt werden. Ein Abbruch vorher ist natürlich ein ernsthaftes Problem für jede Firma.

Ich selbst werde Forth in Zukunft weiter nutzen und sollte es nur zu hause möglich sein. Ich stehe auch unter dem Druck -C- einzusetzen, eben auch bei *Samsung*, wo ich ja arbeite. Ich werde es wohl auch für einige Aufgaben nutzen, andererseits widerstehe ich diesem Druck aber auch. Ich beweise, daß einige Probleme mit Forth sehr gut lösbar sind und daß ich sie eben auch über Forth sehr schnell lösen kann.

Es ist zu hoffen, daß die Entscheidungsträger ihre Entscheidung über die Programmiersprache mit Sachverstand und Logik angehen werden und nicht auf Hörensagen und Trends basierend.

Ich hoffe halt, daß alle ihre Leser weiterhin enthusiastisch Forth einsetzten und auch andere überzeugen, was da alles für Aufgabstellungen mit Forth gelöst werden können. Daß sie andere ermuntern, es mit Forth zu versuchen, wo wir doch schließlich festgestellt haben, daß Forth so ausgesprochen nützlich sein kann.

VD: Danke Tom, für das Interview und auch den aktivierenden Apell, Dank auch im Namen unserer Leser. Es wird sicherlich Interesse bestehen, einmal Hintergrund-Informationen zu F-PC und TCOM und den vielen vielen Quelltexten zu bekommen. Die wichtigen Ergänzungen, gerade aus der letzen Zeit, werden das Interesse nochmal erhöhen. Ganz toll ist natürlich -- wenn ich da auch persönlich Dir nochmal danken darf -- daß TSR jetzt mit TCOM einfach realisiert werden kann. Das war ja mehrfach Gegenstand unserer bisherigen Diskussionen.

Wir wünschen Dir und Deiner Frau eine sichere Heimreise und wünschen uns natürlich, daß Du nocheinmal über den großen Teich kommen wirst. Ich hoffe, daß Ihr ein wenig von Deutschland kennenlernen und schätzen konntet und die Reise auch forthmäßig ein Gewinn war.

Interviewparter der VD (Fahrer und Interpreter) war Arndt Klingelnberg. Das Gespräch verbrauchte 2.5 l Diesel.

Weitere Anmerkungen der Redaktion:
Aus dem 'Y' in F83Y ergab sich wohl die F-PC interne Bezeichnung des Header-Segmentes als 'Y'-segment zusammen mit den vereinfachenden Y-Operatoren für diesen Bereich. Die so völlig getrennten Header vereinfachen natürlich TURNKEY und HEADERLESS . Ich (akg) nutzte 'Y' schon immer gerne in 'transienten' Dateinamen als Indikator zum späteren wegwerfen und wünsche nun noch ein angenehmes 'BEHEAD'.

❑

Am F-PC verdienen... kps

...darf jeder, der F-PC als Basisprodukt verwendet und auch bereit ist, entsprechende Benutzerhilfen zu bieten! Das jedenfalls bestätigte Tom Zimmer, der Vater des F-PC!

Bei der Jahrestagung der Deutschen Forth Gesellschaft e.V. am 25. April 1993 in Nürnberg war als Gast auch *Tom Zimmer*, Autor des verbreiteten PD-Forth F-PC.

Neben den Verfechtern seines Original-F-PC taucht in der VD auch schon mal eine abgewandelte Version auf, die von den erwähnten Rechtgläubigen scharf angegriffen und an den Rand der Legalität gerückt wurde. Ich nahm deshalb in Nürnberg die Gelegenheit wahr, *Tom Zimmer* nach seiner Meinung zu fragen. Die wohlüberlegte Antwort fasse ich hier zusammen:

Tom Zimmer bedauert, daß er sein F-PC nicht professionell unterstützen kann. Wegen dieser fehlenden Unterstützung der Anwender ist es auch richtig, daß F-PC zu Selbstkosten verteilt wird. Sollten Benutzer an ihn Fragen zu einem von ihm selbst herausgegebenen Release richten, wird er nach Möglichkeit antworten. Wenn sich jemand die Mühe macht, Zusätze und Verbesserungen jeder Art anzubringen, dann freut er sich darüber. Derartige Versionen dürfen auch gerne verteilt werden, wenn dabei klar und deutlich auf folgendes hingewiesen wird:

- *Basis-Produkt:*
  F-PC, Version xxx, Tom Zimmer
- *Bearbeitung:*
  Autor mit Anschrift
- *Rückfragen:*
  NUR an den Bearbeiter

Falls der Bearbeiter seinen Kunden professionelle Unterstützung gewährt, so hat Tom Zimmer nichts dagegen einzuwenden, wenn der Bearbeiter ein kommerzielles Entgelt für seine Lieferung und Leistung verlangt und erhält.

Diese - wie ich meine - sehr vernünftige und faire Einstellung läßt keinen Raum für die erwähnte Auseinandersetzung, die nun hoffentlich endgültig der Vergangenheit angehört.

(Die Redaktion wird dieser Diskussion jedenfalls keinen kostbaren Raum mehr in der VD zubilligen! rk)

Forth-Gruppen regional

Berlin
Claus Vogt
Tel. 0+30 - 2 16 89 38 p
Treffen: nach Absprache

Rhein-Ruhr
Jörg Plewe
Tel. 0+208 - 49 70 68 p
Treffen: jeden 1. Samstag im Monat im S-Bahnhof Derendorf
Münsterstr. 199, Düsseldorf

Moers
Friederich Prinz
Tel. 0+2841 - 5 83 98 p
Treffen: jeden Samstag 14:00
Arbeitslosenzentrum, Donaustr. 1
Moers

Aachen
Klaus Schleisiek,
Tel. 0+241 - 87 34 62 gaf
Treffen: jeden 1. Montag im Monat als Gruppe des Computer-Club der RWTH, Seminargebäude,
Raum 214, Nähe Wüllnerstraße
Aachen

Darmstadt
Andreas Soeder
Tel. 0+6257 - 27 44

Mannheim
Thomas Prinz
Tel. 0+6271 - 28 30 p
Ewald Rieger
Tel. 0+6239 - 86 32 p
Treffen: jeden 1. Mittwoch im Monat, Vereinslokal Segelverein Mannheim e.V., Flugplatz Mannheim-Neuostheim

µP - Controller Verleih

Rafael Deliano
Steinbergstr. 37,
82110 Germering
Tel.:0+89 - 8 41 83 17

Gruppengründungen, Kontakte

Regional

Stuttgart
Wolf-Helge Neumann
Tel. 0+711- 8 87 26 38 p

Fachbezogen

8051 ... (Forth statt Basic,e-FORTH)
Thomas Prinz
Tel. 0+6271 - 28 30 p

Forth-Hilfe für Ratsuchende:

Forth allgemein

Jörg Plewe
Tel. 0+208 - 42 35 14 p
plewe@mpi-dortmund.mpg.de

Karl Schroer
Tel. 0+2845 - 2 89 51 p

Jörg Staben
Tel. 0+2103 - 24 06 09 p

Spezielle Fachgebiete

Anfänger und Wiedereinsteiger
Gerd Limbach
Tel. 0+2051 - 25 51 12 p
Mo. + Di. 20:00 - 22:00

32FORTH (Atari)
Rainer Aumiller
Tel. 0+89 - 6 70 83 55 gp

FORTHchips (FRP1600, RTX, Novix ...)
Klaus Schleisiek-Kern
Tel. 0+40 - 2 20 25 39 gp

F-PC & tCOM, ASYST (Meßtechnik), embedded controller (H8/5xx//TDS2020, 8051 ... eFORTH...)
Arndt Klingelnberg
Tel. 0+2404 - 6 16 48 agp

Gleitkomma-Arithmetik
Andreas Döring
Tel. 0+721- 59 39 35 p

HS/Forth (Harvard Softworks)
Wigand Gawenda
Tel. 0+30- 44 69 41 p

KI (Künstliche Intelligenz), OOF (Object Oriented Forth)
Ulrich Hoffmann
Tel. 0+431 - 80 12 14 p

Unterricht mit FORTH
Rolf Kretzschmar
Tel./Fax 0+2401 - 8 88 91 ap

UUCP (FORTH ... per eMAIL)
Andreas Jennen
Tel. 0+30 - 3 96 52 27 ap

volksFORTH/ultraFORTH/RTX-FG-Forth/Super8Forth
Klaus Kohl,
Tel. 0+8233 - 3 05 24 p
Fax. 0+8233 - 99 71 f

Forth-Vertrieb

volksFORTH (PC, ST, C64) / F68K (68000) / ...
Johannes Teich
Ettaler-Mandl-Weg 19
82418 Murnau
Tel. 0+8841 - 29 43
jgt@bbs.forth-ev.de

Forum: Forth-Mailbox

Forth-Mailbox
Jens Wilke (SysOp)
Tel. 0+89 - 8 71 43 52 p
Mailbox 0+89 - 8 71 45 48,
300-2400 baud (8N1)

Hinweise

Zu den Telefonnummern

f == FAX
a == Anrufbeantworter, hier können Sie Ihren Ansprechpartner eventuell vorinformieren, erwarten Sie bitte keinen (kostspieligen) Rückruf
g == geschäftlich, zu erreichen innerhalb typischer Arbeitszeiten
p == privat, zu erreichen außerhalb typischer Arbeitszeiten

Die Adressen des Forth e.V. (Forth Büro) und der Redaktion / Anzeigenverwaltung finden Sie im Impressum.

Ein Grundprinzip leitet die Entwicklung von Forth und leitet weiterhin die Anwendungen damit, schlichtweg:

Halte es einfach!
(Keep it simple)

Eine einfache Lösung ist elegant. Die elegante Lösung ist das Ergebnis einer gründlichen Anstrengung, das wirkliche Problem zu verstehen und überzeugt durch das beeindruckende Gefühl der Richtigkeit.
Ich betone diesen Punkt besonders, weil er im Gegensatz zur vorherrschenden Meinung steht, die Leistungsfähigkeit steige mit der Kompliziertheit an.
Einfachheit erzeugt Vertrauen, Zuverlässigkeit, Überschaubarkeit und Schnelligkeit.

Chuck Moore auf der Forth Conference UoFR 1990 eingesandt und übersetzt von W. Allinger, Solingen

FORTH-SYSTEME GMBH

Postfach 1103,
7814 Breisach

Telefon (0 76 67) 5 51
Telefax (0 76 67) 5 55

Telefon Schweiz:
(055) 53 65 55

UR/FORTH

- FORTH-83 Standard
- Für MS-DOS, OS/2, 80386
- Direkt gefädelte Code Implentationen mit dem obersten Stackwert im Register um größtmögliche Ausführungsgeschwindigkeit zu erreichen
- Segmentiertes Speichermodell mit Programm, Daten, Headers und Dictionary Hash Table jeweils in einem getrennten Segment
- Komplett gehashtes Dictionary führt zu extrem schneller Übersetzung
- Mächtige neue String Operatoren (Suche, Extraktion, Vergleich und Addition) sowie einen dynamischen String-, Speichermanager
- Kann mit Objektmodulen, die in Assembler oder anderen Hochsprachen erzeugt wurden, gelinkt werden
- Native Code Optimizer zur direkten Umsetzung in 80 x 86 Code im Lieferumfang

FORTH-83 Metacompiler

Der LMI FORTH Metacompiler wird mit komplettem Quellcode für ein ausführlich ausgetestetes, Hochgeschwindigkeits FORTH 83 Kern ausgeliefert, wobei Sie die Auswahl aus folgenden Zielprozessoren haben:

- 8086/8088
- Z80/HD64180
- 8080/8085
- 68000
- Z8
- 1802
- 6809
- 8096/97
- 78310
- 8031/32/535
- 6303
- 6502
- V25
- 68HC11
- RTX 2000
- 80C166

Sie erzeugen schnelle und kompakte Anwendungen, indem Sie Ihre Quellprogramme mit unserem Forth Nucleus zusammenstellen und ihn mit dem LMI FORTH Metacompiler übersetzen.

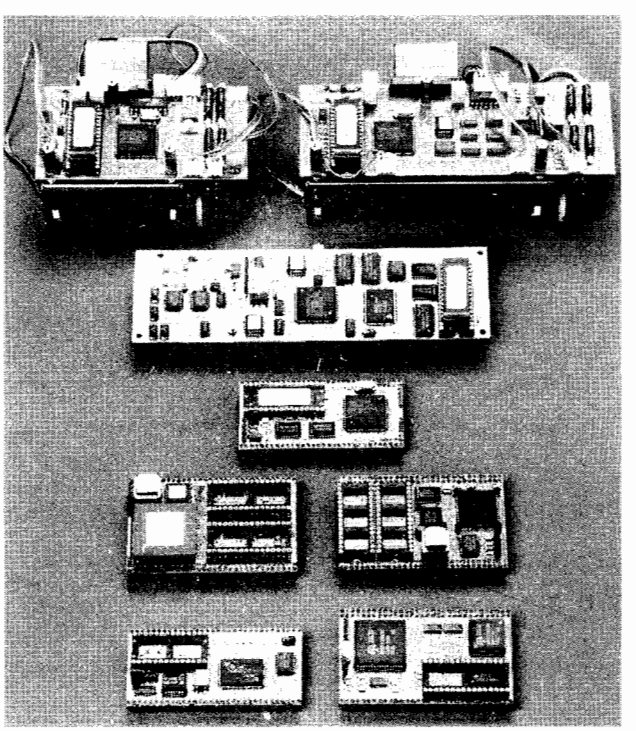
WinFORTH

- UR/FORTH kompatibel
- Windows Funktionen werden voll unterstützt
- Erweiterte Debugging-Hilfsmittel
- Online Windows Hilfe
- Coprozessor Unterstützung möglich
- Software-Gleitkomma-Paket
- Viele Beispielprogramme
- Upgrades von UR/FORTH Systemen auf WinFORTH sind preisgünstig zu erhalten

SRS II

- Serieller ROM Emulator
- Unterstützung folgender Bausteine:
  27256, 27512, 271000, 27010, 27020, 27040
- Minimale Zugriffszeit 100 ns
- Maximale Baudrate 115.200 bits/s
- Highspeed Interface als Option
- Gleichzeitiger Zugriff von Host und Zielprozessor
- Zusätzliche serielle Schnittstelle über den ROM-Sockel
- Intel-Hex, Motorola-S oder ASCII/binär Formate werden unterstützt
- Der SRS II ist nur 157 x 94 x 36 mm groß
- SRS63 kompatibel

ModuNORM



CPU-Steck-Module im Scheckkartenformat:

- 8 Bit z.B. 6303
- 16 Bit z.B. V25
- Highspeed RTX-2000/1
- 80C166
- Softwareunterstützung durch SwissFORTH
- Thermodrucker und Controller
- LCD Grafik-Controller

Bitte fordern Sie unseren Produktkatalog und die Preisliste an. FORTH-Gesellschaftsmitglieder erhalten bis zu 10% Rabatt (artikelabhängig).