Volume IV/Nr.2/3 - September 1988

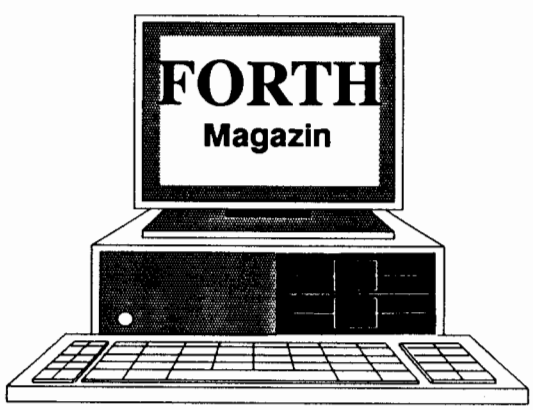
VIERTE DIMENSION

Doppelheft

THEMEN

- FORTH-Gesellschaft intern
- SWOPPERATOREN
- Interview mit Martin Tracy
- Hardware-Realisierung von FORTH

Einzelpreis DM 10,00



Auch im Original leere Seite

EDITORIAL

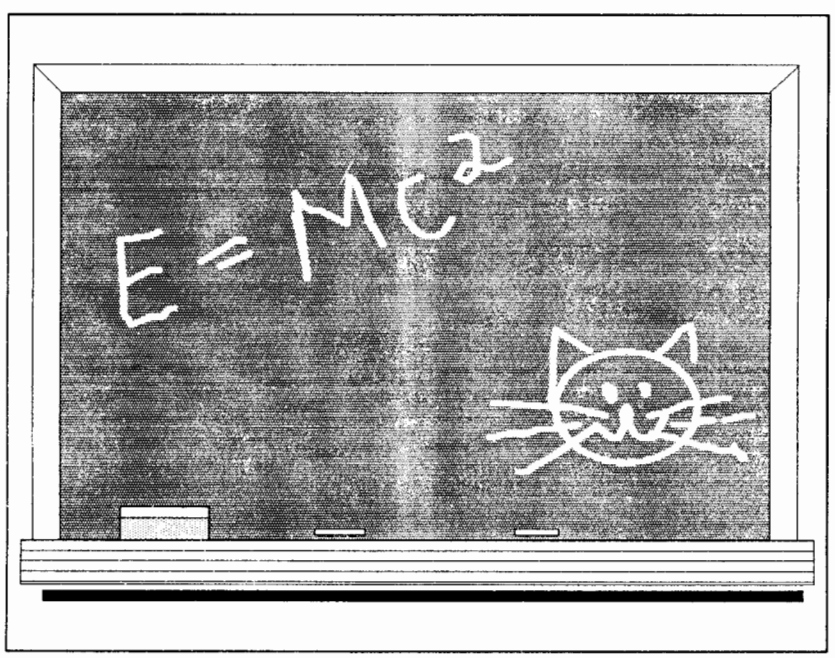
Endlich ist es soweit, nach einer langen Pause liegt nun wieder eine 'Vierte Dimension' vor Ihnen. Aber das Warten hat sich gelohnt. Wir beginnen gleich mit einer Doppelnummer. Das heißt doppelt soviele Seiten, doppelt soviel Information, Spaß usw.

Wie Sie vielleicht schon aus dem Rundbrief (kurze Mitgliederinformation) erfahren haben, den Sie vor noch nicht allzu langer Zeit erhalten haben, liegt die Redaktion nicht mehr in den Händen von Michael Kalus, dem wir an dieser Stelle herzlich für die gute Zusammenarbeit danken möchten, sondern bei uns. Wir sind eine kleine Software-Firma, die sich u.a. mit FORTH beschäftigt. So stammt z.B. der 32FORTH Compiler der im M&T Verlag erschienen ist, von uns. Wir hoffen, daß wir die 'Vierte Dimension' zu Ihrer Zufriedenheit gestalten werden. Sparen Sie nicht mit Kritik, Anregungen und Beiträgen.

In diesem Heft finden Sie u.a. einen Beitrag über 'Peerless' Namen in FORTH, ein Interview mit Martin Tracy, einen Artikel über alternative Stackoperatoren, Firmeninformationen über den neuen Harris-Chip sowie über den MARC4 von EUROSIL.

So, und nun viel Spaß beim Lesen wünscht die Redaktion

Rainer Aumiller und Denise Luda



IMPRESSUM

Titel:
FORTH MAGAZIN 'Vierte Dimension'
Zeitschrift der Mitglieder der FORTH-Gesellschaft e.V.

Herausgeber:
FORTH-Gesellschaft e.V.

Redaktion:
D. Luda Software, Staudingerstr. 65, 8000 München 83, Tel. 089/670 83 55

Kontaktadresse:
Entweder direkt die Redaktion anrufen bzw. anschreiben oder das FORTH-Büro in München, Postfach 1110, 8044 Unterschleißheim kontaktieren.

Autoren dieser Ausgabe:
Rolf Kretzschmar, Mick Ham, Thomas Jung, Andreas Soeder, Andreas Goppold, Jeffrey R. Teza, R. Jones, D. Maier, Gert-Ulrich Vack, J. Staben. Übersetzungen: D. Luda.

Erscheinungsweise:
Vierteljährlich

Redaktionsschluß:
Der mittlere Quartalsmonat

Auflage:
500 Stück

Druck:
Druckservice Ost GmbH, Pfanzeltplatz 4, 8 München 83

Bezugspreis:
Einzelheft DM 5,-

Vierte Dimension

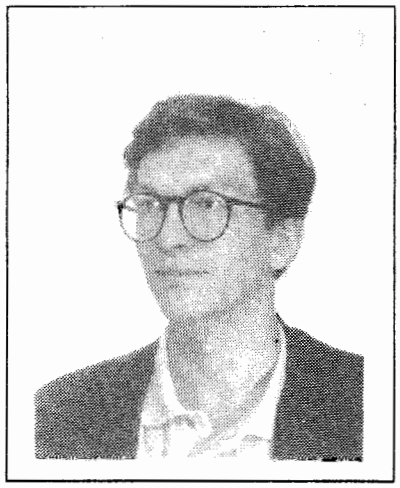
Inhalt

Inhalt

FORTH-Gesellschaft e.V. intern

Da sich die Übergabe der Geschäfte des Direktoriums von Hamburg nach München und die Übergabe der Redaktion der 'Vierte Dimension' verzögert hat, erscheint diese 'Vierte Dimension' nun erst im September 1988 als Doppelheft.

Das neue Direktorium stellt sich vor

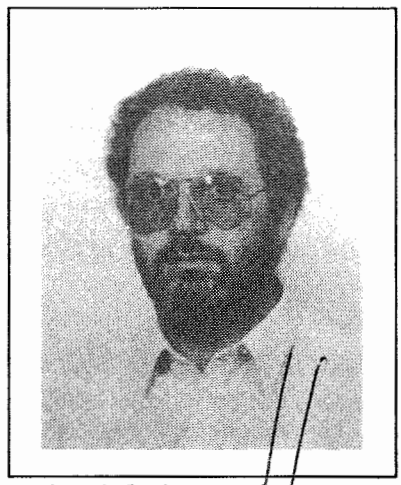


Christoph Krinninger, Jahrgang 1961

Ich kam 1961 zur Welt und befinde mich seit meiner frühen Jugend im Kampf mit Chips und Lötkolben. Zur Zeit studiere ich in München Zahnmedizin und bin auf dem besten Weg, ein guter Zahnarzt zu werden. Ich bin also ein typischer Vertreter des Hobbyisten und nicht-professionellen Anwenders. Meine unmittelbaren Interessen in Sachen FORTH gelten überwiegend Medizin bezogenen Themen, wie z.B. Künstliche Intelligenz, Expertensysteme, Bildverarbeitung, CAD/CAM. FORTH ist meine ideale Programmierumgebung seit ca. 3 Jahren. Meine nächsten Arbeiten mit FORTH betreffen ein Dialogvideo-System für Zahnmediziner, sowie ein Fileinterface für das RTX-2000 Board. Privat arbeite ich überwiegend mit MACH2, einem 32-Bit FORTH für den Macintosh, das ich auf einem Atari ST mit dem Macintosh Emulator "Aladin" benutze.

Mitgliedschaft in der Anwendergemeinschaft 68000-Systeme (AGS).

Hobbys: FORTH, Musizieren, Steppen.



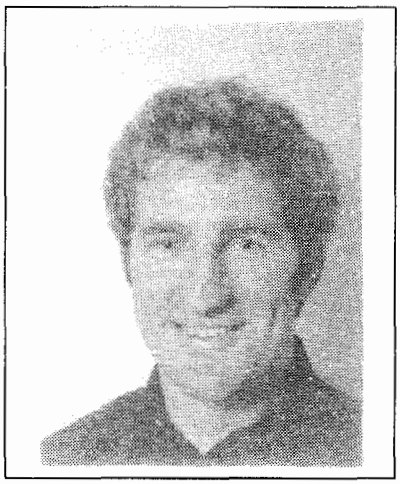
Heinz Schnitter, Jahrgang 1944

Nach meiner Ausbildung an der Flugsicherungsschule der Bundesanstalt für Flugsicherung war ich als Bundesbeamter bis Ende 1968 am Flughafen Stuttgart Echterdingen beschäftigt. Während eines zweijährigen Zwischenspiels bei einer Computerfirma hatte ich die ersten Kontakte mit Computern und Programmiersprachen. 1970 wechselte ich an das Beschleunigerlabor der Universität und Technischen Universität München. Dort bin ich zuständig für die Leitung des Elektroniklabors.

Die ersten Kontakte mit FORTH hatte ich 1984 als Juror beim bayerischen Landeswettbewerb "Jugend forscht". Am Beschleunigerlabor wird FORTH als Programmierumgebung zur Steuerung und Regelung der Beschleunigeranlage mit Strahlführungssystem eingesetzt. Dadurch bin ich in der glücklichen Lage, meinen Lebensunterhalt - unter anderem - auch mit Programmieren in FORTH verdienen zu können.

Mitgliedschaft in der Forth Interest Group (FIG).

Hobbys: FORTH, Hochseesegeln, Wandern



Johannes Reilhofer,
Jahrgang 1938

Studium E-Technik an der TH München Fachrichtung Nachrichtentechnik. 4 1/2 Jahre Siemens München Zentrallabor Impulsmodulationstechnik. 2 Jahre Entwicklungsleiter der Kyberna GmbH.

1969 Gründung der Johne + Reilhofer GmbH. 18 Jahre Geschäftsführer von J+R. Themen: Vielkanalmeßanlagen für die Erprobung von Fahrzeugen, Bergbaumaschinen, Kraftwerkskomponenten.

1988 Gründung der Reilhofer KG, pünktlich zum 50ten Geburtstag. Themen: Was bis jetzt nur Off-line ging auf On-line umstellen. Das im Bereich Maschinenbau und mit FORTH.

Hobbys: FORTH, Reisen

Allgemeines

Das Direktorium bedankt sich für das ihr auf der Jahreshauptversammlung entgegengebrachte Vertrauen.

Unser besonderer Dank gilt dem scheidenden Direktorium, das die FORTH-Gesellschaft während der Gründerjahre mit viel persönlichem Engagement vertreten hat.

Wichtig für den Erfolg der FORTH-Gesellschaft war sicherlich auch das regelmäßige Erscheinen der 'Vierten Dimension'. Michael Kalus hat als Editor mit jedem neuen Exemplar der 'Vierte Dimension' bewiesen, daß die FORTH-Gesellschaft für alle FORTH-Programmierer die richtige Vereinigung ist.

Nachdem Michael Kalus aus beruflichen Gründen die Redaktion der 'Vierte Dimension' abgeben mußte, konnte Herr Rainer Aumiller für diese Aufgabe gewonnen werden. Herr Aumiller hat sich als Autor des 32FORTH bereits einen Namen gemacht.

Jahrestreffen

Das Jahrestreffen in München war ein großer Erfolg.

Hier einige Zahlen:

- 52 Teilnehmer
- 4 davon Firmen mit Hardware-Ausstellung
- 11 Vorträge
- 4 Referate

Die Vorträge und Referate werden in loser Folge in der 'Vierte Dimension' erscheinen.

Fachgruppe RTX-2000

Am 22.06.88 hat sich die Fachgruppe RTX-2000 in München konstituiert. Koordinator der Fachgruppe RTX-2000 ist Herr Max Diez, HARRIS Semiconductor GmbH Postfach 1232 D-8057 Eching, Tel.: 089/31900536.

Dankenswerterweise hat die Firma Harris der Fachgruppe einen RTX-2000 Chip und 35 ns SRAMs zur Verfügung gestellt. Das Ziel der Gruppe ist es, innerhalb kurzer Zeit eine Platine mit FORTH im EPROM zu entwickeln. Der Vertrieb dieser Platine soll über (interessierte) Firmen erfolgen. Für Mitglieder der FORTH-Gesellschaft ist sie zu einem ermäßigten Preis erhältlich.

Die Mitglieder der Fachgruppe arbeiten an diesem Projekt unentgeltlich. Mehr über dieses Projekt in einer der nächsten 'Vierten Dimensionen'.

LESERBRIEFE und Zuschriften

MVP-FORTH Integer, and Floating Point Math

In der 'Vierten Dimension' 1/88 lese ich, daß ihr das Buch von Koopmann "MVP-FORTH Integer, and Floating Point Math" sucht.

Ich habe es mir im vorigen Jahr aus den USA bestellt. Es ist nicht gerade billig und besteht eigentlich nur aus einer Anzahl von "fliegenden Blättern" (ca. 300). Vom Inhalt jedoch war ich sehr positiv überrascht: Es wird der Quelltext zu einem sehr umfangreichen Fließkommapaket und zu einem Paket für 64-Bit-Integerzahlen geliefert. Das Ganze ist ausführlich dokumentiert. Zusätzlich enthält das Buch Assemblerlistings für die Prozessoren 6502, 8080 und 8088, um vertretbare Ausführungsgeschwindigkeiten zu erreichen.

Ich habe den gesamten Sourcecode auf volksFORTH83 für den Atari ST umgearbeitet und benutze das Fließkommapaket jetzt fleißig.

Leider fehlte mir bisher die Zeit, alle Worte, die es nötig haben, in 68000er Assembler zu konzipieren. Dadurch sind die Ausführungsgeschwindigkeiten nicht gerade berauschend, aber akzeptabel.

Wenn Interesse am Buch oder am Code besteht, bitte meldet Euch bei mir!!

Thomas Jung
Carl-Goerdelerstr. 3
6500 Mainz

Turbo-FORTH

Lieber Michael Kalus,

auf Ihren Bericht über Turbo-FORTH im FORTH-Magazin 3/88 hin habe ich mich an Marc Petremann von JEDI gewandt und jetzt die erste Diskette bekommen. (Die beiden anderen habe ich auch bestellt.)

Ich bin begeistert, da das System voll auf die DOS-Organisation meines AT-Kompatiblen abfährt. Ich kann mir vorstellen, daß auf einige unserer FORTH-Freunde das für uns ungewohnte Französisch befremdlich wirkt. Ich selbst jedenfalls habe keine Lust, beim Arbeiten jedesmal nachdenken zu müssen.

Daher habe ich jetzt, nur mal so zum Spaß, mit der Eindeutschung von Turbo-FORTH begonnen. Etwa 10% sind schon geschafft. Als Referenz greife ich hauptsächlich auf die Bücher von Zech über F83 zurück.

Besteht von Seiten der FORTH-Gesellschaft ein offizielles Interesse zum Nutzen aller Mitglieder an diesem Unterfangen? An Geld bin ich nicht interessiert. Und jeder sonstwie geartete Anerkennungsvorschlag soll mir recht sein.

Ein Schreiben ähnlichen Inhalts werde ich an Marc Petremann richten. Denn da sind ja die Besitzrechte von JEDI zu beachten. Und eine Eindeutschung ohne Auftrag überträgt mir ja sicherlich keinerlei Rechte. (?)

Zwei Bugs in Turbo-FORTH:

- **E:** ist in FORTH.VOC nicht dokumentiert (also über HELP nicht abrufbar), arbeitet aber einwandfrei. (E: ist bei mir die RAM-Disk; D: ist A: als 720-K-Laufwerk.)
- **DRV** enthält, entgegen der Angabe in FORTH.VOC, nicht 1=A:, 2=B: etc., sondern 0=A:, 1=B:, ..., 4=E: .

Beziehen sich diese Erkenntnisse vielleicht nur auf meine Anlage?

Fred Behringer, München

JEDI-Mailbox

Die Association JEDI bietet ab sofort allen FORTH-Programmierern einen Mailboxservice an. Die Mailbox ist unter folgendem Code zu erreichen:

1. Innerhalb Frankreichs

3515 SAM*JEDI

2. Aus dem Ausland

(33) 36.43.15.15 SAM*JEDI

Bis zu 32 Anrufe können gleichzeitig angenommen werden. Das Übertragungsprotokoll entspricht der Norm von VIDEOTEX 1200/75 Baud, 7 Bits, gerade Parität, 1 Stoppbit (1200 Baud in Richtung Anbieter - Anwender und 75 Baud in Richtung Anwender - Anbieter).

Im Augenblick werden folgende Möglichkeiten angeboten:

- Austausch von FORTH-, PROLOG- und PASCAL-Programmen, sowie aller Programme, welche im Magazin JEDI erscheinen.
- ein FORUM; dort können Fragen von allgemeinem Interesse aus dem Microcomputerbereich gestellt werden, diese Fragen beantwortet, sowie die Antworten zu diesen Fragen gelesen werden.

- Briefkasten: Dort können vertrauliche Mitteilungen abgelegt und empfangen werden.

Zur BRD besteht eine provisorische Verbindung. Und zwar zwischen BTX und TELETEL. Erkundigen Sie sich bei der für BTX zuständigen Postabteilung über die Möglichkeit mit TELETEL zu kommunizieren. Sie können auch SAM*JEDI konsultieren und dort weitere Informationen erhalten.

Paris 06.07.1988

Association JEDI
17, rue de la Lancette
75012 Paris

Hinweise für Autoren

Auch in Zukunft möchten wir Beiträge veröffentlichen, die Sie uns hoffentlich in großer Zahl liefern werden. Schicken Sie Ihre Manuskripte bitte an die Redaktion der 'Vierten Dimension' D.LUDA Software Staudingerstr. 65, 8000 München 83, Tel. 089/6708355.

Am liebsten hätten wir die Manuskripte auf einer 5 1/4" Diskette im IBM-Format. Ist Ihnen das nicht möglich verwenden Sie bitte eine Schreibmaschine mit normalgroßer, nicht zu dünner Schrift. Das Farbband sollte möglichst frisch sein. Schreiben Sie nicht breiter als 80 Spalten cpi und nicht länger als 72 Zeilen pro DIN A4 Seite. Die Arbeiten sollten in dieser Reihenfolge enthalten:

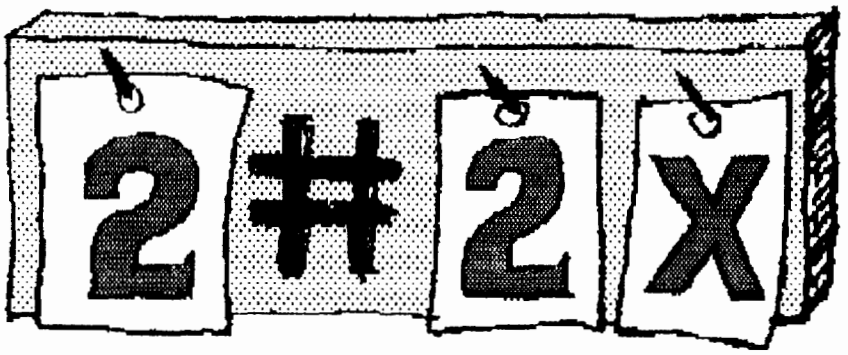
- Kurzer Titel,
- Autor,
- Zusammenfassung (ca. 50 Worte),
- Schlüsselworte (ca. 5), Text,
- Quellenangaben,
- Illustrationen,
- Tabellen,
- Quellcode.

Die Beiträge werden überarbeitet. Falls ein ausführliches Lektorieren erforderlich ist, erhalten Sie vor der Wiedergabe den Beitrag zur Korrektur und Zustimmung zurück. Layouts werden nicht mehr zur Prüfung durch die Autoren vorgelegt. Autoren erhalten auf Wunsch ein kostenloses Exemplar der 'Vierten Dimension' mit ihrem Artikel.

SWOPPERATOREN

Stackoperatoren mit System

Rolf Kretzschmar



Einblick

Es wird eine Methode beschrieben, wie man die üblichen und weitere mögliche Stackoperatoren als Worte mit wenigen Elementen in charakteristischer Anordnung bilden kann. Damit läßt sich u.U. das Fehlen *lokaler Parameter* leichter verschmerzen.

Einleitung

FORTH zählt zu den Computersprachen, die für den Informatikunterricht an allen Schulen grundsätzlich geeignet sind. Die Tatsache, daß der Einsatz von FORTH in diesem Bereich bisher bedeutungslos geblieben ist, liegt vermutlich an der fehlenden FORTH-Ausbildung für Lehrer. Nur wer - wie ich - auf anderem Weg auf FORTH gestoßen ist und die unglaublichen Möglichkeiten für den Einsatz im Unterricht erkannt hat, wird konsequenter Weise FORTH in allen Bereichen der informations- und kommunikationstechnischen Grundbildung einsetzen wollen.

Seit der Unterricht am Computer Einzug in den obligatorischen Fächerkanon gefunden hat und der/die Informatiklehrer(in) nicht mehr - wie zu Zeiten der Computer-AGs - nur die hochmotivierten Computerfreunde vor sich hat, muß er/sie sich schon etwas einfallen lassen, um dem interessierten und intelligenten Schüler auf der einen Seite der Skala, als auch dem uninteressierten und gleichzeitig leistungsschwachen Schüler am anderen Ende der Skala gerecht zu werden.

Mein Anliegen ist es, exemplarisch an einem Detail zu beweisen, wie leicht es sein kann, FORTH den unterrichtlichen Belangen anzupassen.

Problem

Schwache Schüler haben u.a. Mühe sich die Bedeutung der in FORTH wichtigen Stapeloperatoren zu merken. Für jemanden der kein Englisch kann, ist "swap" keineswegs aussagekräftig. Es widerspricht aber meinem Verständnis von FORTH, den Schülern zu raten, ständig die Liste der Stapeloperatoren mit entsprechenden Erläuterungen griffbereit zu halten.

Als Problemstellung ergab sich demnach für mich: Wie kann ich den Schülern ohne viel Zeitaufwand die Stapeloperatoren so beibringen, daß sie weder deren Namen, noch deren Wirkungen für die Dauer des Kurses vergessen?

Lösung

Sollen die neuen Worte aussagekräftig sein, müssen sie erkennen lassen, a) WAS sie machen, b) mit WELCHEM(N)

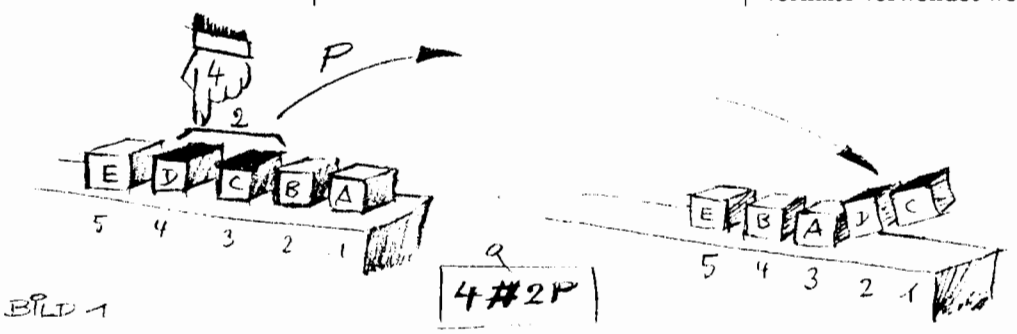
(Kreberenar)

Wert(en) des Stapels sie etwas machen und C) WOHIN sie diese(n) ggf. befördern.

Zusammengefaßt besteht ein SWOPPERATOR also mindestens aus einer Ziffer und aus einem Buchstaben K, X und P.

lesen im Listing sicher schwer gefallen. Zudem können Worte wie *löschen*, *kopiere* und *picken* nun für problemgebundene Sachverhalte verwendet werden.

10



zu a)

Das WAS läßt sich mit drei Buchstaben erledigen:

- mit **K** für Kopieren,
- mit **X** für Löschen (Symbol für Durchstreichen) und
- mit **P** für Picken (im Sinne von *Aufpicken*)

(Anmerkung: Lieber hätte ich statt P ein M (*move*) genommen, doch leider fällt mir dazu kein passendes deutsches Wort ein.)

zu b)

WELCHER Wert gemeint ist, kann einfach durch die Platzziffer (TOS ist der Platz Nr.1!) gekennzeichnet werden. Sind mehrere Werte gemeint, wird durch eine weitere Zahl die Menge der zu behandelnden Werte angegeben.

zu c)

Das WOHIN liegt auf der Hand: eine Stapeloperation hat in der Regel zum Ziel, Werte zur weiteren Bearbeitung bereitzustellen. Der Platz, der in FORTH zur Bereitstellung von Werten vorgesehen ist, ist der Platz Nr. 1 des Stapels, der TOS. Somit ist es sehr naheliegend, daß ein kopierter oder herausgepickter Wert dort landet. Mit dieser Vereinbarung entfällt die explizite Zielangabe.

Sollen mehrere Werte kopiert oder herauskopiert werden, landet das ganze Paket am Anfang des Stapelspeichers (Bild 1).

Der erste FORTH-Didatiker, Leo Brodie, nennt in "Thinking FORTH" die DUPs und SWAPs in FORTH-Programmen "noisy words". Gemeint ist das "Rauschen", von dem sich die eigentliche Information abhebt. (Unter Umkehrung der Bedeutung wird leider in der deutschen Übersetzung von '"lauten" Worten' gesprochen!! (Brodie 1986)). Wenn dieses "Rauschen" schon nicht beseitigt werden kann, so sollte man doch dafür sorgen, daß es leicht auszufiltern ist. Im Fall der SWOPPERATOREN (engl.: swapperators, oder kurz: swappers) glaube ich die Lösung in einer speziellen Art der *Hervorhebung* zu sehen: so sind alle SWOPPERATOREN durch das "#" zwischen dem Zeiger (Ziffer) und dem Operator (Buchstabe) *auffällig* markiert.

Beispiele:

1#X	(lies: ersten Wert löschen)
2#P	(lies: zweiten Wert picken)
2#K	(lies: zweiten Wert kopieren)

Ähnlich einem markanten Geräusch, das man nach einiger Zeit der Gewöhnung schlicht überhört, erwarte ich, daß es beim schnellen Durchsehen eines Programmes leichter gelingt, das Rauschen der Stapeloperatoren zu übergehen. Hätte ich versucht, den englischen Begriffen sinnvolle deutsche Bezeichnungen zu geben, wäre das wichtige Über-

Die Gleichartigkeit der Hervorhebung wird auch durch die Mengenziffer hinter dem "#" nicht gestört.

Beispiele:

2#2X	(lies: ab zwei zwei löschen)
3#2P	(lies: ab drei zwei picken)

(Weitere Beispiele und Übersetzungen sind im Anhang zu finden.)

Tauglichkeit

Meine Theorie bezüglich der leichteren Erlernbarkeit und der gesteigerten Behaltensleistung konnte im praktischen Einsatz bei über sechzig Schülern mit unterschiedlichen Lernvoraussetzungen gestützt werden. So wurden z.B. in entsprechenden Tests sowohl bei der Anwendung vorhandener, als auch bei der Benennung neuer Swopperatoren unerwartet wenig Fehler gemacht. Folge: mein "Meßinstrument" zur Leistungsbeurteilung (auch Klassenarbeit genannt) ging an den oberen Anschlag (sprich: die Arbeit war zu leicht). (Zum Nachvollziehen siehe original Aufgabenblatt)

Ausblick

An die SWOPPERATOREN kann man sich schnell gewöhnen. So fallen mir inzwischen bei der Suche nach passenden Stapeloperationen schneller die passenden SWOPPERATOREN ein, als die passende Folge von swaps, rots oder rolls. Was mich dann regelmäßig stört, ist, daß mein volksFORTH einen SWOPPERATOR wie z.B. 3#2X nicht sofort akzeptiert. Gerne würde ich ihm - wie meinen Schülern - die entsprechenden Regeln beibringen. Es müßte doch möglich sein, FORTH so *intelligent* zu machen, daß es ohne Zeitstrafe solche Worte richtig kompiliert; oder...?

Als Übergangslösung (wer füllt die Lücke?) habe ich N#X, N#K und N#P generiert, die für N einen entsprechenden Wert vom Stapel benötigen.

Auch für beliebige Operationen ist eine Lösung denkbar. So könnte man übereinkommen, daß zwei Ziffern vor dem "#" den Ort und die Anzahl der gewünschten Werte angeben und daß die eine Ziffer hinter "#" den Zielort kennzeichnet.

Beispiel:

52#3P (A B c d e - c A B d e)

Abschluß

FORTH zu verändern ist *nicht* meine Absicht. Neben der Vorstellung einer Idee geht es mir um den Beweis der Schultauglichkeit von FORTH. Zeigt doch die Tatsache, daß man sich als Lehrer(in) eine Programmiersprache entsprechend den didaktischen Forderungen für seinen Unterricht einrichten kann, welche Möglichkeiten aus Unkenntnis ungenutzt bleiben.

Die etablierten Sprachen nimmt man, wie sie sind; mit FORTH schafft man sich die Lern- und Lehrumgebung, die man benötigt.

Literatur:

Brodie, L. (1986). In FORTH denken. München: Hanser.

Goppold, A. (1984). Entwicklung einer orthogonalen Syntax für die Sprache FORTH. In: Vierte Dimension. Vol.1/Nr.1. 3-4.

Hoemann, D. (1986). A Universal Stack Word. In: FORTH Dimensions. Vol. VII, Nr. 5. 25-26.


```
Scr 1 Dr 2 SWAPERS.SCR
0   \          Swopperatoren
1
2   ' drop     Alias     1#X      \ (1)sten Wert(#) löschen(X)
3   ' dup      Alias     1#K      \ (1)sten Wert(#) (K)opieren
4   ' nip      Alias     2#X
5   ' swap     Alias     2#P      \ (2)ten Wert(#) heraus(p)icken
6   ' over     Alias     2#K
7   ' 2drop    Alias     2#2X     ' 2#2X Alias 1##X
8   ' 2dup     Alias     2#2K     ' 2#2K Alias 1##K
9   ' rot      Alias     3#P
10  ' -rot     Alias     3#2P
11  ' 2swap    Alias     4#2P     ' 4#2P Alias 2##P
12  ' 2over    Alias     4#2K     ' 4#2K Alias 2##K
13  ' pick     Alias     n#K      \ ... keine Regel ohne Ausnahme
```

SWOPPERATOREN – TEST

(Auszug aus dem Original)

Welche Stapeloperatoren wurden verwendet? (Nur vorhandene Operatoren!)

Stapel vorher	verwendete Stapeloperatoren	Stapel nachher
10 20 30		10 20 30 30
10 20 30		20 30 10
10 20 30		10 20 30 20 30
10 20 30		10 30 20
10 20 30		10 20 30 20
10 20 30 40		30 10 20
10 20 30 40		30 40 10 20
10 20 30 40		10 20 30 40 10 20
10 20 30 40		10 20 20
10 20 30 40		10 40 20
TOS		TOS

Welche Namen haben die Operatoren, die die folgenden Ergebnisse liefern?

Vorhandene Operatoren:

1#X	1#K
2#P	2#K
2#2X	2#2K
3#P	3#2P
4#2P	4#2K

Stapel vorher	Name:	Stapel nachher
10 20 30		20 30
10 20 30		30
10 20 30		10 20 30 10 20
10 20 30		10 30
10 20 30		10 20 30 10 20 30
10 20 30 40		10 20 30 40 20
10 20 30 40		30 40
10 20 30 40		40
10 20 30 40		40 10 20 30
10 20 30 40		10 20 30 40 10 20 30
TOS		TOS

Aufgabenblatt

FORTH-Profil(e): Martin Tracy

Mick Ham

Übersetzung aus 'FORTH Dimensions' Vol IX,#6: D. Luda

Martin Tracy gehört schon viele Jahre zur FORTH-Gemeinschaft. Er lieferte viele wertvolle Beiträge, sei es als Vertreiber von FORTH, Programmierer oder aktives Mitglied der FORTH Interest Group Direktoriums. Mike Ham interviewte ihn für 'FORTH Dimensions'. Er erfuhr dabei allerlei über FORTH und die FIG und erhielt darüber hinaus einen Einblick in das Leben von Martin Tracy.

MH:

Arbeiten Sie immer noch für die FORTH, Inc.?

MT:

Ja. Ich bin dort Oberprogrammierer. Letztes Jahr implementierte ich das digitale Signale verarbeitende FORTH für den TMS 320-22 von Texas Instruments. Das wird jetzt verkauft und ich arbeite im Moment an anderen Projekten, was gerade so anfällt.

MH:

Für welche Computer?

MT:

Viele wollen, daß wir Software für den IBM schreiben, meistens für den AT. Aber das variiert. Etwas machen wir für den 68000er. Viele Leute aus der Prozeßsteuerung benutzen IBM PCs. Ich würde zwar nicht sagen, daß dieser Computer das Beste ist was es momentan gibt, aber er kommt dem nahe.

MH:

Wie sind Sie zu FORTH gekommen?

MT:

Meine erste Begegnung mit FORTH fand statt, als ich an einem elektrischen über Muskelimpulse gesteuerten künstlichen Arm für jemanden arbeitete, dessen Arm unterhalb des Ellbogens amputiert ist. Das bedeutet, daß noch genügend Armstumpf zur Verfügung steht, um den künstlichen Arm am Stumpf zu befestigen und Kontakt zu den verbleibenden Muskeln herstellen zu können.

FORTH muß anders als andere Sprachen behandelt werden.

Jeden Morgen, wenn man die Prothese anlegt, wird sie durch Konzentration auf eine bestimmte Bewegung eingerichtet - dadurch werden die Reste der Muskeln aktiviert. Der Computer gibt acht, erkennt die Absicht und bewegt den künstlichen Arm entsprechend. Man hat dadurch eine gewisse Kontrolle über den Arm, aber natürlich kein Gefühl.

Der Prozessor im Arm war ein RCA 1802. Er wurde auf einem Decus FORTH Entwicklungssystem programmiert. Deshalb mußte ich das Decus FORTH Handbuch lesen. Ich verstand es damals nicht und beschäftigte mich deshalb nicht weiter mit FORTH. Das blieb für ein, zwei Jahre so. In dieser Zeit schrieb ich einen Pascal Compiler, der einige Jahre von Programma International verkauft wurde.

Dann fing ich an die ideale Programmiersprache zu suchen, worunter ich die größtmögliche Portabilität verstand. D.h. es sollte mit dieser Programmiersprache möglich sein, meine Tools auf den verschiedenen Computern, mit denen ich arbeitete, unterzubringen. Unser Desktop-Computer im Labor verstand nur BASIC, unser Minicomputer lediglich Fortran, die Statistikprogramme, mit denen ich arbeitete, waren in APL und ich ärgerte mich natürlich darüber, daß ich die Tools dauernd hin und her übersetzen mußte.

MH:

Welchen Beruf übten Sie aus, als Sie an der Armprothese arbeiteten?

12

Interview mit Martin Tracy

Tracy

MT:

Ich hatte eine Ganztagesstelle als Lehrbeauftragter in der Tanzabteilung der UCLA (engl. University College Laboratories). D.h. ich lehrte dort Anatomie für Tänzer, während ich meinen Dr. phil. in Biotechnik machte.

MH:

Und der Arm war Teil Ihrer biotechnischen Ausbildung?

MT:

Ja, genau. In den biotechnischen Labors der UCLA.

MH:

Sie beschäftigten sich also nicht mehr mit FORTH und machten sich an die Entwicklung der idealen Programmiersprache.

MT:

Zuerst war es ein Makroassembler. Phil Wasson bemerkte dann, daß ich eine Sprache entwickelte, die dem FIG-FORTH ziemlich glich. Ich schaute mir daraufhin das FIG-FORTH an und glaubte, daß ich dieses Modell auf einen Apple Computer implementieren könnte. Es war zwar schon auf einem Apple Computer, aber es hatte noch nicht die Form, die ich benötigte. Ich meinte, daß die Implementierung wohl in einem Monat zu schaffen sei. Phil war zu dieser Zeit Programmierer bei Programma International und arbeitete mit deren FORTH-Version. Und tatsächlich gelang es mir, FORTH in kurzer Zeit zu konvertieren. Deshalb beschlossen wir eine Firma zu gründen - MicroMotion - und FORTH für den Apple Computer zu verkaufen. So kam ich also zu FORTH.

Damals erreichte ich das siebenjahres Limit als Lehrbeauftragter der UCLA. Die UCLA sieht es nicht gern, wenn Lehrbeauftragte länger als sieben Jahre bleiben. Deshalb verließ ich die UCLA Tanzabteilung und begann mit MicroMotion, in der Hoffnung, daß die Firma sich selbst tragen würde, wenn ich immer wieder zum Tanzen weggehen würde. Das lief dann auch einige Jahre ganz gut.

MH:

Welche Art von Tanz machen Sie?

MT:

Klassisches Ballett und tänzerische Charakterrollen. Aber das mache ich jetzt nicht mehr.

MH:

Was sind "tänzerische Charakterrollen"?

MT:

Diese Rollen werden von den weniger hübschen Ballettänzern übernommen; z.B. der gestiefelte Kater, König Drosselbart, böse Zauberer usw.

MH:

So überließen Sie also die Firma sich selbst, während Sie nicht da waren ...

MT:

Ja, ich wollte mich zwar schon gerne FORTH und Computern widmen, aber man kann nur tanzen, solange man jung ist. Deshalb entschloß ich mich dem Tanzen zunächst Vorrang zu geben. Das ist natürlich bei einem sich schnell wandelnden technischen Gebiet nicht gerade gut, wenn man die Absicht hat viel Geld zu verdienen, aber für mich war der Tanz die richtige Wahl. Linda Kahn leitete die Firma während meiner Abwesenheit. Das waren also die Anfänge von MicroMotion.

MH:

Jetzt verstehe ich auch, wie die kleine Tänzerfigur in das Firmenzeichen von MicroMotion kommt. Tanzten Sie meistens an der Westküste?

MT:

Die meisten Engagements hatte ich im Orient, in Japan und in Taiwan. Ein paar auch in Texas und New York. Ich tanzte beim American Festival Ballett, beim Radio City Music Hall Ballett, bei West Side Tours ...

MH:

Ergaben sich eigentlich direkte Berührungspunkte von FORTH und Tanz? Ich denke dabei z.B. an Labanotation (Labanotation oder das Laban System ist ein relativ unbekanntes System zum Aufzeichnen von Tanzbewegungen. - die Red.).

MT:

Ja, tatsächlich bin ich Lehrer für Labanotation. Ich habe es mehrere Jahre unterrichtet. Die Grundlage meines Unterrichts war ein computerunterstütztes Aufzeichnungssystem für Bewegungen.

MH:

Auf der Basis der Labanotation?

MT:

Es beinhaltete zwar Elemente der Labanotation, aber das Aufzeichnungssystem basierte auf der Tatsache, daß Muskeln und Knochen mathematisch ziemlich genau nachgebildet werden können. Auch die Labanotation besitzt als Kernstück ein Modell des Körpers, das einfach in Computer-Ausdrücke umgewandelt werden kann. Ich habe dann ungefähr 1975 im Rahmen der Konferenz des Komitees "Research on Dance" einen Vortrag über Computer und Tanz gehalten. Aber das Thema interessierte mich eigentlich nicht so sehr. Es lief schließlich darauf hinaus, daß ich nichts mehr in der Richtung unternahm.

MH:

Nach der Gründung von MicroMotion erschienen in rascher Folge andere Versionen Ihres FORTH.

MT:

MicroMotion existiert zwar immer noch, aber ich habe nichts mehr damit zu tun. Nach der Sache mit Apple, widmeten wir uns dem Z80, dann dem Commodore 64, dann dem IBM PC

und schließlich produzierte Ray Talbot ein Macintosh MasterFORTH für uns.

MH:

Während dieser Zeit schrieben Sie die Einführung in FORTH mit dem Titel "Mastering FORTH".

MT:

Ja, richtig. Die allererste Version davon war ein kleines gelbes Buch. Sie erschien als MicroMotion noch immer mit FORTH-79 arbeitete. Es war meiner Meinung nach die erste Einführung in FORTH überhaupt. Das Buch von Brodie "Starting FORTH" kam erst anderthalb Jahre später heraus. Die Version die als "Mastering FORTH" bekannt wurde, kam zwei Jahre nach "Starting FORTH" heraus. Im Augenblick bereite ich eine zweite Ausgabe von "Mastering FORTH" für Brady Books (Prentice Hall) vor.

MH:

Stimmt diese Version immer noch mit MasterFORTH überein?

MT:

Sie wurde lediglich erweitert. Die Grundlagen - obschon überarbeitet - sind dieselben wie bei "Mastering FORTH", aber es kamen Kapitel über Themen hinzu, von denen ich glaube, daß sie in anderen FORTH-Büchern nicht oder nicht ausreichend behandelt werden, wie z.B. Target Compilation, Graphik und Fließkommazahlen.

MH:

Kommt es 1988 heraus?

MT:

Es ist vorgesehen das Buch bis Ende März 1988 herauszubringen.

MH:

Warum haben Sie MicroMotion verlassen?

MT:

Als ich vierzig wurde, entschloß ich mich vom Ballett zurückzuziehen und ein Heim zu gründen. Um das zu verwirklichen, braucht man in Südkalifornien eine Menge Geld. Deshalb verkaufte ich MicroMotion und begann für die FORTH, Inc. zu arbeiten.

MH:

Während Sie als Biotechniker arbeiteten, hatten Sie ja schon Programmiererfahrung. Wie kamen Sie eigentlich zum Programmieren?

Interview mit Martin Tracy

Tracy

MT:

Was mich immer schon am meisten interessierte sind Menschen in Bewegung. Das liegt zum Teil an meiner Arbeit als Tänzer und teils an der Labanotation sowie der Effort-Shape-Methode und anderen Methoden zur Aufzeichnung von Bewegungen.

MH:

Was bedeutet "Effort-Shape"?

MT:

Damit bezeichnet man das Wesen der Bewegung, im Gegensatz zur Richtung, in der sich die Gliedmaßen bewegen.

Ich habe verschiedene Formen der Bewegungsaufzeichnung unterrichtet. Außerdem habe ich einige Aspekte der nonverbalen Kommunikation und der Anatomie für Tänzer in meinen Unterricht miteinbezogen.

MH:

Was versteht man unter "Anatomie für Tänzer"?

MT:

Dabei handelt es sich um ein Hand- und Übungsbuch für den Körper. Was man mit ihm anfangen kann, was nicht, wie die einzelnen Komponenten des Körpers zusammenspielen und was man tun kann, daß sie zusammenspielen. Am Ende meiner Lehrtätigkeit habe ich es so gestaltet, daß es den Bedürfnissen von Tänzern und Leuten, die asiatische Kampfsportarten betreiben, entsprach. Ballett wird überwiegend von Frauen ausgeübt, asiatische Kampfsportarten von Männern. Aber da die Körper von Männern und Frauen sich sehr gleichen, konnte ich mit meinem Handbuch genug Studenten ansprechen, die sich entweder für das eine oder das andere interessierten.

...

MH:

Haben Sie eine Ausbildung in asiatischen Kampfsportarten?

MT:

Ich habe siebzehn Jahre lang die verschiedensten chinesischen Kampfsportarten studiert.

MH:

Wie kamen Sie eigentlich zum Tanz? In unserer Kultur ist es ja nicht üblich, daß die Kinder von klein auf an den Tanz herangeführt werden.

MT:

Als ich ca. 15 Jahre alt war, fragte mich eine Freundin, ob ich nicht Lust hätte, ihr bei ihren Tanzübungen zu helfen. Ich hatte keine Ahnung vom Ballett, aber ich ging trotzdem hin und half ihr ein bißchen beim Tanzen. Dabei sah mich der Direktor des American Festival Balletts und meinte ich sei begabt. Da begann ich Stunden zu nehmen und nach einem Jahr hatte ich schon meinen ersten Auftritt - was übrigens bei einem männlichen Tänzer nichts ungewöhnliches ist. Für eine Frau wäre es schon ungewöhnlich, für einen Mann dagegen nicht.

MH:

Wo waren Sie damals?

MT:

Ich wohnte in Providence, Rhode Island. Meine Aufgaben als Tänzer hatte ich meistens in New York. Von da an war ich immer Tänzer und etwas anderes - das "etwas andere" änderte sich von Zeit zu Zeit.

Im UCLA nahm ich Unterricht in Kinesik, der Lehre von der Bewegung des Körpers. Später dann als Assistent hielt ich Vorlesungen und gab Stunden in Elektromyographie und Biotechnik am Institut für Kinesik. Zu diesem Zeitpunkt wurde es dann notwendig, mathematische Aspekte zu berücksichtigen und sich mit Computern zu beschäftigen. Nehmen wir einmal an, ich muß wissen, was im Innern der Hüfte vorgeht, da es mich interessiert, warum Tänzer über 30 sehr oft an Hüftgelenksarthrose erkranken und manchmal sogar künstliche Hüftgelenke benötigen. Natürlich kann man keinen Wandler in die Hüfte eines Tänzers einsetzen. Das einzige was man tun kann, besteht darin die Hüfte mit hoher Geschwindigkeit von außen aufzunehmen, schauen welche Kräfte auf die Hüfte einwirken und daraus ableiten, was in der Hüfte vorgeht. Um all' dies bewerkstelligen zu können, benötigt man raffinierte und ausgereifte Werkzeuge: Hochgeschwindigkeitsfilmkameras, mechanische und mathematische Modelle der sich bewegenden Objekte usw. Als ich alles über Kinesik gelernt hatte, ging ich ans Institut für Biotechnik, um weiter zu lernen. Je mehr ich mich mit Biotechnik befaßte, um so mehr näherte ich mich den Computern und mußte deshalb auch immer mehr über sie lernen. Und so kam ich zum Programmieren.

MH:

Gibt es irgendwelche Bewegungsprojekte bei der FORTH, Inc.?

MT:

Da gibt es eigentlich nichts dergleichen. Was dem noch am nächsten kommt sind Roboter, doch ihnen fehlt die Menschlichkeit. Sie sind für meinen Geschmack einfach zu berechenbar.

MH:

Meinen Sie das Fehlen des Willens oder zu wenig Bewegungsmöglichkeiten?

MT:

Das Fehlen des Willens. Wenn Sie z.B. verlegen sind, bewegen Sie sich ganz anders. Ich weiß aber nicht, wie ich einen Roboter in Verlegenheit bringen soll.

MH:

Ich möchte gerne wissen, was Sie über die FIG denken. Woher kommt sie und wie es mit ihr weitergeht?

MT:

Ich glaube sie kommt mehr aus der Ecke der Hobbyanwender. Es wäre aber meiner Meinung nach nicht schlecht, wenn daraus eine

Tracy

professionellere Organisation würde. Ich weiß nicht, ob das möglich ist oder nicht, ich glaube wir probieren es gerade aus, dann wird man weiter sehen.

MH:

Welche Möglichkeiten hat Ihrer Meinung nach eine professionelle Organisation? Gibt es schon einen Plan oder muß das sich im Lauf der Zeit entwickeln?

MT:

Das Modell, das ich im Kopf habe, ist irgendwo zwischen einer professionellen Gemeinschaft und einer Public Relations Firma angesiedelt. Eines der Dinge, die die FIG für FORTH tun kann, ist die Verbreitung des Namens. Ich finde es eigentlich seltsam, daß es eine Organisation für FORTH gibt. Für mich ist eine Programmiersprache immer das Medium, das zwischen dem Anwender und dem Problem steht; manche Programmiersprachen behindern dabei mehr als andere. Was mir besonders an FORTH gefällt ist, daß es nicht so sehr bei der Bewältigung eines Problemes im Wege steht. Aber es steht natürlich im Weg. Es ist doch so, wenn ich die Temperatur einer Lampe bestimmen will und Sie sagen mir, daß ich dazu zuerst Maschineschreiben lernen muß, dann ist das eine Fertigkeit, die ich nicht brauche.

Für mich ist eine Programmiersprache in erster Linie dazu da, ein Problem zu lösen und mich bei der Problemlösung in Frieden zu lassen. Das ist ein Grund, warum mich FORTH anzieht: es läßt mich in Ruhe arbeiten. Wäre ich ein Manager und Sie würden zu mir sagen "Treten Sie der FORTH Interest Group bei" oder "Kommen Sie und erleben Sie FORTH auf der FORTH Tagung", so würde ich sagen "Warum?".

Würden Sie hingegen sagen "Kommen Sie zur FORTH Tagung, dort können Sie gute Programmlösungen zu Problemen der real existierenden Welt auf existierender Hardware sehen" oder "Treten Sie der FORTH Interest Group bei, um mehr über die Lösung solcher Probleme zu erfahren", so würde ich ja sagen.

MH:

Wir stehen uns also gewissermaßen selbst im Weg, indem wir Versammlungen abhalten, bei denen das Thema FORTH ist, anstatt uns mit Lösungsmöglichkeiten für bestimmte Probleme zu befassen.

MT:

Ja, genau. Wir sollten uns auf das konzentrieren, was wir können. Ich glaube wirklich, daß FORTH die beste Sprache zur Lösung von bestimmten Problemen ist. Aber anders als manche meiner Kollegen, glaube ich nicht, daß damit alle Probleme gelöst werden können.

MH:

Die Rochester Group hatte ja sehr viel Erfolg damit, als sie ein bestimmtes Problem zum Thema ihrer Versammlungen machte - AI, Roboter u.ä.. Es kamen daraufhin die Leute, die sich für diese Themen interessierten. Sie erfuhren unzweifelhaft eine Menge über FORTH, wurden aber ursprünglich von einem Problem und dessen Lösung angezogen. Sehen Sie darin eine Möglichkeit für zukünftige FIG Tagungen?

MT:

Ja. Ich habe mich bereit erklärt, die nächste FIG-Tagung in Anaheim zu organisieren. Dabei möchte ich Menschen erreichen, die FORTH noch nicht benutzen. Ja, ich will noch mehr: Leute, für die es ganz abwegig klingt, FORTH zu lernen, sollen angesprochen werden, indem wir ihnen sagen "Schaut Euch die Arbeitsweise des Nervensystems auf der FORTH Tagung an" oder "Schauen Sie sich RISC oder WISC orientierte Prozessoren auf der FORTH Tagung an". Sie sollen die Möglichkeit haben, FORTH mit kleinen Buchstaben anstatt mit großen kennenzulernen. Sie sollen etwas Sichtbares und Hörbares erfahren - echte Probleme der real existierenden Welt. Es macht Spaß bei deren Lösung zuzusehen. Da FORTH eine solche Sprache ist, ist es doch nicht notwendig dauernd mit einem Lehrbuch, einem Stück Papier oder einer Theorie zu winken.

MH:

Oder mit einer Case-Anweisung?

MT:

Genau. Don Colburn hat eine gute Idee und ich versuche sie zu realisieren. Ein Programmierwettbewerb, bei dem der erste Preis mindestens 1000$ betragen soll. Für die Teilnehmer wird ein Raum mit Tischen und Stromanschlüssen zur Verfügung gestellt. Alles andere muß mitgebracht werden: beliebige Computer, sowie beliebige Software. Man kann auch als Team erscheinen und außerdem alles mitbringen, was zur Lösung des Problems benötigt wird.

Es wird eine Freude sein, bei der Problemlösung zuzusehen. Danach lassen wir die Programme laufen, damit jeder Interessierte kommen und sie sich anschauen kann. Ich werde Microsoft Quick Basic, Turbo C und andere auffordern teilzunehmen. Alle sollen kommen und mitmachen.

MH:

Sie haben eine bestimmte Art und Weise, wie Sie Dinge erledigen. FORTH entstand in einer Umgebung, in der viele Programmierer für sich selbst arbeiteten. Aber die FORTH, Inc. beschäftigt jedoch ein großes Team von FORTH-Programmierern, die an Teamprojekten arbeiten. Was sagen Sie zum Einsatz von FORTH bei der Teamarbeit?

MT:

Vor allem bin ich der Überzeugung, daß FORTH anders als andere Sprachen gehandhabt werden muß. Umfangreiche Aufgaben werden unterschiedlich aufgeteilt. Ich glaube nicht, daß ein einfaches FORTH (ohne lokale Variablen u.ä.) sich sehr gut zum Einsatz bei großen Projekten eignet. Auch in Anbetracht der Tatsache, daß schon viele umfangreiche Probleme mit FORTH gelöst wurden. Als erstes muß das FORTH erweitert werden, damit

Interview mit Martin Tracy

Tracy

das große Projekt gehandhabt werden kann. Anschließend wird mit dieser Erweiterung gearbeitet. Aber ob es sich dann immer noch um FORTH handelt...

Ich will Ihnen einmal ein Beispiel geben: Sie können ein C in FORTH schreiben und daraufhin ein Programm schreiben. Schreiben Sie dann ein Programm in C oder FORTH? So wie ich das sehe, programmieren Sie dann in C. Mich interessiert aber eigentlich nur, was man mit FORTH selbst machen kann und nicht was man aus FORTH alles machen kann.

MH:

FORTH pur ist für Sie also eine Sprache für den Einzelnen. Ein Team benötigt eine Sprache, die den Aufgaben und dem Team gerecht wird, mit lokalen Variablen usw. Und dann ist es kein FORTH mehr.

MT:

Richtig, aber die FORTH, Inc. wird mir da nicht zustimmen. Ein Versuch, den wir in dieser Richtung unternehmen, besteht darin, ein Problem in einzelne Teile zu zerlegen, die man zur selben Zeit erledigen und ablaufen lassen kann. Verschiedene Programmierer arbeiten an unterschiedlichen Aufgabenbereichen. Diese Einzelteile fügt man dann zusammen und läßt sie gleichzeitig laufen, damit sie ein System bilden.

MH:

Unter Einsatz vieler Vokabulare, damit Kollisionen vermieden werden...

MT:

Nein, wir benötigen nicht viele Vokabulare. Wir arbeiten mit Hilfswerkzeugen, die Namenskonflikte aufspüren und entsprechende Änderungen vornehmen.

MH:

Sie haben schon sehr viel in FORTH programmiert. Gibt es ein Lieblingsprojekt, irgendetwas, das Ihnen am besten gefällt.

MT:

Ja, die LISP-Erweiterungen, die ich für die FORTH Model Library gemacht habe, waren sehr interessant, die in Band 1 der Bibliothek.

MH:

Sie besitzen die Fähigkeit ein interessantes und wichtiges Problem zu erkennen und es komplett zu lösen.

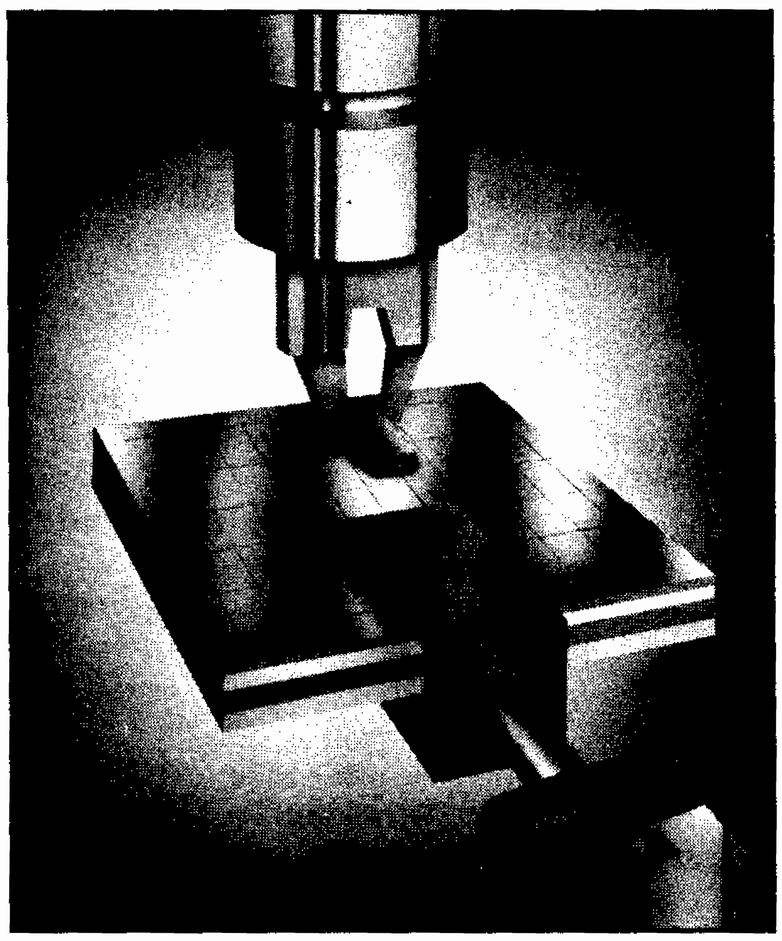
MT:

Es gibt da eine bestimmte Vorgehensweise, wenn ich ein Problem löse. Zuerst vertiefe ich mich in die Materie. Ich trage alles zusammen was ich in kurzer Zeit über den Problemkomplex in Erfahrung bringen kann. Dabei gehe ich z.B. in Bibliotheken, um Bücher zum Thema zu suchen. Bücher sind überhaupt meine bevorzugte Materialquelle.

Ich sammle also soviele Informationen zu dem Thema, wie ich kriegen kann und lese ein bißchen was darüber, ohne viel davon zu verstehen. Es dient nur dazu, mich mit der Materie vertraut zu machen. Dann lasse ich etwas Zeit verstreichen, einen halben oder ganzen Tag, in der ich mich nicht mit dem Problem beschäftige. Anschließend fange ich mit der eigentlichen Problemlösung an.

(aus 'FORTH Dimensions' Vol IX, Nummer 6, Seite 31-35)

EINIGE ROBOTER KÖNNEN NICHT GLEICHZEITIG GEHEN UND ZUGREIFEN...



...WAHRSCHEINLICH HABEN SIE NOCH NIE ETWAS VON FORTH GEHÖRT.

polyFORTH II® ist die vierte Generation professioneller FORTH-Systeme und wurde speziell für Echtzeitanwendungen entwickelt. Es ist im Multiuser-Betrieb außergewöhnlich leistungsfähig, und eine beliebige Anzahl asynchroner Prozesse können konkurrierend ablaufen.

polyFORTH II hält für den Software-Entwickler alle Werkzeuge bereit, um selbst die präzisesten und kniffligsten Roboterleistungen zu programmieren. Weil polyFORTH II maximale Flexibilität und Erweiterbarkeit in einem kompakten Paket vereinigt, hat es neue Maßstäbe für die an Robotersoftware zu stellenden Anforderungen gesetzt.

Die Wahl fällt auf polyFORTH II, wenn es um die Programmierung von Steuerungssystemen für Roboter und Frontend-Rechner geht; auch für alle anderen Echtzeitanwendungen wurde polyFORTH II geschaffen: Meß- und Prüfsysteme, Datenerfassung und -analyse, Bildverarbeitung, Fertigungssteuerung und Automatisation, u. a. m.

Das benutzerfreundliche polyFORTH II® gibt es als Hochleistungs-Native-System für IBM-PC, -XT und -AT, Intel 8086/8088 SBC's, Motorola 68000, RCA 1802 und 1805 und für DEC PDP-11 und LSI-11.

Auch ist es als betriebssystem-residentes und -kompatibles System für MS-DOS, RSX/VMS, CP/M-80 und CP/M-86 erhältlich.

polyFORTH II® ist eine vollständige, voll integrierte Programmierumgebung und alle Funktionen, die gemeinhin durch separate Programme repräsentiert werden, sind gleichzeitig systemresident: Betriebssystem, Editor, Assembler, Compiler, Debugger, Dienstprogramme und Interpreter. Ein Target Compiler ist für verschiedenste Zielprozessoren verfügbar.

Sprechen Sie noch heute mit uns . . . dann tanzt Ihr Roboter vielleicht schon morgen wie Baryshnikov.

Alleinvertretung der FORTH, Inc., USA, in Deutschland, Österreich und Schweiz:

RSO Gesellschaft für technische Kybernetik
Am Moosfeld 85 · 8000 München 82
Tel.: 089-429188, Telex 5212678 rso d

FORTH, polyFORTH und Target Compiler sind eingetragene Warenzeichen der FORTH, Inc. U.S.A.

Der VBL-Interrupt und das volksFORTH83

Thomas Jung, Mainz

Stichworte:
Atari ST, Interrupt,
volksFORTH83

In diesem Artikel werden einige Worte definiert, die eine bequeme Nutzung des Vertical-Blank-Interrupts beim Atari ST aus dem FORTH ermöglichen.

Der VBL-Interrupt des Atari ST wird bei jedem vertikalen Strahlrücklauf des Bildschirms ausgelöst. Je nach angeschlossenem Monitor ist das 50-70 mal pro Sekunde. Unter anderem arbeitet die Interruptroutine auch die sogenannte VBL-Queue ab, das ist eine Reihe von Adressen (die "Slots"), in die der Benutzer Adressen von eigenen Routinen ablegen kann, die dann bei jedem VBL-Interrupt ausgeführt werden.

In den angeführten Screens wird dazu das Wort HOOK definiert, das die Adresse einer solchen Routine auf dem Stack erwartet und sie in einem freien Slot installiert. Mit dem Wort UNHOOK kann eine Routine dann wieder ausgeklinkt werden. Die Routinen werden mit dem Assembler im FORTH erzeugt und können so leicht vorher ausgetestet werden. Im Screen 6 wird ein kleines Beispiel der prinzipiellen Vorgehensweise gegeben.

HOOK und UNHOOK benutzen eine Reihe von Worten, die in den Shadowscreens ausführlich kommentiert sind. Sie können als Handwerkszeug für neue Worte genutzt werden, um zum Beispiel einen Überblick über - auch außerhalb des FORTH - installierte Routinen zu erhalten und diese zu manipulieren.

```
 volksFORTH-83     FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks VBLINTER.SCR  Seite 1

                   1                                                    8

 0 \  Loadscreen                              tj 26.04.88                          tj 26.04.88
 1
 2
 3 Onlyforth
 4
 5 2 5 thru
 6
 7 \ \needs Code   2 loadfrom assemble.scr                 Kleines Beispiel, wozu wir natürlich den Assembler
 8 \ 6 load                                                brauchen.
 9
10
11
12
13
14
15
```

VBL-Interrupt beim Atari ST

2

```
0 \  VBL 1 - Konstanten und Kleinigkeiten          tj 24.04.88
1
2 $454.  2Constant  nvbls
3 $456.  2Constant  vblq
4
5
6 : #slots ( -- n) nvbls 1@ ;
7 : firstslot ( -- laddr) vblq 12@ ;
8 : slot  ( i -- laddr ) 4 * 0  firstslot  d+ ;
9 : slot@ ( i -- laddr ) slot 12@ ;
10 : clear_slot ( i --  ) >r 0. r> slot 12! ;
11
12 : in_forth? ( laddr -- flag)
13    2dup forthstart d<  -rot
14    forthstart $ffff. d+ 2swap d<
15    >r not ;
```

3

```
0 \  VBL 2  - neue Routinen einhängen              tj 23.04.88
1
2 : vbl_free ( -- 0.|laddr)
3     #slots 0 DO
4                 i slot@  d0=
5                 IF i slot endloop exit    THEN
6             LOOP
7     0. ;
8
9
10 : vbl_install ( laddr -- flag)
11    vbl_free  2dup d0=
12    IF 2drop 2drop   false exit   THEN
13    12!   true ;
14
15
```

9

```
                                                   tj 26.04.88

NVBLS (lange Adresse) enthält die Anzahl der möglichen
     VBL-Routinen  (normalerweise 8).
VBLQ  (lange Adresse)  enthält einen Zeiger auf das Feld
     mit den Adressen der VBL-Routinen ( den "Slots").
#SLOTS liefert die Anzahl der VBL-Routinen.
FIRSTSLOT gibt die Adresse des ersten Slots,
SLOT die der weiteren.
SLOT@ legt die Adresse der VBL-Routine in Slot i auf den Stack.
CLEAR_SLOT löscht eine Routine, indem es 0.0 in ihren Slot gibt.

IN_FORTH? prüft, ob die (lange) Adresse im FORTH liegt.
```

10

```
                                                   tj 24.04.88

VBL_FREE sucht einen freien Slot in der VBL-Queue:


        Slot i ist frei.

        Schon alles belegt.


VBL_INSTALL steckt die Routine ab laddr in einen freien Slot
           und meldet
           Fehlschlag
           oder Erfolg .
```

VBL-Interrupt beim Atari ST

```
volksFORTH-83     FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks  VBLINTER.SCR  Seite 1
```

```
                        4                                                  11

0 \  VBL 3  - alte Routinen löschen                tj 24.04.88                                                  tj 26.04.88
1
2 : vbl_kill ( laddr --)                                                  VBL_KILL prüft
3       #slots 0 DO 2dup i slot@                                                  den Inhalt aller Slots auf die Routine laddr
4                   d= IF i clear_slot THEN                                       und löscht sie, falls vorhanden.
5               LOOP
6       2drop  ;
7
8 : vbl_killall                                                           VBL_KILLALL
9     #slots 0 DO  i slot 12@                                                   löscht alle VBL-Routinen,
10                  in_forth? IF  i clear_slot  THEN                              die sich im FORTH befinden.
11              LOOP ;
12
13
14
15
```

```
                        5                                                  12

0 \ VBL 4   - Benutzerworte                        tj 24.04.88                                                  tj 26.04.88
1
2 : hook  ( addr -- )                                                     HOOK klinkt die Routine ab FORTH-Adresse addr in die VBL-Queue
3     >absaddr  vbl_install                                                     ein.
4     not abort" Interrupt not installed!" ;
5
6
7 : unhook ( addr -- )                                                    UNHOOK klinkt sie wieder aus.
8     >absaddr  vbl_kill ;
9
10
11
12 : bye    vbl_killall  bye ;                                            BYE muß dafür sorgen, daß vor dem Verlassen von FORTH alle VBL-
13                                                                            Routinen, die sich im FORTH befinden, ausgeklinkt werden.
14                                                                            Sie würden beim Verlassen von FORTH ins Leere laufen.
15
```

```
                        6                                                  13

0 \ VBL 5  - Beispiel                              tj 24.04.88                                                  tj 24.04.88
1
2 Variable timer                                                          VBL_TIMER ist eine kurze Beispielroutine.
3 Create vbl_timer    Assembler                                                   Sie verwandelt eine Forthvariable in einen Timer.
4                     1  timer )absaddr 1#) addq
5                     rts
6                     end-code
7
8
9
10
11 timer off    vbl_timer hook                                            VBL_TIMER wird jetzt nach jedem VBL-Interrupt ausgeführt.
12
13
14
15
```

Periodenlänge von Dezimalbrüchen 100 Stellen nach dem Komma und mehr

Andreas Soeder,
Gruppe Darmstadt

2401 hätte eine lange Periode, las ich [1] - wie lange stand nicht da, aber ich wollte es gerne wissen. Also mit Taschenrechnerunterstützung ging ich ans Werk. Bald war ein kleiner Zettel voll, dann ein großer. Bei dieser Methode stieg gefühlsmäßig die Wahrscheinlichkeit eines Fehlers langsam gegen 1. Da fiel mir ein: Ich hatte doch mal ein Programm Namens 'Dividieren beliebig vieler Stellen nach dem Komma' geschrieben. Es funktionierte nach dem wie in der Schule gelernten Grundprinzip: Rest mal 10, dividieren: Rest mal 10, dividieren; usw.

Die Periodenlänge daran zu messen, wann die gespeicherte Ziffernfolge sich wiederholt, ist wesentlich mühsamer zu programmieren, als sich den Rest zu merken und immer zu vergleichen, ob er wieder auftritt, wenn ja, ist die Periode durchlaufen. Ist der Rest Null, dann geht die Division auf, und es wird "---" ausgegeben.

Nachdem die erste Tabelle gedruckt war, fiel auf, daß die Zahlen, die mit 1, 3, 7 und 9 enden, bevorzugt lange Perioden haben und häufig Primzahlen sind. Also machte ich mir die Mühe, die Primzahlen mit einem Pfeil zu markieren. Die größte beobachtete Periodenlänge einer Zahl n ist n-1. Das scheint mir verständlich, da 1..n-1 die Reste sind, wo es weitergeht, bei n folgt Null, und es geht auf. Vielleicht kann ein Mathematiker einen strengeren Beweis liefern und weitere Gesetzmäßigkeiten aufdecken.

Zurück zum Ausgangspunkt: 2401 hat die Periodenlänge 2058.

Zu PICK: das hier verwendete PICK entspricht dem von FORTH-79, FORTH-83 Benutzer müssen nur die Zahl davor ändern. fig-FORTH Benutzer, die ihre Source nicht ändern wollen, können es so installieren:

```
: PICK (n -- n.ter Stack-
  parameter auf TOS )
    2 * SP@ + @ ;
```

Zur Laufzeit des Programmes: von 1 bis 8191 dauerte es mit Druck auf dem ITOH 8500 etwa 5 Stunden, der Engpaß ist dabei 1200 Baud Drucker. Eine andere Version, bei der die Daten auf die Floppy kommen, dauerte, wenn ich mich recht erinnere, etwa 3 Stunden.

```
500 599 PERIODENTAB

ZAHL        0     1     2     3     4     5     6     7     8     9

  500     ---   166    50  502←     6     4    22    78    42  508←
  510      16    24   ---    18   256    34    21    46     6    43
  520       6  52←    28  261←   130     6   262   240     2   506
  530      13    58    18    30    44    53    33   178   268    42
  540       3  540←     5   180    16   108     6   91←     8    60
  550       2   252    22    78    69     3    46  278←    15    42
  560       6    16    28  281←    46   112   141    18    35  284←
  570      18  570←     6    95    30    22     1  576←   272   192
  580      28   246    96    26     8     6   146  293←    42    90
  590      58    98     3  592←     6    48   148    99    66  299←
OK
```

Beispielausdruck

1 mc 8/87 Seite 36

Periodenlänge von Dezimalbrüchen

```
0 99 PERIODENTAB

ZAHL        0     1     2     3     4     5     6     7     8     9

     0    ---←  ---←  ---←    1←  ---   ---←     1     6←  ---     1
    10    ---     2←     1     6←     6     1   ---    16←     1    18←
    20    ---     6      2    22←     1   ---       6     3       6    28←
    30      1    15←   ---      2    16       6     1     3←    18       6
    40    ---     5←     6    21←     2       1    22    46←     1    42
    50    ---    16      6    13←     3       2     6    18    28    58←
    60      1    60←    15     6   ---       6     2    33←    16    22
    70      6    35←     1     8←     3       1    18     6       6    13←
    80    ---     9      5    41←     6    16    21    28       2    44←
    90      1     6     22    15    46    18     1    96←    42       2
 OK
8000  8099  PERIODENTAB 7 EMIT

ZAHL        0     1     2     3     4     5     6     7     8     9

  8000    ---    42   500   975   308   200    87   624     6  2002←
  8010     44  2670← 1001  1335  4006   228   166  8016←   90   162
  8020    200   264   570   560   464    53    34  1276   222    30
  8030      8   223    50  1932   102  1606   210   414  4018  4019←
  8040     33   336   268  1146   670   201   444   618   502   447
  8050     66  3936    60  4026← 2013   178   234  3450   208  8058←
  8060     30  2686   644   122     6   403   108    42  2016  8068←
  8070    268  1152   252    66   366   144   224   490   576  1346
  8080      4  2020←   32   232   966    42   930  8086←  336  1348←
  8090    202   420   816  4046←  630  1618    22  2698  2024   132
 OK
( knapp 8 Minuten )  OK
```

```
SCR # 124 ANDREAS SOEDER .6.8.88
  0 ( PRIMZAHLEN NACH VD II/3,S.5          AS.861026 )
  1 HEX 8000 CONSTANT PRIMTAB ( RAM-DISK-BEREICH )
  2     2000 CONSTANT SIZE    ( =8192 DEZIMAL, PRIMZ.BEREICH )
  3 DECIMAL
  4 : PRIMES   ( --- ANZAHL DER PRIMZAHLEN )
  5   PRIMTAB SIZE 01 FILL    ( PRIMZ.FELD, ANF.BEDINGUNG )
  6   0                       ( PRIMZAHL ZAHLER           )
  7   SIZE 2
  8   DO PRIMTAB I + C@       ( PRIMZAHL ?                )
  9      IF I 91 <            ( NUR BIS WURZEL SIZE PRÜFEN )
 10        IF I PRIMTAB SIZE +
 11             PRIMTAB I  I + +  ( VON SIZE BIS 2I       )
 12             DO 0 I C! DUP +LOOP ( ALLE VIELFACHE V.I 0EN )
 13        DROP ENDIF        ( I VOM +LOOP WEGTUN          )
 14        1+                ( SCHLEIFENINDEX ERHOEHEN )
 15     ENDIF LOOP ;   ;S
```

Periodenlänge von Dezimalbrüchen

```
SCR # 226
  0 ( PERIODTAB-1 f. Period.länge v. Dez.brüchen  AS.880721 )          (
  1 ( für   6809 - fig - F O R T H + Erweiterung, hier PICK          ) (
  2 ( Wenn Primzahlen markiert werden sollen, muß PRIMES gelaufen     (
  3 sein, oder man muß die Markierung ausschalten dann sind Zahlen)   (
  4 ( von 0 bis 32768 annehmbar. )                                    (
  5                                                                   (
  6  0 VARIABLE  REST1   ( 1.REST vor dem Zählen )                    (
  7  0 VARIABLE  PERZAHL ( Periodenzähler        )                    (
  8  0 VARIABLE  NCR     ( Zeilenzähler          )                    (
  9                                                                   (
 10 : TABKOPF CR CR ." ZAHL    " 10 0 DO I 6 .R LOOP CR ;            (
 11                                                                   (
 12 : CR1 CR 1 NCR +! NCR § 60 ) IF 12 EMIT TABKOPF 0 NCR ! ENDIF ;   (
 13 ( CR1 gibt nach 60 Zeilen 'Neue Seite' und Tabellenkopf     )     (
 14  HEX  FF C0 C! DECIMAL ( EMIT ändern für Grafikzeichen )          (
 15 -->                                                               (

SCR # 227
  0 ( PERIODENTAB-2, PERIOD? NREST  AS.880721 )                       (
  1 : NREST 10 U* 3 PICK U/ DROP ;  ( n1 n2 --- n1 10n2/n1 )          (
  2                                                                   (
  3 : PERIOD? ( Zahl --- periodisch? periodenlänge )                  (
  4   10000 15 0 DO NREST LOOP                ( Sicherheitsläufe )  (
  5         0 PERZAHL ! DUP REST1 !           ( Anfangsbedingung )  (
  6   BEGIN 1 PERZAHL +! NREST DUP 0=                                 (
  7         IF ."    ---" 1                 ( Rest=0 Ausgabe, f )  (
  8         ELSE DUP REST1 § = DUP     ( DUP als flag f. UNTIL   )  (
  9              IF  PERZAHL § 5 .R    ( Ausgabe: Periodenlänge  )  (
 10              ENDIF                                                (
 11         ENDIF ?TERMINAL OR                                       (
 12   UNTIL                                                          (
 13    DROP DROP ;                                                   (
 14  -->                                                             (
 15                                                                  (

SCR # 228
  0 ( PERIODTAB-3 f. Period.länge v. Dez.brüchen  AS.880721 )         (
  1                                                                   (
  2 : PRIMZ? ( PRIMES muß gelaufen sein im Feld 1 = Primzahl )        (
  3    ( ZAHL )  PRIMTAB + C§ ;                                       (
  4                                                                   (
  5 : PERIODENTAB ( von bis --- Tabelle )                             (
  6   TABKOPF ( drucken )  0 NCR !                                    (
  7   1+ SWAP DO  CR1 I I 6 .R ."    "                                (
  8             10 0 DO DUP I +                                       (
  9                     DUP PERIOD?      ( periodisch ? )           (
 10                         PRIMZ?       ( Primzahl   ? )           (
 11                         IF 194 EMIT  ( Pfeil        )           (
 12                         ELSE SPACE ENDIF                         (
 13                   LOOP DROP                                      (
 14     10  +LOOP CR1 ;                                              (
 15  ;S                                                              (
```

Noch eine Bemerkung zu Scr#226, Zeile 4: 32768 ist zwar für PERIOD? annehmbar, aber wenn in PERIODENTAB 32769 angesprochen wird, hängt sich das Programm auf.

Software-Engineering auf Personal Workstations

Andreas Goppold

Stichworte:
ökonomisches Software-Engineering, Hypertext-Datenbanken für Software-Entwicklung, Subroutinen-Prozessoren, Workstation-Konzepte auf Personal-Computern

Personal Computing und Software-Engineering

Mit den heutigen 32-Bit Personal-Computern ist die Voraussetzung für das Computer-Aided Software-Engineering (CASE) auf Personal Workstations gegeben. Die Implementations-Lösungen für solche Systeme können sich im Einzelnen von den Ansätzen aus dem High-End Bereich unterscheiden, da die Kostenstruktur von Systemen, die dem Personal-Bereich (Microcomputer) entstammen, grundsätzlich anders ist als die im traditionellen Mainframe- oder Minicomputer-Bereich. Mit Hilfe ökonomischer CASE-Systeme läßt sich die exponentiell wachsende Komplexität heutiger und zukünftiger Software-Systeme mit linearem Aufwand von organisatorischer Komplexität bewältigen.

Kurze Bestandsaufnahme des Software-Engineering

Software-Engineering (SE) ist inzwischen eine anerkannte Methode bei der Produktion von Software (siehe SOM87). Allerdings ist die Standardisierung und Formalisierung der verwendeten Techniken noch sehr uneinheitlich, und erstreckt sich lediglich auf Teilbereiche. So ist z.B. die ADA-Produktionsumgebung APSE erst unvollständig spezifiziert und realisiert worden. Ursachen für den langsamen Fortschritt sind die hohen Kosten für solche Systeme, und die Unklarheit über die Aufgaben, die bewältigt werden müssen, sowie die Frage der Investitionsrechtfertigung.

Am weitesten entwickelt sind formale Methoden für die Aspekte des SE, die in engem Zusammenhang mit Programmiersprachen und Compilertechniken stehen. Diese Entwicklung hat in den letzten Jahren große Fortschritte gemacht. Die Konzentration auf Programmiersprachen, über die viel und kontrovers diskutiert wurde, ließ jedoch oft vergessen, daß die Formulierung einer Problemlösung in einer bestimmten Programmiersprache nur einen relativ geringen Anteil im Lebenszyklus der Software hat. (Siehe die Kontroverse um GOTO und BASIC (DIJK68), die Modesprache C, und die trotz allem immer noch vorherrschende Verwendung der Uraltsprachen Fortran und Cobol.)

Folgekosten der Software

Die Phasen: Design, Codieren, Testen und Debuggen machen bei kleinen Projekten unter 1 Mannjahr wohl 30 bis 50 % der Gesamtkosten der Software aus. Dieser Anteil wird aber immer geringer, je größer das Projekt ist. Die wirklich relevanten Kostenfaktoren stellen sich erst heraus, wenn das Projekt schon einige Jahre seines Lebenszyklus hinter sich hat. Sie können mit dem Stichwort Folgekosten charakterisiert werden: Dokumentation, Training von Anwendern und neuen Programmierern (da die Hersteller des Programms das Projekt zumeist nicht mehr betreuen), Optimierung, Modifikation, Portierung in

andere Anwendungen und Maschinen- bzw. Systemumgebungen. In vielen Fällen ist es überhaupt nicht möglich, ein Programm zu modifizieren, oder zu portieren, da die Kosten dafür nicht abschätzbar sind. Indirekte Kostenfaktoren entstehen, wenn mit Software weitergearbeitet werden muß, die den Anforderungen nicht mehr entspricht, weil Neuerstellung zu teuer ist. Die Wiederverwendbarkeit von Software ist heute immer noch sehr gering. Deshalb ist das Konzept des Software-IC schon seit einiger Zeit im Umlauf, aber noch immer recht nebulös.

Zitat: "Der Wert von Computersystemen wird nicht daran gemessen werden, wie gut sie sich für den Zweck eignen, zu dem sie geschrieben wurden, sondern wie gut sie für Zwecke zu gebrauchen sind, für die sie nie gedacht waren." (KAY84)

Software-Entwicklungsumgebungen

Die Notwendigkeit die Software-Produktion in den nicht-formalen Aspekten zu unterstützen, ist früh erkannt worden, und findet ihren Niederschlag in diversen Systemen unter der Bezeichnung Software-Entwicklungs-Umgebungen oder CASE (Computer aided Software Engineering). (Literatur dazu z.B. in ACM84 und IPE). Hier sollen vor allem ökonomische Aspekte der Realisation von Entwicklungsumgebungen im Personal-Workstation-Bereich behandelt werden, Aufwärts-Migration zu High-End-Workstations, sowie spezielle Datenbank- und Virtual-Prozessor-Konzepte, die auf die Workstation-Umgebung optimiert sind.

Ökonomische Aspekte des Software-Engineering

Sicherheit und Planbarkeit sind im Software-Engineering Hauptfaktoren. Viele Ansätze des Software-Engineering, und besonders in der Groß-EDV verbreitete Methoden schränken die Freiheit des Programmierers erheblich ein und man muß eine sehr niedrige Produktivität zu Gunsten der erhöhten Sicherheit in Kauf nehmen (vier COBOL-Zeilen, verifiziert und dokumentiert, pro Mann-Tag als Durchschnittswert der Programmierer-Produktivität). Dies ist im Mainframe-Bereich in der Vergangenheit ökonomisch vertretbar gewesen, da die Hardware-Kosten der verwendeten Maschine und damit die Gesamtkosten des Systems (inklusive Wartung etc.) so hoch waren, daß Software-Kosten dazu noch in einem ähnlichen Verhältnis standen. Diese Situation hat sich im Bereich heutiger Microcomputer-Anwendungen aber ins Groteske umgekehrt, so daß Software-Kosten oft den Kaufpreis der Hardware um den Faktor 100 überschreiten. Modernes Software-Engineering muß daher alle Hilfsmittel, die durch die Anwendung maschinen-unterstützter Methoden gegeben sind, einsetzen. Dadurch kann in Bereichen, in denen auf rasch wechselnde Anforderungen des Marktes und der Technologie reagiert werden muß und wo spezielle Software-Lösungen kurzlebig sind, das gegenwärtige und sich in Zukunft noch verschärfenden Verhältnis zwischen den Personal- und Hardware-Kosten effizient bleiben.

Software-Engineering und Programmierer-Produktivität:

Ein Tradeoff ?

Das Wort Tradeoff ist eines der Schlüsselworte des Computer-Engineering. Es bedeutet, daß man alles, was man auf der einen Seite als Vorteil haben will, irgendwo anders bezahlen muß. Und zwar nicht in Geld, sondern in Minderleistung, was oft ein schwerwiegenderer Faktor ist, da es grundsätzlich bedeutet, daß mehr Geld nicht in einem Ursachenverhältnis zu besserer Leistung steht. Dahinter verbirgt sich die Erkenntnis, daß es sich bei Computer-Systemen auf allen Ebenen um Systeme handelt, also um Komplexe, die nur aus dem Ganzen heraus verstanden werden können. In der Praxis bedeutet dies wiederum, daß es im Systemdesign nicht genügt, daß man mehrere Spezialisten zusammenbringt, die dann ein System entwerfen sollen, sondern daß die Integration der Spezialisten (also das Personal-Management) die Hauptaufgabe erfolgreichen Systemdesigns ist.

Wie die Erfahrung zeigt, steht die Produktivität des Einzelprogrammierers und seine Effizienz im großen Team oft in einem eigentümlichen Spannungsverhältnis. Zwar ist die Lehrmeinung, daß ein Software-Projekt um so sicherer und planbarer ist, je mehr die einzelnen Programmierer strikte Anweisungen und definierte Schnittstellen erhalten. Andererseits kann in der Praxis immer wieder festgestellt werden, daß kleine Teams von bis zu drei Programmierern schnell und effizient Systeme erzeugen, für die im Masseneinsatz (human wave approach) zehnmal mehr Personen eingesetzt werden. Diese Masseneinsätze führen zudem oft zu erheblichen Verzögerungen und Mehrkosten und enden sogar nicht selten mit kompletten Fehlschlägen (BROO75). Der Aufstieg der Microcomputer-Industrie zeigt, daß hier eine von den Arbeitsweisen der Mainframe-Tech-

nologie lange Zeit völlig abgekoppelte Software-Produktion entstanden ist, die es lernte, mit völlig anderen Ökonomie-Verhältnissen umzugehen, und Produkte zu schaffen, die in der Mainframe-Welt nicht machbar waren.

Der Thadhani-Effekt

Analysiert man die entscheidenden Faktoren, die hinter den besonders im Micro-Bereich gefundenen Effekten der Single-User-/Single-Computer-Programmierung liegen, so findet sich immer ein Effekt, der manchmal als Transparenz oder Immediate Feedback, manchmal auch als direkte Interaktion mit dem Computer bezeichnet wird. Bekannt ist dieses Phänomen schon lange. Die Einführung von Timesharing-Systemen beweist, daß man ihm früh Rechnung getragen hat. Das Prinzip wird hier der Thadhani-Effekt genannt, nach Veröffentlichungen von Thadhani (THAD81, THAD84). Es wurde unter diesem Namen auch empirisch erforscht (GOLZ87, MOLZ87). Aufgrund dieser Untersuchungen zeigt sich, daß die Produktivität von komplexen Arbeitsabläufen (wie z.B. Programmieren) an Computersystemen in einem inversen Verhältnis zur Reaktionszeit des Computers steht. Anders ausgedrückt: je schneller der Computer sinnvolle Antworten auf Eingaben macht, desto produktiver kann ein Programmierer arbeiten.

Markt-Verhalten: Immediate Feedback ist essentiell

Dieser Effekt wird ebenfalls durch eine andere Beobachtung belegt: Der phänomenale Erfolg des Programmiersystems Turbo-Pascal ist darauf zurückzuführen, daß hier erstmals ein System zur Verfügung stand, das einen etwa 20 mal schnelleren Feedback-Zyklus hatte als alle anderen vergleichbaren Programmiersysteme. Das wurde einerseits durch die schnelle Compilation bewirkt, andererseits aber dadurch, daß der System-Editor mit dem Compiler integriert war, und bei Auftreten eines Compile-Fehlers das System automatisch an die fehlerhafte Stelle des Programms ging und den Editor aufrief. Der Erfolg von Turbo-Pascal war so durchschlagend, daß dieses System praktisch das Monopol im Low-End Pascal-Markt erlangte und einige andere Systeme (z.B. UCSD-Pascal) völlig verdrängte.

Da die Personen, die Turbo-Pascal kaufen, hauptsächlich daran interessiert sind, ihre Programme schnell zum Laufen zu bringen, stellt der Markt damit einen guten Gradmesser für die Effizienz eines solchen Systems dar. In der Tat zeigt die weitere Entwicklung, daß dieses Prinzip allgemeinen Eingang in die Software-Welt gefunden hat. Besonders die Firma Microsoft, die vor Borland dieses Marktsegment stark dominiert hatte, zeigt, daß sie den neuen Trends Rechnung trägt (Quick-C, Quick-Basic). So werden heute inkrementelle Compiler angeboten, bei denen die Zyklus-Zeit noch einmal verkürzt ist, da nur noch der modifizierte oder neue Teil des Codes compiliert wird. Die Firmen Microsoft und Borland liefern sich im Augenblick ein regelrechtes Rennen, mit immer schneller compilierenden neuen Releases ihrer Produkte.

Natürlich ist die Beschleunigung der Compilation allein für das Software-Engineering kein Produktivitätsfaktor. Wie Weizenbaum treffend bemerkt, ist niemanden damit geholfen, daß er etwas 10 mal schneller oder effizienter tun kann, wenn er dabei ist, in ein Loch zu fallen (WEIZ76). Im Software-Engineering dreht es sich darum, sehr komplexe Strukturen zu erstellen

und zu manipulieren. Die Korrektheit des Vorgehens ist dabei nach wie vor die entscheidendste Komponente, die bei Ansätzen wie Turbo-Pascal, das eher den halb-professionellen Bereich anspricht, vernachlässigt wird.

Das Software-Produktivitäts-System LEIBNIZ

LEIBNIZ ist ein CASE-System, das auf der Basis des Personal-Workstation-Konzepts aufgebaut wurde. Es unterscheidet sich in diesem Ansatz von den meisten anderen CASE-Systemen. Auch die Entstehung und Finanzierung von LEIBNIZ ist hier von Bedeutung: Während fast alle anderen Systeme dieser Art dem industriellen oder universitären Bereich entstammen, wurde LEIBNIZ im wesentlichen von einem mittelständischen Konsortium finanziert. Die stärkste Auswirkung liegt in der Ökonomie des Systems, da eine konstante Anwendungsfähigkeit und Brauchbarkeit des Systems schon in der Entwicklungsphase gefordert war. Eine Konsequenz ist die Fähigkeit zur Aufwärts- und Abwärtsmigration, also daß sinnvolle Subsets des Systems im Microcontroller-Bereich eingesetzt werden können (Single-Chip-Prozessoren), sowie Portabilität auf die gängigsten Single-User Systeme und Workstations mit 80x86 und 680x0 Prozessoren (AT/386, MAC-II, bis SUN, Apollo) gegeben ist. Portierungen auf neue Prozessoren wie Transputer sind geplant. LEIBNIZ berücksichtigt die Entwicklungs- und Ökonomie-Tendenzen heutiger und zukünftiger Technologien.

Systemkomponenten von LEIBNIZ

Der Strukturteil von LEIBNIZ ist ein sehr flexibles Datenbank-Konzept, welches in idealer Weise die Verwaltung und Vernetzung der vielfältigen Informationen, die im Zusammenhang mit dem Software-Lebenszyklus stehen, ermöglicht. Dieses Datenbank-Prinzip ist in der Literatur auch als Hypertext bekannt. (Siehe dazu Bild 1: Hypertext und Programmierung) Als Prozedurteil wurde eine virtuelle Maschine gewählt, ein Subroutinenprozessor, auch F-Code Maschine genannt. Dieses Konzept, das analog zu einer P-Code Maschine leicht auf viele Zielprozessoren zu portieren ist, hat für das Software-Engineering den besonderen Vorteil, daß der verwendete Zwischencode leicht mit dem Sourcecode rück-verkettbar ist, was für Debugging und Testing äußerst wertvoll ist. Besonderes Gewicht wurde auf die Maximierung der Programmierer-Produktivität gelegt, was durch ein windowing User-Interface, integrierte Auskunftsfunktionen und Modellierungsmöglichkeiten (Selbst-Expertensystem) erreicht wird.

P-Code und F-Code-Maschine

P-Code wurde durch UCSD-Pascal bekannt. Es ist im wesentlichen eine virtuelle Maschine, die auf jedem Zielprozessor implementiert wird, und die den vom Compiler erzeugten Zwischencode interpretiert. Der Vorteil einer P-Code Maschine ist der, daß man den Compiler für die Sprache für alle Zielprozessoren nur einmal schreiben muß. Somit ist sichergestellt, daß ein einmal geschriebenes Programm unverändert auf allen Maschinen läuft, was sonst erfahrungsgemäß nicht immer gegeben ist. Auch nicht bei Produkten des gleichen Herstellers. So ist UCSD-Pascal sowohl auf MSDOS-Maschinen, als auch auf dem Macintosh und anderen Computern gleich. Der Nachteil der Methode ist der, daß der Code immer langsamer ist, als ein vergleichbarer Code eines Objektcode-Compilers, da er von einem Interpreter ausgeführt wird. Der hier verwendete Ansatz der F-Code Maschine benutzt ein wesentlich effizienteres Maschinenmodell als P-Code. Es kann mit Hilfe von optimizing Compilern dieselbe Ausführungsgeschwindigkeit wie ein Objekt-Code Compiler erreichen.

Die F-Code Maschine

Das Programmiersystem Forth hat bei der Entwicklung der F-Code Maschine als Modell gedient. (GOP84-1, GOP84-2, GOP84-3, GOP85).

Das Konzept ist einer P-Code Maschine ähnlich, weist aber mehrere gewichtige Unterschiede auf. Während die P-Code Maschine nur für eine Sprache vorgesehen ist (Pascal), ist die F-Code Maschine so allgemein, daß sie als runtime-Zwischencode für eine Vielzahl von Hochsprachen dienen kann. Weiterhin ist die F-Code Maschine ein Subroutinen-Prozessor, also dahin optimiert, Subroutinen unbegrenzter Hierarchietiefe sehr einfach und effizient zu realisieren. Dies hat für das Testen und Debuggen von Modulen, die als Subroutinen realisiert sind, besondere Bedeutung.

Source Code Debugging

Es ist der Trend der Software-Industrie, daß Sourcecode-Debugging zum Standard-Tool wird. Auch hier sei auf die neuen Entwicklungen wie Microsoft-Code-View sowie diverse Produkte auf Workstation-Computern verwiesen. Die logische Rückführbarkeit der implementierten Prozeduren auf den Source Code ermöglicht eine wesentlich schnellere Testphase, verglichen mit Methoden wie Speicherdumps und Assembler-Level Debugging. Für diese Zwecke, sowie Prototyping, ist die schnellstmögliche Ausführung von Programmen durch die Ziel-Maschine zweitrangig. Ein Maschinenmodell, das in Form einer emulierten virtuellen Maschine realisiert ist, erlaubt es, den run-time Code mit Verweisen auf den Sourcecode zu versehen, so daß in jeder Phase verfolgt

werden kann, welchen Teil des Sourcecodes die Maschine gerade ausführt. Von besonderer Bedeutung sind hierbei Windowing und Multitasking-Mechanismen, die die visuelle Verbindung zwischen Sourcecode und Ablauf schaffen.

Subroutinen-Prozessoren

beträchtlichen Overhead bei tief genesteten Subroutinen, da immer wieder ein Link und Unlink des Stacks stattfinden muß. Es wurden daher schon früh Überlegungen angestellt, den Datenfluß und den Kontrollfluß der Maschine zu entkoppeln. In den 50er Jahren wurde von Samelson und Bauer ein Projekt zur hardware-unterstützten Übersetzung von ALGOL konzipiert. Dieses Projekt, unter dem Namen ALCOR (siehe BAU80) sah eine Maschine mit zwei Kellern, heute auch Stacks genannt, vor. Einer

und Modula basierten auf Maschinen mit einem Stack, um das Nesting/Unnesting von Subroutinen zu realisieren. Erst heute sind SP-Maschinen in Hardware realisiert worden (MALI87).

Essentielle Vorteile von Subroutinen-Prozessoren

Multiple-Stack-Maschinen haben

Hypertext und Programmierung

Hypertext ist wie ein normaler Text eine Folge von Worten. Es können aber in jedem Wort Verweise auf andere Texte existieren. Der Leser kann, wenn er will, zuerst den Text ohne die Verweise lesen, und den Inhalt des Textes ohne Abbruch des Leseflusses verstehen. Wenn er aber daran interessiert ist, zu einem bestimmten Thema mehr zu erfahren, "expandiert" er das Wort, über das er mehr erfahren möchte.

Hypertext ist für die Programmierung ein besonders geeignetes Medium. Ein Programm besteht ja praktisch nur aus Verweisen von Text-Teilen hier auf Text-Teile dort. Subroutinen werden

Abb. 1

Dies ist ein Hypertext-Window im Basis-Modus. Der Text erscheint als normaler Fließtext. Die unterstrichenen Worte enthalten Markierungen für das Hypertext-System und zeigen an, daß hier noch weiterer Text vorhanden ist. Der Benutzer kann durch Drücken einer Kontroll-Taste diesen Text hervorholen: Siehe Abb. 2.

Bild 1:Hypertext und Programmierung

Ein Subroutinen-Prozessor (SP) ist ein Maschinenmodell, das speziell für den effizienten Aufruf von Subroutinen optimiert ist. Heutige Prozessoren haben im allgemeinen ein Register, das für den Maschinen-Stack reserviert ist. Programmiersprachen benutzen diesen Stack, um sowohl die Return-Adressen der Subroutinen als auch die Parameter auf diesem Stack zu verwalten (stack framing). Dies führt zu einem

dieser Stacks war ein Operator-Stack, der andere war für die Operanden bestimmt. Diese Maschine wurde nicht gebaut, da die Programmiersprachen der ersten Stunde, FORTRAN und COBOL, ganz auf das lineare Computer-Modell der bekannten Von-Neumann-Prozessoren aufbauten (Die IBM-360 Serie hatte keinen Stack). Compiler für spätere block-orientierte Programmiersprachen, wie PASCAL, C

den besonderen Vorteil, daß die Subroutinen-Kontrolle von der Parameter-Übergabe zu trennen ist, was einerseits einen Geschwindigkeitsgewinn vor allem beim Nesting von Subroutinen bringt, und andererseits leichter logisch zu verwalten ist. Für die in einem Compiler implementierten Mechanismen ist die Subroutinen-Verwaltung auch mit einem ein-Stack-Modell zu bewältigen. Der Unterschied zeigt sich jedoch im

"human interface". SP-Maschinen sind wesentlich leichter direkt in Subroutinen-Hierarchien programmierbar. Subroutinen-Hierarchien sind besonders wichtig für eine effiziente Decomposition von Programmen. Je einfacher und transparenter dies für den Programmierer machbar ist, desto effizienter ist seine Arbeit. Dies ist der Fall mit der F-Code Maschine, sie kann also als Interpreter genutzt werden, und stellt damit ein direktes Interface dar, mit dem der Programmierer die Möglichkeit hat, den erstellten Code auf jedem beliebigen Level des Details auszutesten. Dies ermöglicht eine wirklich modulare Konstruktion von Programmen und die Verwendung von Software-Bausteinen, die auch experimentell entworfen und modifiziert werden können, die weiterverwendet oder verworfen werden.

Das Selbst-Expertensystem

Wichtigster Bestandteil von komplexen Computer-Systemen ist die Auskunftsfunktion über sich selbst. Diese Funktion steckt noch ganz in den Kinderschuhen. Bisher war es paradoxerweise so, daß die modernsten Computerfunktionen mit dem ältesten aller Informationsmedien dokumentiert wurden: mit Papier. Die Gründe dafür sind vielfältig, hauptsächlich ökonomisch, da Computer bisher zu teuer waren, als daß man für solche Extras wie Benutzerhilfen Maschinen- und Programmiererzeit abzweigen konnte. (Ein wesentlicher ökonomischer Faktor ist eben auch, daß nur bei sehr kryptischen Computersystemen die Programmierer eine Art Priesterschaft bilden konnten, die als geschlossener Zirkel sehr wenig Ansätze boten, von außen Kontrolle auszuüben, und etwa billigere Arbeitskräfte einzustellen oder andere Arbeitsmethoden einzuführen. Dadurch befindet sich heute die gesamte Groß-EDV in einer Sackgasse.) Mittlerweile hat eine Trendwende eingesetzt. Heute ist die Auskunftsfunktion der Kernbestandteil komplexer Systeme, auch wenn das noch nicht überall erkannt worden ist.

LEIBNIZ wurde von Anfang an um diese Auskunftsfunktion herum entwickelt. Sie stellte sich aus der Erfahrung schnell als das wesentliche Essential des Systems heraus, da LEIBNIZ selbst ein Software-Produkt von hoher Komplexität ist, dessen Einzel-

Hypertext und Programmierung

Hypertext ist wie ein normaler Text eine Folge von Worten. Es können aber in jedem Wort Verweise auf andere Texte existieren. Der Leser kann, wenn er will, zuerst den Text ohne die Verweise lesen, und den Inhalt des Textes ohne Abbruch

Die Worte, die Verweise enthalten, werden markiert (hier mit Unterstreichung). So können zu jedem Wort Anmerkungen in beliebiger Länge gemacht werden, und es läßt sich damit auch Graphik, Video, oder Tonmaterial verwalten.

irgendwo definiert und in Libraries aufbewahrt, und anderswo

Abb. 2

Hier haben wir mit der Kontroll-Taste den nächsten Level des Textes geholt, der unter "Verweisen" im Hypertext liegt. In dem neuen Window kann der Benutzer genauso wie im Ausgangswindow editieren und zwischen den Windows hin- und herwechseln. Existieren in diesem Level weitere Verweise, kann in diese hineingegangen werden. Es ist sogar möglich, da man in den obersten Level zurückverwiesen wird. Die Zahl und Richtung der Verweise ist völlig unbeschränkt und nur von den Bedürfnissen des Benutzers abhängig. Bei Beendigung der Arbeit in dem Unter-Window wird dieses geschlossen und man arbeitet im ersten Window weiter.

Bild 2:Hypertext und Programmierung

heiten bei der in der Entwicklung auftretenden schnellen Folge der Modifikationen mit konventionellen Methoden kaum geupdated werden konnten, zumindest aber nicht zu den vom Projektrahmen gegebenen Kosten. Daher wurde diese Funktion auch als Expertensystem (über LEIBNIZ selbst) installiert, was man als Selbst-Expertensystem bezeichnen kann. Als Fact-Base dient hierbei der Source Code und die Dokumentation. Als Procedure-Base der compilierte Code oder die ablauffähigen Module und Prozeduren. Wesentlicher Arbeitsfaktor ist vor allem der Rückverweis des compilierten Codes auf Sourcecode und Dokumentation. Es läßt sich sagen, daß ein Software-Engineering-System auf jeden Fall auch ein Selbst-Experten-System sein muß. Die Phasen von Entwicklung, Dokumentation und Training können nicht getrennt werden, sondern müssen vom System in integraler Weise bewältigt werden.

Dies läßt sich in einem kurzen Vergleich von Charakteristiken subsummieren:

Ein Lehrsystem: "What causes have what effects?"

Ein Expertensystem: "What effects have what causes?"

Hieran kann man leicht sehen, daß beide Systeme sozusagen zwei Seiten derselben Münze sind. Als dritte Komponente kann noch hinzugefügt werden:

Ein Entwicklungs-System: "HOW do causes cause effects?"

"What-If" und "Rapid Prototyping"

"What-If" Anfragen sind natürlich in einer Software-Engineering-Umgebung durch Zusammenbau geeigneter Modul-Bausteine und ihr Austesten zu klären. Für diese Zwecke steht die F-Code Maschine als prozentualer Teil zur Verfügung, der die Klärung von What-If unter dem Namen Modellbildung und Rapid Prototyping leistet. Es ist leicht zu sehen, daß die Nutzung eines solchen Entwicklungssystems auch die Ausweitung des Selbst-Expertensystem-Konzepts auf andere als Software-Engineering-Bereiche erlaubt, da die Fact- und Procedure-Basis genausogut manipulierbare Modelle anderer Systeme aus ganz anderen Bereichen des Engineering sein können (z.B. Molekular-Simulationen, oder elektrische oder thermodynamische Modelle).

Schlußfolgerungen

Software-Engineering-Systeme, die auf der Basis der ökonomischen Tendenzen der Personal-Computer-Technologie ansetzen, wählen andere Realisationswege als solche aus dem Groß-EDV Bereich. Groß-EDV-Systeme, so hat die Erfahrung gezeigt, lassen sich nicht auf Personal-Systeme übertragen. Über das Workstation-Konzept jedoch lassen sich die Erfahrungen und Produktivitäts-Vorteile des Single-user dedicated Systems sehr wohl in den Bereich der Groß-Organisationen mit ihren vielfach größeren Komplexitäten profitabel übertragen.

Liste der genannten Software-Produkte

LEIBNIZ
Software-Produktivitäts-System
Kontakt:
EDV-Beratung - Software-Design
A. Goppold
Hauptstr. 31 a
D-8170 Arzbach
Tel. 08042-4343

INMOS Occam-Editor
INMOS GmbH
Danziger Str. 2
8057 Eching
Tel. (089) 319 10 28

MAXTHINK
230 Crocker Ave.
Piedmont, CA 94610
USA
Tel. (415) 428-0104

Quick Basic / CodeView Debugger für Microsoft-C
Microsoft Corporation
16011 NE 36th Way
Box 97017
Redmond, WA 98073-9717
USA

Think Tank
Living Videotext, Inc.
2432 Charlexton Road
Mountain View
California 94043
Tel. (415) 964-6300

Turbo Pascal
Borland International
4585 Scotts Valley Drive
Scotts Valley
CA 95066
Tel. (408) 438-8400

Bibliographie

ACM84 ACM SIGPLAN Notices "Proceedings of the ACM SIGSOFT/SIGPLAN", Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, Pennsylvania, April 23-25, 1984

BAU80 Bauer, Friedrich L. "Between Zuse and Rutishauser - The Early Development of Digital Computing in Central

Europe",in: A History of Computing in the Twentieth Century, N. Metropolis, J. Howlet, Gain, Carlo Rota, Eds., Academic Press 1980

BROO75 Brooks, F. P "The Mythical Man-Month", Addison-Wesley, Reading, Mass, 1975

DIJK68 Dijkstra, E.W. ""Goto statement considered harmful", Comm. ACM, 11(3), 147-8

GOLZ87 Golze, Andreas; Morbach, Oliver "Der Thadhani-Effekt", Diplom-Arbeit, Uni BwM ID-18/87, Universität der Bundeswehr München, Fakultät für Informatik

HOF79 Hofstadter, Douglas "Gödel, Escher, Bach", Basic Books, 1979

IPE Interactive Programming Environments, David R. Barstow, Howard E. Shrobe, Eds., McGraw-Hill, (1983?)

KAY84 Kay, Alan "Scientific American, October 1984

MALI87 Malinowski, Chris W. "F.O.R.C.E.", Implementing High Speed FORTH Processors Using Standard Cell Technology, Semiconductor Division, Harris Corporation, Melbourne, Florida, USA (Harris Semiconductor GmbH, Erfurter Str. 29, 8057 Eching)

MOLZ87 Molzberger, Peter "Möglichkeiten computergestützter Biofeedback-Technik", Institut für systemorientierte Informatik der Fakultät für Informatik der Universität der Bundeswehr München, München 1987

NEL65 Nelson, Theodor H. "A File Structure for the Complex, the Changing, and the Indeterminate.", Proc. 1956 ACM National Conference

NEL80 Nelson, Theodor H. "Replacing the Printed Word: A Complete Literary System", Proc. 1980 IFIP World Computer Conference

NEL87 Nelson, Theodor H. "Literary Machines", Project Xanady, 1987

SOM87 Sommerville, Ian "Software Engineering", Addison-Wesley Publ. Co., Bonn, Reading,Mass., etc. 1987

THAD81 Thadhani, A.J. "Interactive user productivity" in: IBM Systems Journal, Vol. 20, No. 4, 1981, p. 407-423

THAD84 Thadhani, A.J. "Factors affecting programmer productivity during application development" in: IBM Systems Journal, Vol. 23, No. 1, 1984, p. 19-35

WEIZ76 Weizenbaum, Joseph "Computer Power and Human Reason", Freeman, San Francisco 1976

Publikationen von A. Goppold

GOP77 "Artificial Intelligence and Learning", Diplom-Arbeit, Institut für Informatik, Universität Hamburg, Juni 1977, 204 Seiten: Exkurs in die verschiedenen theoretischen und philosophischen Modelle der Lernpsychologie, Behaviorismus (Skinner), Gestalt-Psychologie (Köhler, Koffka, Wertheimer), Developmental (Piaget), Linguistik (Chomsky), Neurologie. -Computer-Modelle des Lernens, einfache Lern-Automaten. -Untersuchung diverser Lern- und Lehrsysteme (Scholar, Plato), Diskussion der AI-Ansätze im Lernen, Kritiken der AI (Dreyfus, Weizenbaum) -Skizze eines lernenden Computersystems auf der Basis von Poly-Automaten (multiprozessor-System) bzw. zellulare Räume

GOP84-1 "Das Paradigma der interaktiven Programmierung", Erschienen in Computerwoche, 24.8. und 31.8.84: Ein neues Paradigma macht sich in der Programmierung bemerkbar. Ausgehend von dem Modell der Benutzerunterstützung, wie es ursprünglich an XEROX PARC von Allan Kay unter dem Namen Smalltalk konzipiert wurde, und in abgemagerter Form auf APPLEs LISA und Macintosh angeboten wird, werden die Prinzipien des interaktiv unterstützenden Programmiersystems aufgestellt. Als Beispiel der Möglichkeiten dient FORTH.

GOP84-2 "FORTH kommt interaktiver Programmierung entgegen", Computerwoche, 14.9.84, S. 14-18: Überblicksartige Beschreibung von FORTH. Vergleich mit der Programmiersprache C, die von ihrer Struktur und ihrem Anwendungsbereich mit einigen Aspekten von FORTH übereinstimmt.

GOP84-3 "Trends in der Entwicklung der Programmiersysteme (Die stille Software-Revolution)", Vierte Dimension Vol_1/No_1, No_2, Okt. 1984, Forth Gesellschaft Deutschland, Friedensallee 92, 2000 Hamburg 50: Fortsetzung zu: Paradigma der

interaktiven Programmierung. Es wird die voraussichtliche weitere Entwicklung der Programmiersysteme skizziert. Die neuen Entwicklungen in MAC-BASIC und Verwandten, zeigen die schon vorhandenen Umwälzungen der Programmierszene. FORTH ist im weiteren das Beispiel für ein Programmiersystem, das auf Grund seiner maximalen Flexibilität alle noch zu erwartenden Entwicklungen der Programmiersysteme unterstützen wird.

GOP85 "Forth: Ein Programmiersystem ohne Grenzen", Edition Aragon, 1985, 162 Seiten: Einführung in das Programmiersystem Forth von einer phänomenologischen Perspektive. Beschreibung der hierarchischen Programmierung in Forth, Anwendung der Russellschen Typenhierarchien auf Programmierung

FORTH-83

MULTITASKING MODEM-Paket

Jeffrey R. Teza - Encinitas, Kalifornien
übersetzt von D.Luda aus 'FORTH Dimensions' Vol IX, #6

Es wurden ja schon einige Modem-Programme in der 'FORTH Dimensions' veröffentlicht (JAM85), (ERI84), (ACK83). Sie stellten in Form von guten Beispielen die Grundlagen für die Programmierung der seriellen Schnittstelle dar. Mit diesem Wissen ist es nun möglich, den etwas komplizierteren Terminalemulator zu verstehen, der einerseits ein nützliches Terminal-Paket darstellt und andererseits als Beispiel für eine FORTH-Multitasking Applikation gelten kann.

Eine gute Einsatzmöglichkeit für einen Computer, mit dem man ein Terminal emuliert, ist die Möglichkeit vor Ort anfallende Aufgaben auszuführen. Kleinere Dinge, wie z.B. automatisches Wählen, Telefonregister und das Ein-/Ausladen von Dateien können durch einen Tastendruck erledigt werden. Dadurch wird das Arbeiten mit einem langsamen Modem etwas angenehmer. So etwas zu programmieren kann sehr riskant sein, da die Echtzeit-Natur eines Modem-Paketes verlangt, daß ein lokaler Prozeß innerhalb einer Zeichenübermittlungszeit zu Ende geführt wird. Wird diese Beschränkung nicht eingehalten, kann es sein, daß das Modem ankommende Daten verliert.

Das asynchrone FORTH-Multitasking gewährleistet eine sehr schnelle Verbindung zwischen den einzelnen Tasks. Oft handelt es sich dabei nur um einige Maschineninstruktionen, und das Ganze kann so schnell sein wie eine High-Level-FORTH-Schleife. Ein Terminalemulator ist meistens als eine unendliche Schleife definiert, welche Zeichen vom Modem zur Konsole und umgekehrt weiterleitet. Der Quelltext in Screen 9 zeigt zwei Tasks, die prinzipiell als eine BEGIN...AGAIN-Schleife geschrieben werden könnten. In unserem Fall wird dabei allerdings der FORTH Multitasker benutzt, um die beiden zusammenzufügen. Dadurch, daß ein Task im 'Vordergrund' und einer im 'Hintergrund' abläuft, bietet diese Struktur einen Vorteil für den Terminalemulator: Der KEYBOARD-Task hat dadurch etwas Zeit andere Funktionen zu erfüllen, während der MODEM-Task (Hintergrund) weiterhin aufpaßt, ob irgendwelche Zeichen am seriellen Port ankommen.

Was ist nun mit den vom Modem kommenden Zeichen zu tun, während der KEYBOARD-Task andere Funktionen ausführt? Schauen Sie sich dazu die Screens 3 und 4 an. Dort wird ein 'zuerst-rein zuerst-raus' Puffer erzeugt, der es den beiden Tasks erlaubt, miteinander zu kommunizieren. Ankommende Zeichen, die darauf warten müßten, daß der KEYBOARD-Task eine Funktion beendet, werden in diesem FIFO-Puffer abgespeichert und können dann später von KEYBOARD aufgenommen und dem Anwender gezeigt werden.

Durch den FIFO-Puffer haben wir ja nun etwas Zeit gewonnen. Es ist nicht mehr notwendig, die Schleife möglichst rasch zu durchlaufen, um ankommende Zeichen in Empfang zu nehmen. Was sollen wir mit all' dieser Zeit anfangen? Vieles ist denkbar, einige Möglichkeiten finden Sie im Beispiel. Screen 8 erzeugt eine Sprungtabelle, die bemerkt, wenn eine Kontrolltaste auf dem Keyboard gedrückt wurde und dann den KEYBOARD-Task entsprechend umleitet. Ich habe ein paar nützliche Tools für Anwender geschrieben, die ein Modem benutzen, um mit einem anderen Computer zu kommunizieren.

Was sollen wir mit all' dieser Zeit anfangen?

Das erste habe ich einem eleganten kleinen Programm entnommen, das von Leo Brodie publiziert wurde [BRO83]. Es handelt sich um einen Breakpoint-Interpreter, der eine FORTH QUIT-Schleife (shell) startet. Dadurch ist es dem Anwender möglich, in den FORTH-Interpreter zu springen und so "über" der Modem-Software zu arbeiten.

Der zugehörige Code ist in den Screens 5 und 6 zu ersehen und der Einsprung in diesen Programmteil wurde in der Sprungtabelle

an der Stelle für ASCII 6 (ctrl-F Taste) eingetragen. Somit haben Sie vollen Zugriff auf das FORTH-Dictionary und dadurch eine adäquate Möglichkeit die einzelnen Tools auszuführen.

Ein anderes Tool (Screens 7 und 11), stellt eine Telefonnummernliste dar. Diese Nummern können entweder einer Taste zugewiesen werden und vom Modem mit Hilfe eines einzigen Tastendruckes automatisch gewählt werden. Oder es ist auch möglich, die Nummern in einem Vokabular (PHONE) abzuspeichern und sie später vom Breakpoint-Interpreter aus ausführen (wählen) zu lassen.

Ich glaube, daß das eine ganz nützliche Applikation ist und habe mich deshalb bemüht, sie mit verständlichen Kommentaren zu versehen. Das End-Anwender Wort ist CONVERSE. Es nimmt eine Baudrate als Parameter und startet die beiden Prozesse (geben Sie z.B. 1200 CONVERSE ein).

Noch ein paar Bemerkungen zum verwandten FORTH-Dialekt. Der vorliegende Code ist in Laxen und Perry's F83 geschrieben. Ich habe versucht, jeden Code, der nicht dem 83er Standard entspricht, in den Shadowscreens zu kommentieren. Aber das Wort BACKGROUND: sollte bei Bedarf entsprechend Ihres Multitasking-Wortschatzes abgeändert werden. Ich möchte an dieser Stelle die Leute, die keinen Multitasker haben, um Entschuldigung bitten. Manche Dinge können auf ein Single-Task-System übertragen werden. Aber in Anbetracht dessen, daß der FORTH-Multitasker einfach aufgebaut ist und die Einführung von Henry Laxen [LAX84][LAX83] diesbezüglich wirklich gut verständlich ist, sollten ernsthafte FORTH-Anwender erwägen, diese sehr wichtige Fähigkeit Ihrer FORTH-Umgebung unbedingt hinzuzufügen.

Dieses Programm lief auf meinem 8 MHz 80186 System bei einer Baudrate von 1200 am besten. Bei langsamen PAUSE-Schleifen auf langsamen Geräten, können verlorene Zeichen immer noch Probleme bereiten. Alles, was ich dazu sagen kann ist, daß der Zeichenempfänger des Modems in Assembler geschrieben werden sollte oder daß er Interrupt-gesteuert werden sollte. Wie hier demonstriert wurde, stellt die Geschwindigkeit von optimierten FORTH-Multitasking-Schleifen eine gute Alternative zu 'High-Level'-FORTH-Schleifen dar. Der einzige Nachteil liegt darin, daß langsame Computer sogar bei einfacheren Terminalprogrammen ein bißchen Assemblercode benötigen.

Literaturangaben:

[ACK83] Ackermann, R.D. "Apple FORTH à la Modem" FORTH Dimensions, Vol. 5 Nr. 4, Nov./Dez.1983.

[BRO83] Brodie, Leo. "Add a Breakpoint Tool" FORTH Dimensions, Vol. 6 Nr. 2, Mai/Juni 1983

[ERI84] Ericson und Feucht. "Simple Data Transfer Protocol" FORTH Dimensions, Vol. 6 Nr. 2, Juli/Aug. 1984.

[JAM85] James, John S. "Simple Modem I/O Word" FORTH Dimensions, Vol. 6 Nr. 5 Jan.Feb. 1985.

[KNU73] Knuth. "The Art of Computer Programming, Fundamental Algorithmus" Vol I. Addison-Weseley, 1973.

[LAX83] Laxen, Henry. "Multitasking" Teil 1, FORTH Dimensions, Vol. 5 Nr. 4 Nov./Dez. 1983.

[LAX84] Laxen, Henry. "Multitasking" Teil 2, FORTH Dimensions, Vol. 5 Nr. 5 Jan./Feb. 1984.

MULTITASKING MODEM-Paket

```
      1
 0 \ Dumb Terminal   load block                          26May85jrt
 1 VOCABULARY TALKING    ONLY FORTH ALSO TALKING ALSO
 2 TALKING DEFINITIONS
 3 1 8 +THRU   \ dumb terminal emulator
 4
 5 FORTH DEFINITIONS
 6 9 +LOAD     \ CONVERSE,QUIET
 7
 8 ONLY FORTH ALSO
 9 CR .( Dumb terminal emulator loaded. )
10
11
12
13
14
15
```

```
      2
 0 \ Dumb terminal   Hardware specific words             19Mar85jrt
 1 HEX
 2 92 CONSTANT STATUS
 3 96 CONSTANT DATA
 4
 5 : INITIALIZE (S baud--) 12C ( 300) =
 6    IF 44 STATUS PC! ELSE ( 1200) 66 STATUS PC! THEN ;
 7
 8 : KEYM? (S --f)  STATUS PC@ 1 AND 0<>  ;
 9 : KEYM (S --c)   PAUSE  BEGIN  KEYM? UNTIL  DATA PC@ ;
10 : EMITM (S c--)
11    PAUSE  BEGIN STATUS PC@ 4 AND 0<> UNTIL  DATA PC! ;
12
13 DECIMAL
14
15
```

```
      3
 0 \ Dumb Terminal   FIFO queues                         26May85jrt
 1 80 24 * 5 * CONSTANT QDEPTH \ 5 Page buffer
 2 : 4+  2+ 2+ ;
 3 \ Create 2 queues with front and back pointers
 4 CREATE INCOMING  QDEPTH 4+ ALLOT
 5 CREATE OUTGOING  QDEPTH 4+ ALLOT
 6
 7 : OQUEUE (S qaddr--) DUP 4+ DUP  ROT 2! ;
 8 INCOMING OQUEUE  OUTGOING OQUEUE  \ initialize the 2 queues
 9
10 \ Increment a queue pointer
11 : +QUEUE (S qaddr paddr--paddr) OVER QDEPTH 4+ +  OVER @ =
12    IF  SWAP 4+ OVER !  ELSE  NIP  1 OVER +!  THEN ;
13
14
15
```

```
      4
 0 \ Dumb Terminal   queue i/o                           17Mar85jrt
 1 : OVERFLOW (S c qaddr--)  INCOMING =
 2    IF   CR ." ERROR...Incoming queue overflow "
 3    ELSE CR ." ERROR...Outgoing queue overflow " THEN DROP ;
 4
 5 : !QUEUE (S c qaddr--) DUP +QUEUE
 6    DUP 2@ =  IF  OVERFLOW  ELSE  @ C!  THEN ;
 7
 8
 9
10
11 : @QUEUE (S qaddr--c true!false)
12    DUP 2@ =  IF DROP FALSE EXIT THEN \ ?underflow
13    DUP 2+  +QUEUE  @ C@ TRUE ;
14
15
```

MULTITASKING MODEM-Paket

```
     5
 0 \ Dumb Terminal   breakpoint interpreter            26May85jrt
 1 \ See FD vol5/#1 pp19
 2 VARIABLE CHECK
 3 : BREAK ( --)  \ invokes QUIT shell
 4    CR  RP@ 4 ( mvp=6) -  CHECK ! 0 BLK !
 5    BEGIN   QUERY  INTERPRET  ." ok" CR AGAIN ;
 6
 7 FORTH DEFINITIONS
 8 : RESUME ( --) ['] (?ERROR) IS ?ERROR \ resume normal abort
 9    RP@ CHECK @ =  \ aborts top QUIT shell
10    IF  R> R> ( mvp R> ) 2DROP ( mvp DROP) CR
11    ELSE ." Can't resume" QUIT THEN ;
12
13 TALKING DEFINITIONS
14
15
```

```
     6
 0 \ Dumb Terminal   local processing                  26May85jrt
 1 : (?BRKERROR) (S addr len f--)
 2    IF  >R >R  SP0 @ SP!  PRINTING OFF
 3       R> R> SPACE TYPE SPACE  ELSE 2DROP THEN ;
 4
 5 : BREAK  2DROP  ['] (?BRKERROR) IS ?ERROR  BREAK ; \ ctrl F
 6 : BEEP   2DROP  BEEP ;                      \ ctrl G
 7 : PRINT  2DROP  PRINTING @ NOT PRINTING ! ; \ ctrl P printer
 8
 9 : QTYPE (S addr len qaddr--) ROT ROT
10    BOUNDS DO I C@ OVER !QUEUE PAUSE LOOP  DROP ;
11 : QCR (S qaddr--)   13 SWAP !QUEUE ;
12
13
14
15
```

```
     7
 0 \ Dumb Terminal   Autodialing phone numbers         28May85jrt
 1 : DIAL (S addr len--) OUTGOING QTYPE  OUTGOING QCR ;
 2
 3 FORTH DEFINITIONS
 4 : BOOK   WORDS ;
 5 VOCABULARY PHONE   PHONE DEFINITIONS
 6
 7 11 LOAD \ Phone numbers
 8 : CALL"   HEADER ASCII " WORD COUNT DIAL ;
 9
10 TALKING DEFINITIONS
11
12
13
14
15
```

```
     8
 0 \ Dumb Terminal   local escape table                26May85jrt
 1
 2
 3 CREATE FILTERTABLE  ]
 4 !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  BREAK   BEEP
 5 !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE
 6 PRINT   !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE
 7 !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE  !QUEUE [
 8
 9
10 : FILTER (S c qaddr--) OVER 32 <
11    IF  OVER 2* FILTERTABLE +  PERFORM  ELSE  !QUEUE  THEN ;
12
13
14
15
```

```
     9
 0 \ Dumb Terminal    keyboard/modem tasks             17Mar85jrt
 1
 2 BACKGROUND: MODEM
 3    BEGIN  OUTGOING @QUEUE  IF EMITM THEN
 4       KEYM? IF  KEYM INCOMING !QUEUE  THEN  PAUSE
 5    AGAIN  ;
 6
 7 : KEYBOARD
 8    BEGIN  INCOMING @QUEUE  IF EMIT THEN
 9       KEY? IF  KEY OUTGOING FILTER THEN  PAUSE
10    AGAIN  ;
11
12
13
14
15
```

```
    10
 0 \ Dumb Terminal    converse/quiet                   26May85jrt
 1 \ Converse invokes the dumb terminal
 2 : CONVERSE (S baud--) INITIALIZE  INCOMING OQUEUE
 3    OUTGOING OQUEUE  MULTI  MODEM WAKE  KEYBOARD ;
 4
 5 : QUIET (S --)  MODEM SLEEP   ['] (?ERROR) IS ?ERROR  ABORT  ;
 6
 7
```

```
    11
 0 \  Phone Numbers                                    26May85jrt
 1 : HEADER  " ATDT " OUTGOING QTYPE ;
 2 : FIG   HEADER " 4155383580" DIAL ;
 3
 4 -->
 5
 6
 7
```

```
   18
\                                                    17Mar85jrt
This is the load block for the dumb terminal emulator.

This dumb terminal emulator uses the Laxen & Perry F83
   multitasking capabilities by defining separate keyboard and
   modem tasks. These two tasks communicate via a FIFO queue.
   This structure allows local processing without loosing
   characters.

A convenient technique for invoking local processing words is
   used via a control character table as in F83. One local
   word that can be invoked in this example is a breakpoint
   interpreter as published by Leo Brodie. This allows you
   to exit to a higher forth QUIT "shell" giving you complete
   access to the forth dictionary while terminal emulating.


   19
\                                                    17Mar85jrt
This screen contains all of the hardware specific code to
   talk to the modem port. These words are analogous to
   FORTH i/o words.


   20
\                                                    26May85jrt
QDEPTH  is the depth of the FIFO queues. This depth should
   be adjusted according to how smooth the multitasking loop
   is running ( total task activity ). Note that at
   1200 baud a character comes in every MS or so and without
   interupt driven modem control the loop must average shorter
   than this to avoid queue overflow and lost characters.
INCOMING and OUTGOING are both byte queues of QDEPTH length
   whose first cell is a pointer to the front of the list and
   second cell is a pointer to the back of the list.
OQUEUE initializes a queue so the two pointers point at the
   same queue entry.
+QUEUE  increments a queue pointer circularly.
   ( NIP is SWAP DROP )


   21
\                                                    17Mar85jrt
OVERFLOW  issues an error message for an overflowed queue.


!QUEUE  puts a byte at the front of a queue. It first increments
   the front pointer then checks for an overflow. Note that if
   an overflow occurs it will continue to place characters in
   the queue causing the queue to be dumped by incrementing
   the front pointer past the rear pointer.

@QUEUE  removes a byte from the back of the queue if one is
   available and returns either the character and a true or
   a false flag if the queue was empty.


   22
\                                                    26May85jrt
This breakpoint interpreter was published by Leo Brodie in
   FD vol5 no1. MVP-FORTH changes are shown as inline comments.
BREAK  invokes an outer interpreter or "shell".


RESUME  is used to resume from the BREAK shell. If the return
   stack is messed up use KEYBOARD to restart.


   23
\                                                    26May85jrt
(?BRKERROR)  ?ERROR is the F83 vectored ABORT error handler.
   this word is used to return to the BREAK shell on errors.


BREAK calls the breakpoint interpreter after cleaning up
  the stack and vectoring the new error handler.
BEEP  rings the bell after cleaning up the stack.
PRINT  toggles the printer on/off in an F83 system.
QTYPE  types a string to a queue.

QCR  puts a carriage return ( ascii 13) in a queue.


   24
\                                                    26May85jrt
DIAL  types a string to the outgoing queue followed by a cr.

Put the PHONE vocabulary in the FORTH vocabulary.
Use PHONE BOOK to see the phone numbers.

Put the phone numbers in the PHONE vocabulary.
   Use PHONE <Phone#> e.g. PHONE FIG to dial a number.
Use CALL" XXXXXXX" to call a number not in the PHONE BOOK.

Back to the application vocabulary.
```

```
   25
\                                                    17Mar85jrt

FILTERTABLE is a table indexed into by a control character.
   Note it contains 32 entries which can be any FORTH word
   which will be invoked by a control character pressed at the
   keyboard.


FILTER  takes a character and queue address and looks up
   control characters in FILTERTABLE for execution, otherwise
   sticks it in the queue. ( PERFORM is @ EXECUTE)
```

```
   26
\                                                    17Mar85jrt

MODEM  is a background task which gets outgoing characters
   and writes them to the modem port and incoming characters
   and writes them in the incoming queue.


KEYBOARD is the terminal task which gets incoming characters
   and prints them. Keyboard entered characters are FILTER'd
   which either does something or sends them to the outgoing
   queue.
```

```
   27
\                                                    26May85jrt

CONVERSE  takes a baud rate as a parameter, initializes the
   modem port, zero's the queues, fires up the multitasker
   and enters the KEYBOARD infinite loop .

QUIET  puts the modem task to sleep and aborts the system.
```

```
   28
\                                                    19Mar85jrt
Phone numbers can be entered into the control character table
   or defined to be executed while in the break shell.
```

'Peerless' Namen in FORTH

(peerless = einzigartig, eindeutig)

R. Jones

Deutsche Übersetzung: D.Luda

Einleitung

FORTH ist keine standardisierte Sprache. Die Sprachen, welche FORTH-ähnlich sind, enthalten immer etwas, das sie erheblich von z.Bsp. FORTH-83 unterscheidet. Einige der weniger standardisierten FORTH-Dialekte ermöglichen einen verständlicheren Gebrauch von Namen und Definitionsworten als das Standard-FORTH. Wenn die Konventionen für Definitionsworte in FORTH in eine einheitlichere Form gebracht werden, ist es möglich, 'Peerless' Namen zu definieren. Der vorliegende Beitrag untersucht, wie und warum solche Veränderungen in FORTH bei der Implementation und in der Praxis nützlich sind.

Der FORTH-Familienstammbaum

Ting zeigte (in Tin84) einen Stammbaum der FORTH-Dialekte. Dieser Stammbaum - abgeändert in eine stilisierte Landkarte (siehe Bild 1) - siedelt das Standard-FORTH (FORTH-83) westlich an, die Ursprünge von FORTH in der unteren Mitte und Stoic östlich:

Gibt es auch einen identifizierbaren, östlichen FORTH-Dialekt? Stoic (MA-Massachusets), RTL (MI-Missouri), Pistol (PA-Pennsylvannia) und Reptil (OH-Ohio) bilden eine Untergruppe der FORTH-Dialekte und befinden sich entweder ganz östlich oder außerhalb der Grenzen der FORTH-Zivilisation. Der vorliegende Beitrag behandelt einige der Charakteristika und der Vorteile dieser östlichen FORTH-Dialekte. Der Hauptvorteil liegt wohl in der Handhabung von Namen in östlichen FORTH-Dialekten.

Übernommene Familiencharakteristika

Östliche FORTH-Dialekte besitzen einige typische Merkmale die allen FORTH-Dialekten zu eigen sind. Wie z.Bsp. die UPN-Schreibweise und der Einsatz eines Parameter- und eines Returnstacks. Es gibt jedoch auch verschiedene Abweichungen, die das östliche FORTH von den meisten anderen FORTH-Dialekten unterscheidet:

- Informativ-Prompt
- 'Token threaded' Code oder alternativ, 'Status threaded' Code
- "Deferred" Execution
- syntaktische Lesbarkeit
- Elimination des Problems der Handhabung von verschiedenen Quellcodeversionen durch den Decompiler
- die 'TO' Lösung
- Elimination von SMUDGE
- Namen in Anführungszeichen
- verschachtelte Namen

Diese Abweichungen von standardisierteren FORTH-Dialekten führen zu einer homogeneren Sprache. D.h. dieser FORTH-Stil ist leichter zu lesen und handzuhaben (bzw. zu warten). Diese Merkmale werden weiter unten besprochen. Es werden desweiteren Gründe dafür angeführt, warum die Meinung vertreten wird, daß eindeutigere Namen eher durch die Fähigkeiten des östlichen FORTH-Dialekts als durch die des Standard-FORTH unterstützt werden.

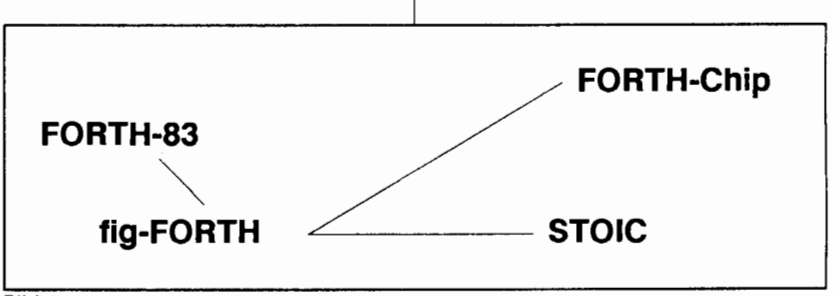


Bild 1

'Peerless' Namen in FORTH

Informativ-Prompt

Das Prompt beim östlichen FORTH enthält Informationen über die Tiefe der Klammernverschachtelung, die Zahlenbasis und die aktuelle Tiefe des Parameterstacks. Das leichter zu lesende Prompt ist der erste Schritt in einer Reihe von Schritten, die zu einer leichteren Les- und Erstellbarkeit von FORTH-Programmen führen.

Status threaded Code

Buege in (Bue84) plaziert Informationen in einem 16-Bit 'Threading'-Wort, welche die Ausführung und Dekompilation steuern. 'Token threading' ist eine Besonderheit von RTL und REPTIL.

Klammern in FORTH

Glass und Bergmann, in (Gla83), (Ber83) und (Ber84), führen Klammern und Kommata in FORTH ein, damit die Lesbarkeit verbessert wird. Klammern verändern die UPN-Ausführung von Worten nicht, aber sie ermöglichen die schnelle visuelle Erfassung von Wortgruppen.

"Deferred" Execution

Beim Bergmannschen Schreibstil können Klammern, so wie er sie in seiner Doppelpunktdefinition in (Ber84) definiert hat, dazu benutzt werden, die Ausführung zu verzögern. Obwohl die Kommentare im PISTOL-Code von Bergmann eigentlich nur den Code in Gruppen einteilen, damit er besser zu lesen ist, haben sie in seiner Implementation auch den Effekt, daß sie die Ausführung verzögern.

Lesbarkeit

Fast alle abweichenden Merkmale der östlichen Dialekte erhöhen die Verständlichkeit, vereinfachen das Lesen, Erstellen und die Handhabung von FORTH-Programmen.

Decompiler-Philosophie

Der Versuch von RTL Decompiler so einzusetzen, wie es in (Bue84) beschrieben wird, ergibt ein sehr nützliches Decompiler-Output-Listing und hilft dabei Versionsprobleme in den Griff zu bekommen. 'Token threading' ist eine gute Methode, wie dieser Gebrauch des Dekompilierens eingesetzt werden kann.

Namen in Anführungsstrichen

Mit Hilfe von Namen in Anführungszeichen wird der Einsatz von Namen, wie sie von STOIC (Sac83), PISTOL (Ber83), SPHERE (Sol84), REPTIL (Uri86) und (Uri87) benutzt werden, übersichtlicher. REPTIL geht bei der Nutzung dieser Technik am weitesten. Die vorderen Anführungszeichen werden zum Zugriff auf das Verb-Token benutzt. Auf den Inhalt einer Variablen oder einer Wertzelle wird durch seinen Aufruf zugegriffen.

Die 'TO' Lösung

Die 'TO' Lösung von Bartholdi wird bei diesen Sprachen als Standard genommen. Die Funktionen seiner Namen erhöhen die Verständlichkeit der Arithmetik, wie es aus den Beispielen in (Bar79) zu ersehen ist. Die Unterscheidung zwischen :IS und :HAS in REPTIL, z.Bsp. implementiert den Unterschied zwischen Konstanten und Variablen.

Die Elimination von SMUDGE

Ist SMUDGE der Grund der Unverständlichkeit und der Code-Überprüfungsprobleme? Eine Besprechung von SMUDGE finden Sie bei (SCH82). Der Gebrauch von SMUDGE erübrigt sich, falls man explizite Rekursionen und verschachtelte Namen verwendet. Rekursionen sind in FORTH Standard und können z.Bsp. mit dem Wort MYSELF (siehe dazu LeV80) implementiert werden. Das Wort MYSELF, erlaubt jedoch nicht alle Arten von Rekursionen. So z.Bsp. nicht die zirkulare Rekursion, bei der zwei Worte sich gegenseitig aufrufen. Die östlichen Sprachen der FORTH-Familie ermöglichen den direkten Aufruf eines Wortes innerhalb der Doppelpunktdefinition des Wortes. Beispiele dazu findet man bei REPTIL.

Abonnement

Wir haben vor in Kürze die 'Vierte Dimension' auch im Abonnement anzubieten. Was halten Sie davon? Teilen Sie uns bitte Ihre Meinung dazu mit, damit wir uns ein Bild davon machen können, ob es sich lohnt.

'Peerless' Namen in FORTH

FORTH mit verschachtelten Namen

Bill Ragsdale erkannte in (Rag81) einen Problemkomplex innerhalb von FORTH. Er bemerkte die Abweichungen in FORTH von der bestehenden Postfixnotation und die damit verbundene Probleme mit BUILDS, DOES und SMUDGE.

In (Uri87) wird die Verschachtelung von Doppelpunktdefinitionen im Zusammenhang mit einer Sprache (REPTIL) vorgestellt, die verständlich und gut lesbar ist. Das wurde mit Hilfe des Definitionsverbpaares DO: :END bewerkstelligt, welches widerspruchslose und verschachtelte Definitionen erlaubt.

Das Muster von verschachtelten Doppelpunktdefinitionen sieht folgendermaßen aus (siehe Bild 2):

```
DO:
                DO:
                :END
:END
```

Bild 2

Feldworte

Felddefinitionsworte findet man in (Duf84). Die Beispiele dort zeigen, wie Definitionsworte ohne Verschachtelung dem FORTH-Code hinzugefügt werden können.

Verschachtelte Definitionen

Beispiele von effektiv verschachtelten Definition, welche auch die CREATE-DOES-Konstruktion benutzen, sind im System der Objekt-orientierten Programmierung von Pountain (Pou86) und (Pou87) enthalten. Diese Definitionen veranschaulichen die Nützlichkeit von verschachtelten Definitionskonstruktionen in FORTH.

Das Klassensystem von Kimball in FORTH

Die Einführung Kimballs von Klassen in FORTH (Kim87) zeigt eine FORTH-Implementation von Klassen, die von Smalltalk abgeleitet ist. Seine Definitionen stellen eine Alternative zu den Objekt-orientierten Definitionen von Pountain dar.

Peers

Die DO: :END-Konstruktion ermöglicht die Definition von Namen, die mit einer Verschachtelungsebene verbunden sind. Namen sind Peers, wenn sie sich auf derselben Verschachtelungsebene befinden. Namen sind demnach 'peerless', wenn sich kein Peername innerhalb der Definition befindet, in welcher sie auftauchen.

Lambda-Calculus

Der Lambda-Calculus ist ein Formalismus, mit dem Funktionen und Verschachtelungen formal untersucht werden können. Er ist außerdem der mathematische Hintergrund von LISP. Die Merkmale von RTL, REPTIL, PISTOL und STOIC ermöglichen - durch Aufzeigen des Verschachtelungsphänomens an einer durchschaubaren RTL-Syntax - eine Anwendung dieser leicht zu verstehenden mathematischen Basis. In (Hin86) finden Sie eine Beschreibung des Lambda-Calculus und einige seiner Methoden, wie z.B. Alpha- und Beta-Konversion. Die Alpha-Konversion entspricht der Ausführung eines FORTH-Wortes mit einem Parameterstack als Eingabe.

Alpha- und Beta-Konversion

Der Ablauf der Alpha- und Beta-Konversion wird in (Hin86) beschrieben. Diese Konversionen stellen die Basis für eine Logik dar, die bei Systemen der künstlichen Intelligenz benutzt wird. In (Jon88) befindet sich eine Diskussion, wie die Alpha- und Beta-Konversion bei Planungsproblemen der künstlichen Intelligenz eingesetzt werden könnte.

Applikationen

Applikationen von lesbarem FORTH mit 'peerless' Namen enthalten so gut wie sicher künstliche Intelligenz. Die Innovationen bei der Lesbarkeit wurden von Glass eingeführt und von Bergmann und Ureli weiterentwickelt. Sie sind beim Ausdruck von logischen Bedingungen sehr nützlich und auch bei den unüblichen Formen der Schlußfolgerungen, welche in Zukunft bei der künstlichen Intelligenz eingesetzt werden müssen. Die nicht-monotone Logik wurde schon von (Jon88) als Möglichkeit der Informationsabfrage von einer Datenbank, welche Techniken der künstlichen Intelligenz benutzt, untersucht.

Die Zukunft von lesbarem FORTH

Wie sieht nun die Zukunft von lesbarem (also östlichem) FORTH aus? Eine Objekt-orientierte Version dieser FORTH-Dialekte ist wohl am wahrscheinlichsten. Sehr wünschenswert, aber weniger wahrscheinlich, wäre die Erzeugung eines FORTH-Codes durch die CASE-Technologie, wie es z.B. in (Ric78) beschrieben wird.

Fazit

Es wurde gezeigt, daß ein FORTH-Dialekt entsteht, der in vielen Dingen nicht standardgemäß ist. Dieser Dialekt bietet eine Lösung für einen gravierenden Mangel von FORTH an; nämlich der Tendenz zu einem Schreibstil der kompakt und schwer zu verstehen ist. Innerhalb dieses entstehenden Dialektes entwickelt sich vielleicht eine mathematische Basis für ein leichter verständliches FORTH. 'Peerless' Namen sind ein Nebenprodukt der besseren Strukturierung von Namen und Definitionsworten in diesem Dialekt.

Quellenangaben:

(Bar79) Bartholdi, Paul. The "TO" solution. Forth Dimensions, Vol.1, Nr. 4, 1979, S. 38-40.

(Ber83) Bergmann, Ernest E. PISTOL -- a FORTH-like portably implemented stack oriented language. Dr. Dobb's Journal, Nr. 76, Feb. 1983, S. 12-15.

(Ber84) Bergmann, Ernest E. Languages and parentheses -- a suggestion for Forth-like languages. Dr. Dobb's Journal, Nr. 93, Juli, 1984, S. 102-108.

(Bue84) Buege, R. Status threaded code. Proceedings of the 1984 Forml Conference, S. 251-254.

(Duf84) Duff, Charles B. A group construct for field words. The Journal of Forth Application and Research, Vol. 2, Nr. 3, 1984, S. 83-85.

(Gla83) Glass, H. Towards a more writable FORTH syntax. 1983 Rochester Forth Conference Proceedings, S. 125-132.

(Hin86) Hindley, J. Roger, and Seldin, Jonathan P. Introduction to combinators and lambda-calculus, Cambridge, 1986.

(Jon88) Jones, R.M. Non-monotonic interrogation. In Henson, Troy (Hrsg.) Artificial Intelligence and Simulation, Proceedings of the SCS Multiconference, San Diego, CA, 1988, S. 78-82.

(Kim87) Kimball, Vince D. Classes in FORTH. FORTH Dimensions, Vol. 8, Nr. 5, 1987, S. 24-27.

(LeV80) LeVan, Jerry. The eight queens problem. FORTH Dimensions, Vol. 2, Nr. 1, 1980, S. 6.

(Pou86) Pountain, Dick. Object oriented extensions to Forth. The Journal of Forth Applications and Research, Vol. 3, Nr. 3, 1986, S. 51-73.

(Pou87) Pountain, Dick. Object-oriented Forth. Academic Press, 1987.

(Rag81) Ragsdale, William F. A new syntax for defining words. Forth Dimensions, Vol. 2, Nr. 5, 1981, S. 121-128.

(Ric78) Rich, Charles, and Shrobe, H.E. Initial report on a LISP programmers apprentice, IEEE Transactions on Software Engineering, Vol. 4, Nr. 6, Nov. 1978. Reprinted in David Barstow, E. Sandewall, und H. Shrobe (Hrsg.) Interactive Programming Environments, McGraw-Hill, 1984.

(Sac83) Sachs, J und Burns, S. K. STOIC an interactive programming system for dedicated computing. Software Practice and Experience, Vol. 13, 1983, S. 1-16.

(Sch82) Schleisiek, Klaus. What's wrong with SMUDGE? 1982 FORML Conference Proceedings, S. 47-49.

(Sol84) Solley, Evan L. SPHERE: An in-circuit development system with a FORTH heritage. 1984 Rochester Forth Conference Proceedings, S. 25-31.

(Tin84) Ting, C. H. Inside F83, Offete Enterprises, San Mateo, CA, 1984.

(Uri86) Urieli, Israel. REPTIL -- promoting dialog between humanoid and computer. 1986 Rochester Forth Conference Proceedings, S. 229-232.

(Uri87) Urieli, Israel. REPTIL -- Bridging the gap between application and education. The Journal of Forth Application and Research, Vol. 4, Nr. 3, 1987, S. 405-434.

Der RTX 2000™

Ein neuer Prozessor für eingebettete Echtzeit-Steuerungs-Aufgaben kombiniert die Geschwindigkeit von RISC-Architekturen mit der Integrationsdichte von Microcontrollern

Melbourne, Florida, den 16. Mai 1988 -- Mit dem Harris RTX-2000™ [1], einem programmierbaren 16 Bit-Microcontroller für den Einsatz in zeitkritischen hochintegrierten Echtzeitsteuerungen wird das erste Standardprodukt vorgestellt, das mit Hilfe des FORTH-orientierten RISC-Prozessorkerns (RISC = Reduced Instruction Set Computer) von Harris Semiconductor entwickelt wurde. Der RTX 2000 stellt auch das erste Mitglied des Harris Real Time Express™ (früher F.O.R.C.E), einer Familie von Standard-ICs für Echtzeit-Anwendungen, dar. Der Einsatzbereich schließt Bildverarbeitungssysteme, Robotersteuerungen, digitale Signalverarbeitung und künstliche Intelligenz in Echtzeitsystemen ein.

Der RTX 2000 kann Befehlsraten von mehr als 10 MTPS (Million Instructions per Second) durchschnittlich erreichen, durch die direkte Ausführung der Programmiersprache FORTH entfällt die Notwendigkeit, Programme für Echtzeit-Anwendungen in Assemblersprache zu fassen. Der Baustein enthält auf dem Chip Speicherbereiche für Daten- und Rücksprungstapel (je 256 Worte), einen 16x16 Bit Hardware-Multiplizierer, der Multiplikationen in einem Taktzyklus ausführen kann, sowie einen Interrupt-Controller und drei frei programmierbare Timer.

Ein ASIC-Bus™ zum Anschluß von externer ASIC-Peripherie an den RTX 2000 bildet ein weiteres neuartiges Merkmal dieser Harris-Premiere. Der RTX-ASIC-Bus kann über sein Parallelinterface zur Leistungssteigerung eines Systems beitragen, indem er ASIC-Peripherie zur Hardware-Beschleunigung anbindet oder durch seine anwendungsspezifische Ein-/Ausgangsstruktur die Flexibilität des Designs erhöht. Das Vorhandensein dieses ASIC-Busses erlaubt es dem RTX 2000, externe ASIC-Hardware gleichzeitig während der Ausführung eines Befehles anzusprechen, indem ein direkter Pfad von diesem Bus durch die ALU auf den Parameterstack geschaltet wird.

Nach Angaben von Michael Graff, Vice President Marketing bei Harris Semiconductor, bietet der RTX 2000 die Kombination von hoher Geschwindigkeit und hoher Integrationsdichte, die zuvor nur bei Semicustomschaltungen denkbar war. Darüberhinaus wurde dieses Produkt - und die ganze Familie des Real Time Express - auf die speziellen Anforderungen der Entwicklung von Echtzeitsystemen zugeschnitten."

Was die Optimierung für Echtzeitbetrieb betrifft, so erreicht der RTX 2000 seine Leistungsfähigkeit durch einen einfach gehaltenen RISC-ähnlichen Aufbau. Ein hoher Grad an Parallelität wird durch eine besondere Architektur mit zwei Stapeln und vier Bussen (Quad Bus™) erreicht, die zum Erreichen ihrer Leistungsdaten ohne Pipelines auskommt. Durch Umgehen eines Pipelinebetriebs, wie er bei anderen RISC-Prozessoren Verwendung findet, vermeidet das Harris-Produkt erhöhte Komplexität und verbessert das Reaktionsverhalten auf zeitkritische Daten. In der Tat macht der RTX 2000 extrem schnelle Reaktionen auf Interrupts (weniger als 400ns bis zur Ausführung des ersten Befehls der Interrupt-Routine) ebenso möglich, wie eine schnelle Umschaltung zwischen Tasks (weniger als zwei Mikrosekunden).

Daneben wurde der RTX 2000 für den Betrieb mit Hochsprachen entworfen; er ist der erste Microcontroller, der (die Hochsprache) FORTH

1 RTX™, RTX 2000™, Real Time Express™ und ASIC-Bus™ sind eingetragene Warenzeichen von HARRIS Semiconductor.

direkt ausführen kann und so die Notwendigkeit der Programmierung in Assemblersprache für Echtzeitanwendungen völlig überflüssig macht. Die Pläne für den Ausbau des Harris Real Time Express sehen die Einführung von Crosscompilern für die Sprachen C, Prolog und ADA vor.

Die RTX-Familie wurde von Harris auch unter dem Gesichtspunkt anwendungsspezifischer Designs optimiert. Der RTX 2000 wurde unter der Verwendung der Harris Standardzellen-, Makrozellen- und Compiler-Bibliotheken realisiert. Dadurch wird eine direkte Umsetzung von Entwicklungen mit dem RTX 2000 in andere Konfigurationen von Standardprodukten ebenso möglich, wie kundenspezifische Semicustom-Bausteine. Letztere können unter Verwendung der RTX Toolbox™ und der Harris/SDA-CAD-Entwicklungsumgebung, welche die erprobten Standardzellen, Makrozellen und Compiler zur Verfügung stellen, entwickelt werden. Harris plant, plant diese Möglichkeiten Ende dieses Jahres auch für Kunden zu öffnen.

Von Harris wird unter der Bezeichnung RTX Development System™ ein auf dem IBM-PC™ lauffähiges interaktives Hardware-/Software-Entwicklungssystem für Echtzeitanwendungen angeboten. Dieses System unterstützt die Entwicklung und das Debuggen bei voller Prozessorgeschwindigkeit und stellt ein benutzerfreundliches integriertes Paket zur Verfügung, das Editor, Compiler, Disassembler und Debugger enthält.

Muster des RTX 2000 sind jetzt verfügbar. Produktionsstückzahlen werden im dritten Quartal 1988 lieferbar sein.

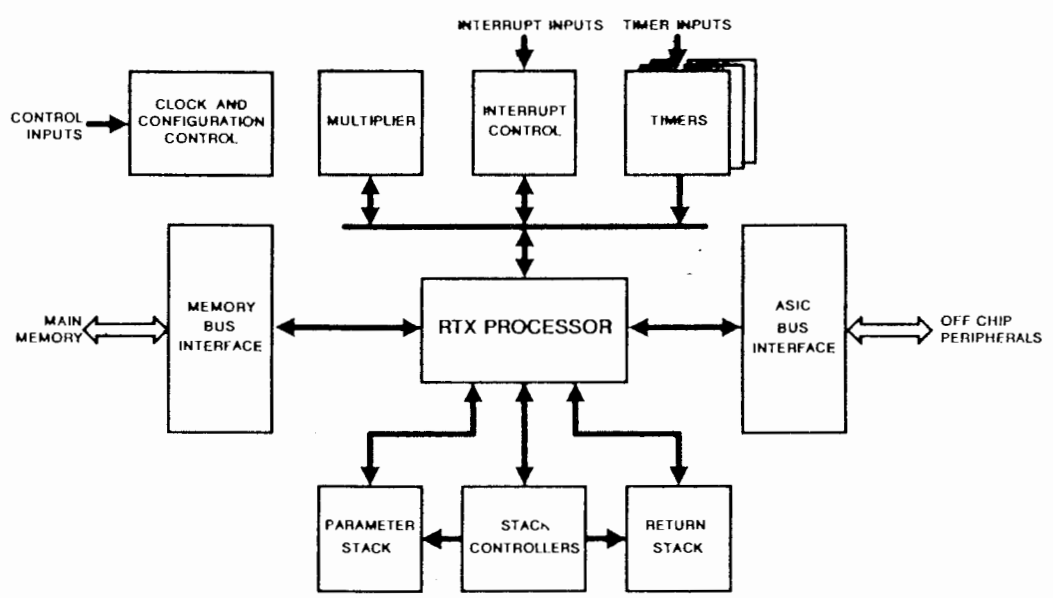
Anfangs wird der RTX 2000 in einem 84 Pin Pin-Grid-Array-Gehäuse geliefert, für die Zukunft sind Versionen im PLCC-Gehäuse und im Keramik-Quadpack vorgesehen. Ende des Jahres wird ein Produkt nach dem MIL-Standard 883C zur Verfügung stehen.

Quad Bus™, RTX Toolbox™ und RTX™ Development System™ sind eingetragene Warenzeichen von Harris Semiconductor. IBM PC™ ist ein eingetragenes Warenzeichen der IBM Corporation.)

Als nächsten Baustein der RTX-Familie plant Harris, eine vereinfachte Ausführung des RTX 2000 für weniger komplexe Anwendungen einzuführen. Dieser Baustein wird bei einem Preis von weniger als 100 US$ einen Befehlsdurchsatz von mehr als 10 MIPS liefern. Die Einführung ist von Harris für das nächste Quartal geplant.

Harris Semiconductor ist der achtgrößte US-Hersteller von Integrierten Schaltungen und ein Unternehmensbereich der in Melbourne, Florida beheimateten Harris Corporation, einem Hersteller von Produkten für Informationsverarbeitung, Kommunikation und Mikroelektronik mit einem Umsatz von 2,1 Milliarden US-Dollar.

RTX Block Diagram



Der RTX 2000

Vakuum-Pumpstandsteuerung für die Hochvakuumexperimentiereinrichtung (Hexe) am Beschleunigerlabor der LMU und TU München.

D. Maier , R. Ecker , F. Reither , G. Ziegler
TU München, Fak. für Physik, Elektronikabteilung.

1. Hexe (siehe Bild 1)

Die Hexe ist ein Vakuumkessel mit knapp 4 m Durchmesser, 3 m Höhe und einem Rauminhalt von 36 m^3. Sie besteht aus einem vertikal verschiebbaren Topf und einem feststehenden Deckel. Zwischen Deckel und Topf befinden sich 2 O-Ringe zur Abdichtung. Das Volumen zwischen den beiden O-Ringen muß evakuiert werden damit an der Topf-Deckel-Naht kein zu großes Druckgefälle entsteht (Zwischenvakuumpumpstand).

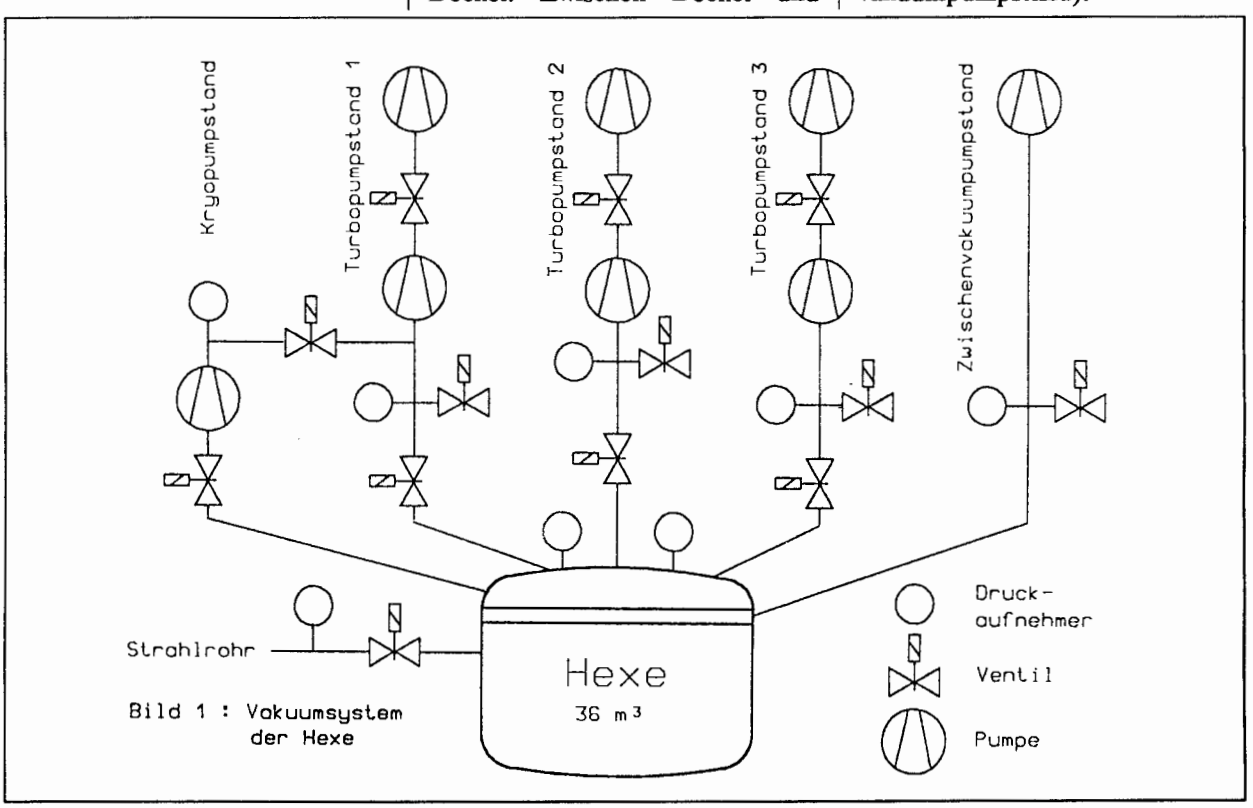


Bild 1 : Vakuumsystem der Hexe

Bild 1

Um in vertretbarer Zeit ein Vakuum von mindestens 10^{-6} hPa zu erreichen, wird die Hexe mit 3 Turbopumpständen, einem Kryopumpstand und dem Zwischenvakuumpumpstand betrieben.

Die Turbopumpstände bestehen jeweils aus einer Vorpumpe und einer Turbopumpe.

2. Forderungen

Es wurde gefordert eine automatische Steuerung zu entwickeln und zu bauen, die sämtliche Komponenten (Pumpen, Ventile usw.) des Vakuumsystems steuert und überwacht.

Die Bedienung muß sowohl vor Ort, als auch in der Meßbox (ca. 40m entfernt) möglich sein.

Es müssen folgende Betriebsmodi anwählbar sein.

Mode AUS:

- Die Ventile zwischen den Pumpständen und der Hexe sowie das Ventil zum Strahlrohr gehen zu.
- Alle Pumpstände werden ausgeschaltet und belüftet.
- Wichtige Vakuum-Komponenten werden überwacht.

Mode GRUNDSTELLUNG:

- Die Steuerung bringt das Vakuumsystem von jedem möglichen Istzustand in eine definierte Ruhestellung.
- Wichtige Vakuum-Komponenten werden überwacht.

Mode PUMPEN OHNE GASZÄHLER:

- In diesem Mode wird die Hexe vollautomatisch evakuiert.

Mode PUMPEN MIT GASZÄHLER:

- Die Hexe wird vollautomatisch evakuiert. Zusätzlich werden die Druckunterschiede zwischen Innen- und Außenseite der Gaszähler überwacht, um das Platzen der sehr dünnen Gaszählerfolien zu verhindern.

Mode MANUELL: (Service-Mode)

- Sämtliche Vakuum-Komponenten lassen sich manuell steuern.
- Aktivierte Vakuum-Komponenten werden überwacht.

3. Komponenten, die angesteuert/ überwacht werden

Ansteuerung:

Die Ansteuerung wird von den in Tabelle 1 aufgeführten Geräten vorgenommen.

Überwachung:

Die in Tabelle 2 aufgeführten Komponenten werden überwacht.

4. Realisierung

Die Steuerung kann über zwei VT100 Terminals bedient werden. Der Zustand des Vakuumsystems wird durch rot/grün leuchtende/blinkende LEDs auf einer Frontplatte mit Pumpstandschema dargestellt. Dadurch und unter Berücksichtigung der unter 3. aufgeführten Komponenten ergeben sich für die

Bedienerschnittstelle:

- 50 Bit out (LEDs auf der Frontplatte)
- 2 V.24 Schnittstellen

und für die

Hexe-Schnittstelle:

- 46 Bit out
- 75 Bit in
- 21 Analog in.

Anzahl	Gerät	Schaltzustand
7	Pumpen	(ein/aus)
1	Kompressor (Kryopumpe)	"
4	sonstige Geräte	"

Tabelle 1: Anzusteuerende Geräte

Anzahl	Gerät	Überwachungsgröße
3	Turbopumpen	(Drehzahl erreicht?)
1	Kompressor	(Fehler?)
1	LN_2-Control	(Förderung flüssiger) (Stickstoff ein? Fehler?)
10	Druckaufnehmer	(Fehler?)

Tabelle 2: Komponentenüberwachnung

4.1 Hardware (siehe Bild 2)

Die Steuerung ist als ECB-Bus-System aufgebaut. Auf allen Karten mit Z80-CPU ist ein Forth83-System mit Multitasker im EPROM implementiert.

Die Masterkarte steuert und überwacht das Vakuumsystem und ist gleichzeitig eine der beiden Bedienerschnittstellen.

Beide Analog-in Karten übergeben Druckwerte an den Master wobei der Slave zusätzlich als zweite Bedienerschnittstelle eingesetzt wird.

Die restlichen I/O-Karten treiben über Optokoppler galvanisch getrennt Ventile, Schütze, LEDs usw., bzw. übergeben die Zustände der Überwachungskontakte an den Master.

Die Steuerung befindet sich in einem 19" Gehäuse mit 9 Höheneinheiten. Zur Verdrahtung mit der Hexe wurde eine Verteilerbox gebaut. Ebenso wurde zum Testen der Hard- und Software eine Hexe-Simulationsbox (Hexchen) gebaut.

4.2 Software (siehe Bild 3)

Da die Hexe von wechselndem Personal betreut wird, legte man großen Wert auf Bedienerfreundlichkeit. So gibt es z.B. Help-Kommandos um die möglichen Befehle und den aktuellen Mode zu erfahren. Die Fehlermeldungen geben detailliert eventuell defekte Vakuumkomponenten und den erreichten Zustand an. Druckwerte können jederzeit an das Bedienerterminal ausgegeben werden. Pumpstände können während des Betriebs detached, die Pumpstandkomponenten getestet/repariert und der Pumpstand anschließend wieder attached werden, wobei das Programm selbstverständlich verhindert, daß die Hexe versehentlich belüftet wird.

Das Programm im Master besteht aus 12 Tasks, die über 5 Listen gesteuert werden. Dadurch wird eine bessere Programmstruktur erreicht und die Flexibilität erhöht. Die Pumpstandkomponenten werden über einen Teil dieser Listen an das Programm angepaßt. Für Programmänderungen / Erweiterungen bzw. den Einbau neuer Pumpstandkomponenten genügt die Änderung/Erweiterung der Listen bzw. weniger Tasks.

Bild 3 zeigt die grobe Struktur des Masterprogramms. MODE.CNTRL steuert über die Liste UNIVERSAL 5 Pumpstandtasks welche wiederum über die Listen LED, FORCE, TEST und TIME die Tasks LED.CNTRL, PUMP.CNTRL und VALVE.CNTRL steuern. Letztgenannte Tasks treiben direkt die Vakuumkomponenten und die LEDs auf der Frontplatte.

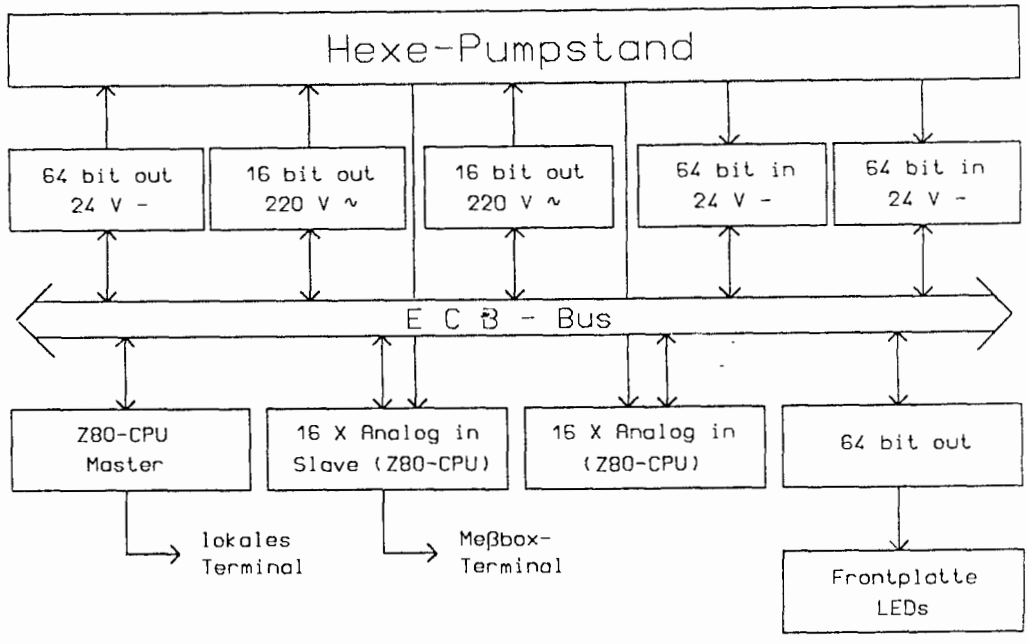


Bild 2 : Blockschaltbild der Steuerung

Bild 2

Die Task ERR.MESS wird von MODE.CNTRL bei jedem Durchlauf einmal aktiviert. Sie testet Fehlerbits in Universal, gibt gegebenenfalls Fehlermeldungen am Bedienerterminal aus und geht anschließend in den Ruhezustand.

Die Task RX.FR.ADC, die vollkommen selbstständig läuft, fordert von den beiden Analog-in-Karten Druckwerte an und legt sie in einem Speicherfeld ab. Außerdem übernimmt sie die Verbindung zum Meßboxterminal über den Slave.

Jeder Komponente des Vakuumsystems ist ein Zeiger zugeordnet, über welchen der Zugriff auf die Listen erfolgt.

Alle Listen sind in einem batteriegepufferten CMOS RAM untergebracht, so daß nach einem Stromausfall das Programm definiert weitermachen kann.

Die Programme in den ADC-Karten übergeben auf Anforderung Druckwerte an den Master, wobei das Programm im Slave zusätzlich die Verbindung des Meßboxterminals zum Master übernimmt.

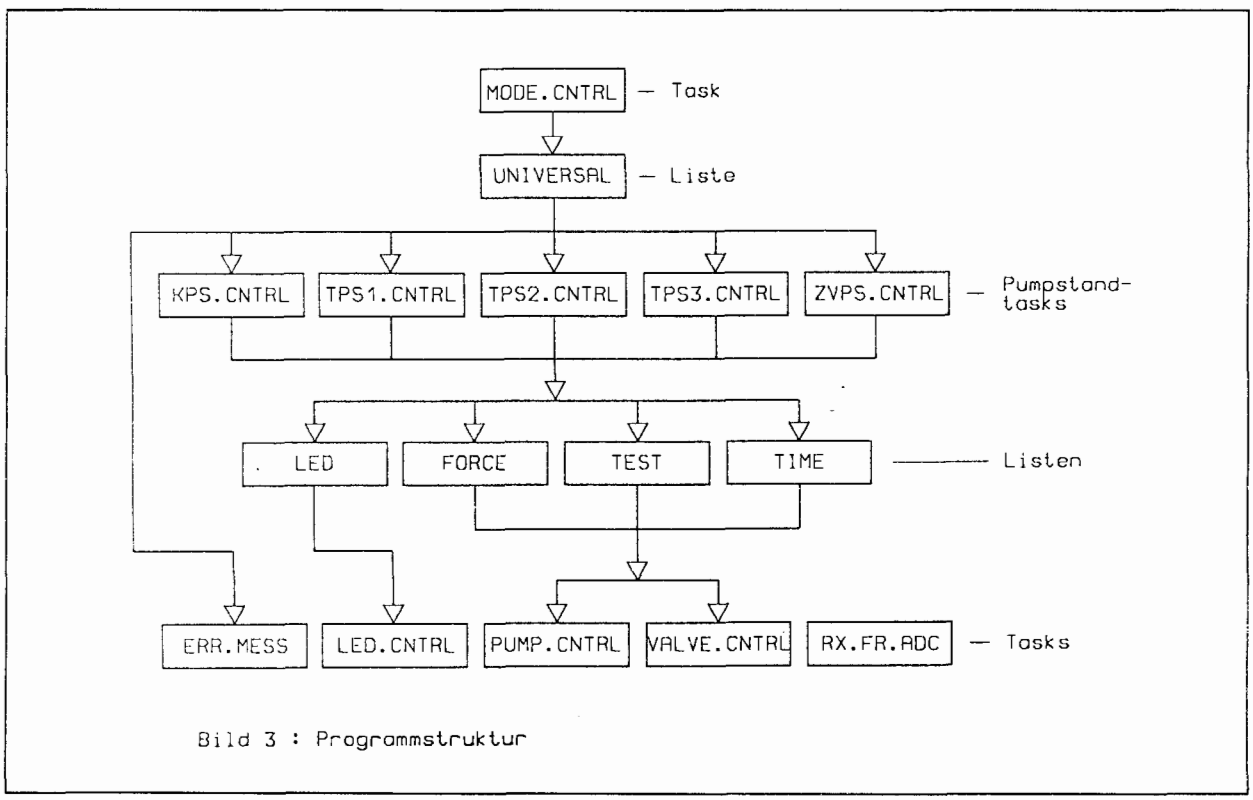


Bild 3

Hardware-Realisierung von FORTH

Gert-Ulrich Vack
Akademie der Wissenschaft der DDR, Zentralinstitut
für Kybernetik und Informationsprozesse

aus: Mikroprozessortechnik, Berlin/Ost 1 (1987) 6

1. Grundlagen

Bei der Anwendung von FORTH wirkte sich in einigen Fällen bisher gegenüber reiner Assemblerprogrammierung niedrigere Laufzeiteffektifität aus; in der Literatur werden für den Geschwindigkeitsverlust Faktoren von 1,2 bis 10 (im Vergleich zu reinen Assemblerprogrammen) und 2 bis 4 (bezüglich der Programmierung in C oder FORTRAN) ausgewiesen. (Dennoch gehört FORTH zu den schnellen Programmiersprachen!) Bereits in den siebziger Jahren wurde deshalb mit "reinen" Hardware-Lösungen von FORTH experimentiert. Unabhängig von FORTH wird seit längerem eine Diskussion zu RISC-(Reduced Instruction Set Computer) Architekturen geführt. Bei diesem Konzept wird die Architektur bewußt einfach und orthogonal gehalten, womit VLSI-Prozessoren großer Verarbeitungsbreite effektiv realisierbar sind. Der "Verlust" an Leistung gegenüber CISC (Complex Instruction Set Computer) Architekturen (dazu gehören die heute bekannten 16- und 32-Bit-Universalprozessoren) wird durch mächtigere Compiler und Betriebssysteme ausgeglichen.

Mit jüngsten Hardware-Entwicklungen wird der Beweis angetreten, daß FORTH-Prozessoren im Sinne von RISC-Architekturen implementierbar sind (unteres VLSI-Niveau) und eine hohe Systemleistung ermöglichen. Daneben werden in Universalprozessoren (z.B. Z8800/1) Maßnahmen zur Hardware-Unterstützung von FORTH-ähnlichen Sprachen vorgesehen (einfache Realisierungsmöglichkeiten für den Adreßinterpreter als zeitkritischen Teil eines FORTH-Systems). Die Relevanz eines FORTH-Prozessors liegt in der flexiblen Einsetzbarkeit für unterschiedlichste Anwendungsklassen, bevorzugt in den Bereichen der Kleinautomatisierung, der Meßtechnik, sowie in flexibel an neue Einsatzbedingungen anpaßbaren gerätetechnischen Lösungen (etwa intelligente Sensoren, Prozeßsteuerung, Antriebstechnik und damit z.B. das weite Feld der Robotertechnik, der Werkzeugmaschinen oder der flexiblen Fertigungssysteme).

Im folgenden soll deshalb ein Überblick zu den international bekannt gewordenen Lösungen gegeben werden.

2. FORTH-Karte mit 6502

Einer der ersten Versuche in Richtung FORTH-Prozessor war die Essex Forth Microcard aus dem Essex Electronics Centre of the University of Essex. Auf einer 80 mm mal 100 mm großen Leiterplatte wird mit 7 IS auf Basis des Rockwell-Prozessors R 6502 (hier zum FORTH-Prozessor R65F12 modifiziert) ein FORTH-System im ROM angeboten, welches bei 1 MHz Taktfrequenz betrieben wird. Der Speicherraum beträgt 16 KByte. 40 E-/A-Leitungen werden betrieben. Über RS 232 ist ein Terminal anschließbar. Der RSC-FORTH-Kernel belegt 3 KByte. 10 Interruptquellen und 2 programmierbare 16-Bit-Zähler/Timer können für Echtzeitaufgaben genutzt werden. Mit Einsatz des 65-FRI-Development-ROM wird die Karte zum Entwicklungssystem erweitert. Ein Anwendungsbeispiel ist ein automatischer Tester für die Strom-/Spannungskalibrierung (6 Analogeingänge, 3 zusätzliche Schaltkreise einschließlich 8-Kanal-AD-Umsetzer). Das Programm (Umfang eine A4-Seite Quelltext, 15 FORTH-Worte) realisiert eine Toleranzfeldprüfung jedes Einganges und zeigt optisch die erforderlichen Abgleicheingriffe an.

3. MF16LP-Single Board Language Processor

Die britische Firma Metaforth Computer Systems Limited entwickelte im Doppeleuropaformat eine Leiterkarte mit Bipolar-Kundenwunschschaltkreisen und HMOS-RAM zur Hardware-Realisierung von FORTH. Der Rechner hat eine Verarbeitungsbreite von 16 Bit sowie 24 Adreßleitungen. Für Stacks und Register werden 2 KByte High-Speed-RAM verwendet.

Der Prozessor wird mikroprogrammiert und realisiert etwa 60 FORTH-Worte. Bei Verwen-

dung von Speichern mit 60 ns Zugriffszeit werden die meisten FORTH-Operationen in 100 bis 550 ns abgearbeitet; das entspricht etwa 2 bis 10 Millionen FORTH-Befehlen je Sekunde. Ein solches FORTH-System ist damit etwa 50- bis 100mal schneller als eine Software-Emulation von FORTH auf dem Z 80. Die Nestungsoperationen werden im Zeitschatten der Befehlsabarbeitungen realisiert, so daß kein Interpretationsoverhead für die Unterprogramm- bzw. Fadencodetechnik entsteht. Auf dem System sind Multitasking-Versionen von FORTH 79 und FORTH 83 sowie ein C-Compiler einsetzbar. Außerdem ist auch PASCAL verfügbar. Module, die in verschiedenen Sprachen programmiert wurden, können gemixt und gemeinsam abgearbeitet werden.

Zur Systementwicklung sowie -realisierung werden eine CPU-Karte, ein Entwicklungsmodul für IBM-PC sowie ein VME-Bus-Entwicklungssystem angeboten. Damit sind die Anwendungslösungen (Echtzeitsysteme für Bildverarbeitung, Grafik, CAD/CAM oder Expertensysteme usw.) zu implementieren. Für Multiprozessorkonfigurationen bestehen Kopplungsmöglichkeiten über 6 Hochgeschwindigkeits-E-/A-Kanäle mit Übertragungsraten von 10 MBit/s.

In /2/ werden Ergebnisse von Laufzeituntersuchungen angegeben. Das Sieb des Eratosthenes (ein Standard-Benchmark-Programm, bei dem die Primzahlen zwischen 4 und 16381 zehnmal gezählt werden) benötigt danach für den MF16LP in FORTH 1,09 s, auf dem 4-MHz-Z80 76 s, auf dem 5-MHz-8086 64 s und auf einer PDP 11/70 11,8 s.

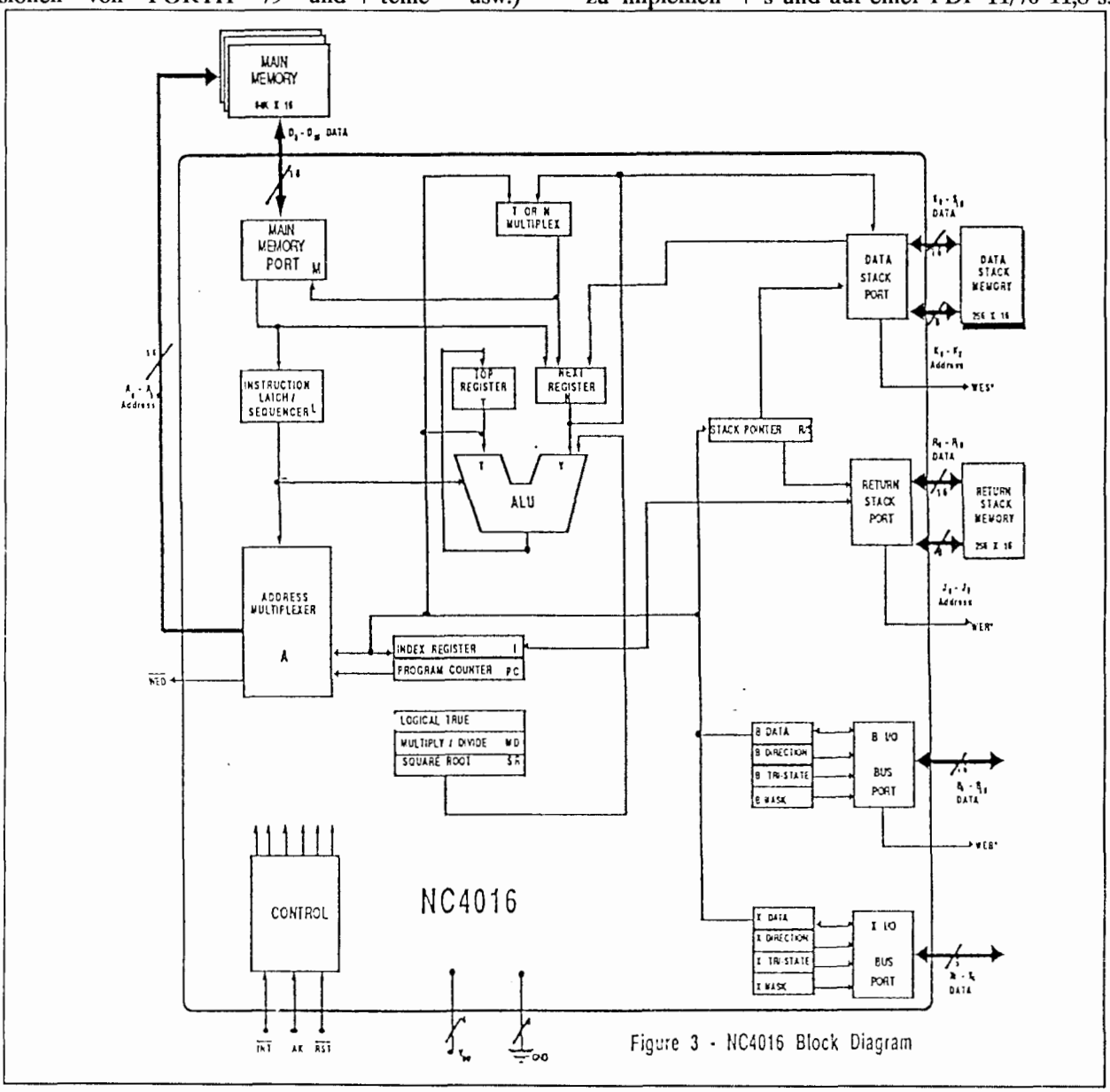


Figure 3 - NC4016 Block Diagram

Bild 1: Interne Struktur des NC 4000

Eine C Realisierung dieses Algorithmus auf der PDP 11/70 benötigt 1,52 s.

Mit aller Vorsicht läßt sich abschätzen, daß ein FORTH-Prozessor mit einer solchen Verarbeitungsleistung eine Reihe von (nicht zu anspruchsvollen) Signalverarbeitungsaufgaben lösen kann.

Eine Besonderheit des MF16LP ist die Metaforth Soft Architecture. Dabei wird der Anwender in die Lage versetzt, nach der Austestung eines Programmsystems mit dem Metaforth Profiler zeitkritische Stellen im Programmlauf zu identifizieren. Spezialisten der Herstellerfirma können anschließend Ergänzungen im Mikroprogramm des Rechners vornehmen, um eine höhere Abarbeitungsgeschwindigkeit der fraglichen Programmteile zu erzielen. Mit diesem Konzept wird eine Brücke geschlagen zwischen der Starrheit von universellen, komplexen Mikroprozessorarchitekturen und den wesentlich höheren Kosten für den Anwender erkauften Flexibilität von Bit-Slice-Realisierungen eines Spezialprozessors. Die Softwarekonzeption gestattet es, einen Spezialprozessor in der Firmware des FORTH-Prozessors zu realisieren.

4. FORTH-Prozessor NC 4000

Der markanteste Schritt in Richtung FORTH-Prozessor war die Entwicklung des NC 4000 durch Cupertino/Kalifornien gemeinsam mit dem Begründer von FORTH, Charles Moore. Inzwischen sind bereits verschiedene Realisierungsvarianten dieses Prozessors bekannt geworden.

Der NC 4000 (Bild 1) ist ein 16-Bit-Prozessor und arbeitet vollstatisch mit maximal 8 MHz Taktfrequenz. Der Chip ist in einem 120poligen Gehäuse untergebracht. Erste Muster wurden in HCMOS-Gate-Array-Technik gefertigt; intern ist der Schaltkreis überwiegend in Bit-Slice-Technik

Dipl.-Ing. Gert-Ulrich Vack studierte von 1972 bis 1976 an der Technischen Universität Dresden Informationstechnik. Nach der Erlangung des Diploms begann er seine Tätigkeit als wissenschaftlicher Mitarbeiter im ZKI der AdW der DDR. Er beschäftigte sich mit dem Aufbau, der Programmierung und der Anwendung von Mikrorechnern für industrielle Steuerungsaufgaben. Außerdem befaßte er sich mit Architekturen von modernen Mikroprozessoren und mit dem objektorientierten Softwareentwurf. Seit 1981 arbeitet er intensiv an der Implementierung, Nutzung und Verbreitung von FORTH-Systemen.

ausgeführt. Der Prozessor hat keine Mikroprogrammsteuerung und ist deshalb in der Firmware nicht erweiterbar. Daraus ergeben sich jedoch Geschwindigkeitsvorteile. Die meisten der 40 implementierten FORTH-Primitive werden in einem Takt ausgeführt (einschließlich Anweisungen für strukturierte Programmierung). Zusätzlich können auch 130 Kombinationen aus Primitiven in einem Wort codiert und somit in einem Takt ausgeführt werden. Daraus resultiert eine enorme Verarbeitungsleistung, die über der des M 68000 liegen soll /2/.

In den Befehlscodes wird das oberste Bit ausgewertet. Ist es auf Eins gesetzt, so bezeichnen die nachfolgenden Bits die Adresse eines anderen Wortes innerhalb des Fadencodes (Secondary, Nestungsoperation). Ist das Bit 0, so handelt es sich um den Code einer sofort ausführbaren Primitive. Ein weiteres Bit kennzeichnet in den Befehlscodierungen eine Rückkehranweisung in die aufrufende Ebene. Dieses "Return" wird parallel zur Verarbeitungsoperation ausgeführt, so daß (ebenso wie beim MF16LP) der Interpretationsoverhead entfällt.

Die Reduzierung des Adreßraumes durch die vordefinierten Bits in den Befehlscodes ist bedeutungslos, da der generierte Code außerordentlich kompakt ist. Der gesamte adressierbare Speicher umfaßt 64 KWorte. Mit Nutzung der Erweiterungsleitungen kann er bis auf 2 MByte expandiert werden. Daten- und Programmstacks werden als getrennte Speicherblöcke realisiert und über getrennte Busse erreicht. Das ermöglicht eine extrem kurze Umschaltzeit bei Context-Switch sowie eine Parallelarbeit bei Hauptspeicher- und Stackzugriffen. Die oberste der 257 Programmstackzellen sowie die obersten 2 der 258 Datenstackzellen liegen im Schaltkreis als Register vor. 16 E-/A-Kanäle können bidirektional, maskierbar, selbstvergleichend und für gelatchte bzw. Tristate-Ausgaben genutzt werden. Gegebenenfalls können auch die 5 Erweiterungsleitungen für die Expandierung des Speicheradreßraumes als E-/A-Leitungen betrieben werden. Eine dieser Leitungen ist darüber hinaus auch als Interrupteingang nutzbar.

Der Prozessor greift auf die Stacks, auf den Hauptspeicher und auf die E-/A-Adreßräume parallel während der Verarbeitungsoperationen zu.

Auf Basis des NC 4000 wird ein Einplatinenrechner unter der Bezeichnung SBC Beta-Board als Zusatzkarte für den IBM-PC angeboten. 28 KWorte RAM und 4 KWorte ROM sowie 8 Stacksegmente (Umschaltung in weniger als 5 s) stehen zur Verfügung, um ein Programmentwicklungssystem auf der Basis von PolyFORTH zu unterstützen. Das Delta Evaluation System SC 10 000 Delta Board ist ein Entwicklungsmodul (4MHz, 8 Stacksegmente, je 4KWorte RAM und EPROM, RS 232-Schnittstelle, 21 E-/A-Kanäle). Ergänzungsmoduln (Bus, Peripherie, Speicher) sind angekündigt.

5. Schlußfolgerungen

Aus Untersuchungen der Universität von Hull /3/ geht hervor, daß prozentuale Programmiersprachen auf der Grundlage von FORTH-Architek-

turen implementierbar sind. Der FORTH-Prozessor könnte dabei als Zielhardware für eine Vielzahl von Programmiersystemen sowie Fachsystemen genutzt werden.

Mit den Möglichkeiten der VLSI-Technologie sind FORTH-Prozessoren als Chip realisierbar. Auf Grund der Flexibilität der Software (Implementierungsprinzip, Erweiterbarkeit des Sprachkonzeptes) wird auch die Hardware universeller einsetzbar, womit große Stückzahlen bei der Herstellung des Prozessors ökonomisch realisierbar sind. Die Leistungsmerkmale erreichen oder überschreiten die von 16-Bit-Prozessoren. Selbst für komplizierte Anwendungsfälle (Bildverarbeitung, Künstliche Intelligenz) ergeben sich mit dem FORTH-Prozessor interessante Realisierungsalternativen gegenüber Universalprozessoren, wobei einfachere, zuverlässigere, leistungsfähigere und insgesamt ökonomischere Lösungen zu erwarten sind.

Bei der Konzipierung eines FORTH-Prozessors sind immer folgende Aspekte zu berücksichtigen:

- Realisierung als fest verdrahtete oder mikroprogrammierte Hardware
- Auswahl des Basisbefehlssatzes
- Eignung für den Einsatz in Multiprozessor- und Mehrrechnerkonfigurationen (auch in heterogenen Systemen, gemeinsam mit Universal-, Spezial- und Koprozessoren)
- Auswahl einer geeigneten Programmierumgebung (FORTH 83, Poly-FORTH usw.).

Literatur

/1/ Meininger, W.:"Super 8": Microcontroller mit 8-Bit-Architektur und Hochsprachunterstützung DESIGN & ELEKTRONIK 5/86, S. 94-99

/2/ FORTH-Systeme für die Industrie. Katalog der Firma Angelika Flesch, Titisee-Neustadt, 1986. S.27-43

/3/ Kundeninformation zur Interkama 1986

/4/ Strass, H.; Brodie, L.: Der FORTH-Mikroprozessor : Programmiersprache und CPU aufeinander optimal abgestimmt. DESIGN & ELEKTRONIK 5/86, S. 94-99

/5/ NOVIX NC 4000 Microprocessor. Firmenmaterial von Novix, Cupertino

/6/ forth microcard incl. application note. Firmenmaterial von Essex Electronics Centre, Univ. of Essex, Colchester, Essex

KONTAKT

Akademie der Wissenschaften DDR, Zentralinstitut für Kybernetik und Informationsprozesse, Kurstr. 33, PF 1298, Berlin, 1086; Tel. Dresden 00375/4633255

Modulares Mikrokontroller Konzept mit LCD Treiber von EUROSIL electronic GmbH

EUROSIL electronic GmbH, Erfurter Straße 16, 8057 Eching
Ansprechpartner: Herr Peter Stelzner, Applikations-Ingenieur,
Telefon.(089)31906-150

Die EUROSIL electronic GmbH hat ihren Firmensitz in Eching/München und ist bekannt als Hersteller von integrierten Schaltkreisen in 1,5 V CMOS Technologie. Die weltweiten Aktivitäten der Firma haben hier ihren Ursprung - alles aus einer Hand. Forschung, Entwicklung, Design, Test, Produktion, Qualitätssicherung in hervorragender geographischer Lage im Herzen Europas machen EUROSIL electronic zum EUROPÄISCHEN CMOS Partner.

Für Applikationen, welche ein LCD Display und intelligente Logik bei geringstem Stromverbrauch erfordern, bietet das 4 Bit Mikrocomputer-Konzept von EUROSIL electronic eine wirtschaftliche Lösung. Bei dem in CMOS Technologie realisierten Controller handelt es sich um ein modulares Konzept mit bemerkenswerten Eigenschaften.

Modular ARCitecture 4 Bit Microcomputer (kurz MARC4)

Die Familie der MARC4 beinhaltet unter anderem auch eine Version mit LCD Treibern on-chip.

Die in der Harvard-Architektur aufgebaute CPU kann Programm und Datenspeicher separat adressieren und dadurch den Befehlshol- und Datentransferzyklus überlappen (pipelining). Die stack-orientierte Architektur des MARC4 und der zugehörige kompakte Befehlssatz einer "Null-Adress-Maschine" erlauben eine effiziente Programmierung in der Hochsprache FORTH.

Die bekannten Vorzüge einer höheren Programmiersprache und die vorhandene Software-Modulbibliothek führen zu sehr kurzen Programm-Entwicklungszeiten.

Der Speicherbereich der CPU umfaßt ein bis zu 4096 Byte großes ROM für Programm und Tabellen und ein bis zu 256 x 4 Bit großes RAM. Dieses interne RAM enthält neben Variablen und Arrays auch den Daten-Stapel (Expression Stack) und den Rücksprungadress-Stapel (Return Stack).

Der Informationsfluß erfolgt im wesentlichen über zwei Busse.

Der interne Speicher-Bus vermittelt alle Adress- und Datentransfers zwischen ROM, RAM und ALU. Der externe I/O Bus stellt die Verbindung zu den peripheren Blöcken unterschiedlichster Art her (z.B. zum A/D Wandler oder zum LCD Treiber). Maximal sind bis zu 32 Peripherieadressen ansteuerbar.

Das derzeit realisierte LCD-Modul beinhaltet die Logik für 16 Segment-Leitungen und 4 Leitungen für die Backplane Multiplex-Steuerung. Die Multiplexrate (statisch, 2:1, 3:1 oder 4:1) sowie die Zuordnung der einzelnen Segmente zu den Segment- und Backplaneleitungen sind softwareprogrammierbar. Somit lassen sich heute maximal 64 Segmente LCD-Displays ansteuern, eine Erweiterung auf 128 Segmente ist für künftige Versionen geplant.

Das LCD-Modul enthält außerdem einen Spannungsverdoppler und -verdreifacher, um auch bei Versorgungsspannungen von minimal nur 1,5 Volt noch ein LCD-Display treiben zu können.

Der MARC4 beinhaltet je einen Quarz- und einen RC-Oszillator.

Beim Quarz Oszillator handelt es sich um einen preisgünstigen 32 kHz Uhrenquarz. Mit Hilfe dieser exakt programmierbaren Zeitbasis lassen sich leicht Funktionen wie z.B. die einer Echtzeituhr realisieren.

Der RC Oszillator mit einer Nominal-Frequenz von 1MHz steuert den internen CPU-Takt und ermöglicht so eine Befehlszykluszeit von 2 sec.

Dieser interne Taktgeber wurde als RC Oszillator ausgeführt, um bei einem Interrupt den Prozessor sehr schnell vom Stand-by Betrieb (Sleep-Mode) in den aktiven Modus zu bringen. Im Sleep-Mode bleiben die Ausgangszustände und die LCD-Anzeige erhalten und der 32 kHz Oszillator aktiv. Der Stromverbrauch wird jedoch auf 1 A reduziert.

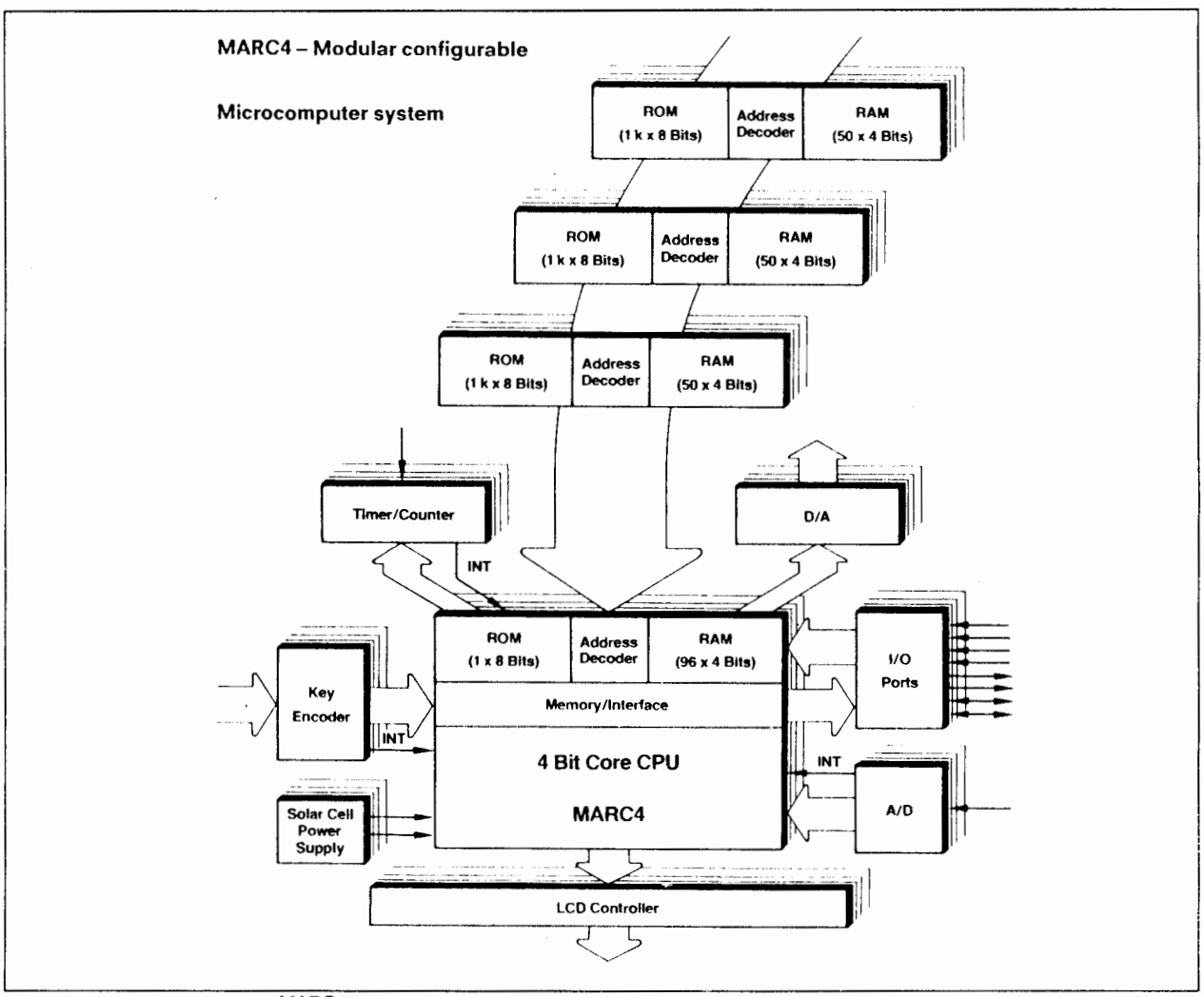
Das MARC4 Software Entwicklungssystem mit einem integrierten Editor, optimierendem qFORTH Compiler und einem Software Simulator läßt sich über anwenderfreundliche "Pull down" Menüs sehr einfach bedienen. Es ist auf jedem IBM-PC kompatiblen Rechner (mit 640 KByte RAM und 2 Floppy-Disk Laufwerken) ablauffähig.

Der Hardware Emulator ist als Einsteckkarte für einen IBM-PC konzipiert und wird etwa im Oktober zur Verfügung stehen.

Zur Unterstützung des System Entwicklers sind folgende Dokumentationen vorgesehen:

- MARC4 Evaluation Package
- qFORTH Programmer's Guide
- Development System Reference Guide
- qFORTH Dictionary

Neben dem jetzt vorhandenden LCD Modul sind in nächster Zukunft ein Counter/Timer, ein A/D-Wandler Modul und ein EEPROM geplant. Die MARC4 Produktlinie stellt ein nach oben offenes Konzept dar, auf dem sich sogar kundenspezifische Module in Standard Cell Technologie mitintegrieren lassen, um einen kundenspezifischen Mikrocomputer zu konfigurieren.



MARC4

BÜCHER

J.Staben

I. 83 Standard

BEILSTEIN, H.-W.: Wie man in FORTH programmiert

In meinen Augen neben dem Buch "Starting FORTH" von Brodie eine der bestgemachten Einführungen in FORTH 83.

Beilstein führt ganz behutsam in FORTH ein und baut sein Buch gut auf.

Er geht konsequent vom Einfachen zum Schwierigeren und vergißt nicht, daß die meisten von uns im Mathematikunterricht immer Comics gelesen haben. Wenn er mathematische Beispiele bringt, führt er auch die Entwicklung der Lösung vor. Die binäre Logik wird ebenfalls einsichtig abgehandelt und nach 130 Seiten müßte jeder aufmerksame Leser in der Lage sein, alle Steueranweisungen richtig einzusetzen.

Das Kapitel über den Editor muß notgedrungen systemabhängig sein, aber da ja sowieso zu jedem System ein Handbuch gehört ...

Anschließend geht's langsam ans Eingemachte: Die FORTH-Ein-/Ausgabe, die FORTH-Pufferbereiche und dann tauchen schon Stackpointer und andere fremde Wesen auf.

Danach führt Beistein vor, wie man FORTH einsetzt: Gerade das Beispiel einer kleinen Dateiverwaltung ist eine sinnvolle Anwendung.

Ab S. 245 werden dann Interna des FORTH-Compilers durchdiskutiert, die einen FORTH-Anfänger aber erst viel später interessieren. Daher werden die eigentlichen Höhepunkte von FORTH nur knapp und wenig detailliert beschrieben - nur sind Compilererweiterungen, objektorientierte Programmierung und FORTH/Assembler-Mix kaum Themen für die Zielsetzung dieses Buches.

Man siehts daran: Die CREATE ... DOES>-Konstruktion wird erst auf S. 276 eingeführt.

Trotz einer Reihe schmerzhafter Fehler werde ich das Buch als Bezug für den Einstieg ins volksFORTH benutzen und die Möglichkeit anbieten, das volksFORTH über ein zuladbares Paket an dieses Buch anzupassen.

Allerdings hat diese Anpassung dort ihre Grenzen, wo Beilstein Möglichkeiten nur anreißt, die im volksFORTH konsequent verwirklicht wurden. So sind Vorwärtsreferenzen, lokale Variablen und headerless Code nach dem aktuellen Stand in der FORTH-Welt im volksFORTH implementiert.

BRODIE, Leo: Denken in FORTH

Dies ist ein wirklich erfrischendes Buch und eine Fundgrube - nicht nur über FORTH, sondern auch über das Programmieren allgemein. Brodie scheut sich auch nicht zu sagen, daß ihm Programmieren und das Ausprobieren am Gerät Spaß macht: Die sogenannte Fun-down Methode. Bei Brodie wird deutlich: Er beherrscht FORTH und hat auch die Fähigkeit, davon etwas weiterzuvermitteln.

Dr. Dobb's Toolbook of FORTH, Volume II

Wer sich für den aktuellen Stand der Programmiertechniken in FORTH interessiert, kauft sich für etwas über 100 DM (mit Diskette) das

Dr. Dobb's Toolbook of FORTH,
Volume II
M&T Publishing, Inc.

zu beziehen über Markt&Technik, München; den Buchhändler fragen. Erschienen ist es Ende Januar 1988.

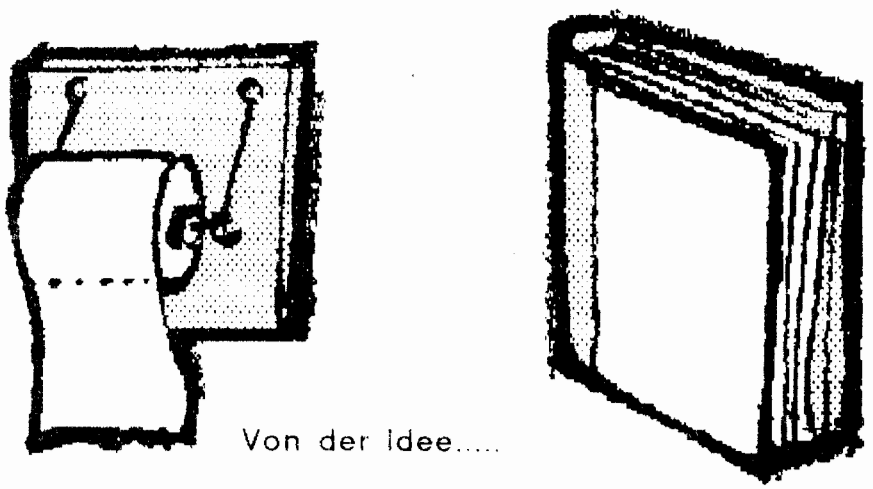
Auf 500 Seiten breiten FORTH-Cracks in fünf Kapiteln ihre Techniken und Gedankenansätze zur Diskussion aus.

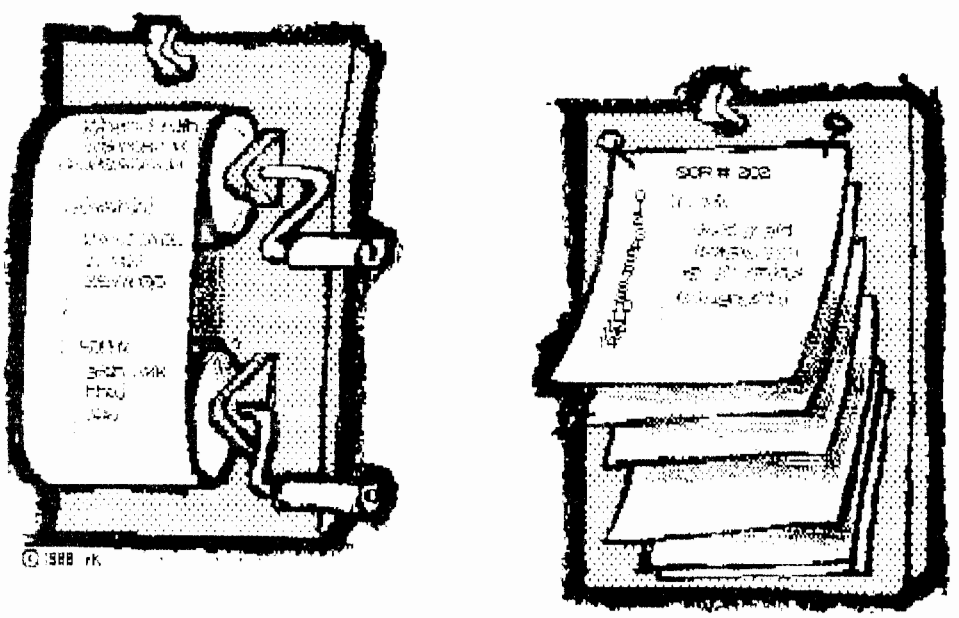
Das hier schon die Schaffung neuer Datentypen über CREATE..DOES> schon auf S. 32 diskutiert wird, mag einen ersten Eindruck geben. Soll man

ein Array zur Laufzeit oder zur Compilezeit auf die Indexgrenzen hin überprüfen? oder gar nicht?

Wie weit soll ein FORTH-Programm modularisiert werden? Wenn schon der Quelltext in kleinen Modulen geschrieben wird, dann muß das FORTH-System auch die Verwaltung und Einbindung über Directories Subdirectories bereitstellen. VolksFORTH macht das übrigens.

Details des FORTH-Compilerbaus, Cross-Compiler, Meta-Compiler sind weitere Themen, an die sich dann die Umsetzung der FastFourierTransformation anschließt.





I stream, you screen ?

Window-handling für F83 auf dem PC, fraktale Grafiken auf dem PC mit CGA oder EGA-Grafik und eine vollständige Tabellenkalkulation runden das Kapitel über Anwendungen ab. Oder was dachtet Ihr, worin die neue Datenbank "RapidFile" von AshtonTate geschrieben ist??

Das letzte Kapitel "Herausforderungen" zeigt auch den Leuten, die nicht den in FORTH geschriebenen LISP-Interpreter von U.Hoffmann benutzen, wozu FORTH fähig ist:

Ein Echtzeit-Expertensystem, zum LISP-Interpreter noch ein LOGO-Interpreter, Aufbau von Wissensbasen, Übernahme von SmallTalk-Eigenschaften in FORTH und der Versuch der Analyse natürlicher Sprachen.

Was mich besonders beeindruckt hat, war ein Artikel über den Einsatz von FORTH in der Behindertenrehabilitation.

Daß das volksFORTH83 auf dem Stand der Technik ist, sieht man daran, daß K. Schleisiek seine Implementierung von benannten lokalen Variablen in FORTH hier in diesem Buch vorstellt. Diese Implementierung ist selbstverständlich schon jetzt im volksFORTH83 zuladbar!!

C.H. Ting: Inside F83

Dies ist die in Englisch geschriebene Grundlage für die Anwendung des Laxen&Perry F83-Systems.

Offete Enterprises, Inc. 1984
bei
FORTH Systeme A. Flesch
Pf.1226, 7820 Titisee-Neustadt

Zech, Ronald: FORTH 83

Dieses Buch kann wohl als das deutsche Handbuch zum Laxen&Perry F83 angesehen werden. Darüberhinaus geht Zech sehr detailliert auf den neuen 83er Standard ein, führt seine Beispiele aber immer am F83 durch.

Dieses Buch hat weniger erklärenden als beschreibenden Charakter. Vorteilhaft bei Zech ist, daß er FORTH83 am konkreten Beispiel des Laxen&Perry F83 FORTH Modells beschreibt. Dadurch kann man alle Beispiele schön nachvollziehen und fällt mit dem sehr komplexen F83 nicht so oft auf die Nase.

Die "Einführung" geht über knappe 10 Seiten und dann winkt schon der erste Bruch unter einem langen Wurzelzeichen. Bereits auf S. 33 hat Zech die Zahlensysteme und die Boolsche Logik hinter sich.

Das nächste Kapitel heißt "Programmieren in FORTH" und bietet schon auf S. 64 Inhalte, die uns Beilstein - wohlweislich - 250 Seiten lang verschwiegen hat. Dafür weiß man spätestens nach S. 66, ob man überhaupt noch in FORTH programmieren möchte.

Danach kommt es Schlag auf Schlag:

- CASE-Konstruktionen als Verknüpfung von Tabellen.
- Vektor-Execution und Vorwärts-Referenzen.
- Die Definition von Compiler-Anweisungen.

Einer der Höhepunkte von FORTH ist die Schaffung beliebiger Datentypen über die CREATE ... DOES>-Anweisung. deren Einsatz beschreibt Zech ab S. 136 sehr detailliert, wobei er auch die Programmierung zeitkritischer Teile in Assembler vorführt.

Über den Aufbau und die Auswertung von Tabellen (Beispiel: Sinus-Tabelle) und die Diskussion der Massenspeicher-Einbindung geht Zech dann weiter zur Echtzeitprogrammierung. Hier zeigt er an einem netten Beispiel den Einsatz des Multitaskers.

Der Rest des Buches beschreibt nur noch Werkzeuge und weiterführende Techniken. Rekursion, FORTH/Assembler-Mix, Segment-Adressierung, ein komplettes Grafikpaket für die CGA und ein Basic-Interpreter ist auch noch irgendwo abgedruckt.

Ich möchte noch erwähnen, daß es vom Franzis Verlag auch ein lila Zech-Buch gibt, daß vorrangig den fig-FORTH-Standard beschreibt. Aber für den PC-Besitzer ist doch eher der 83er Standard gültig. Zusammenfassend kann man sagen: Jemand, der das Buch von Zech durcharbeiten kann, hat auch mit dem Einsatz von volksFORTH keine Schwierigkeiten.

Die nächste 'Vierte Dimension' erscheint im November/ Dezember 1988

II. fig-Standard

Glasmacher, Peter: FORTH-Handbuch

Der Titel ist leider etwas zweideutig: Ohne jeden Zweifel ist dieses Buch ein Handbuch für FORTH, aber anders als man sich dies gemeinhin vorstellt. Weder Einführung noch Bedienungsanleitung - so wendet es sich an den erfahrenen Anwender, der eine Funktion benötigt und kurz nachschaut, wie sie formuliert wird. Dieses Werk von P. Glasmacher entspricht eher dem Sinn der bekannten TurboPascal-Toolboxen. Dies sind ebenfalls Sammlungen von Prozeduren, die der Anwender

einsetzen kann, auch ohne den letzten mathematischen Hintergrund zu verstehen.

Glasmacher, Kiesenberg: FORTH Handbuch

Dieses Handbuch ist eine sehr schöne Einführung in FORTH. Ich selbst habe mit diesem Buch, das auch oftmals in öffentlichen Leihbüchereien zur Verfügung steht, meine ersten Gehversuche mit FORTH gemacht.

E. Floegel: FORTH Handbuch

Dieses FORTH-Handbuch stellt ebenfalls eine Einführung in FORTH dar. An einer Reihe von sinnvollen Beispielen wird hauptsächlich der Umgang mit den Rechenfunktionen gezeigt, so z.B. die oftmals notwendige Skalierung von ganzzahligen werten.

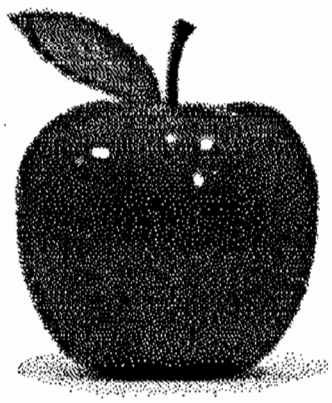
Die hier verwendeten Stack-Diagramme sind nicht so einsichtig, aber das ständige Bemühen, FORTH dem Einsteiger an Beispielen näher zu bringen, macht dieses Buch auch zu einem vernünftigen FORTH-Handbuch.

Allerdings benutzt Floegel den polyFORTH-Dialekt, der wohl wenig verbreitet ist, aber zum 79er und zum fig-Standard kompatibel ist.

Leider sind recht viele Tippfehler drin, aber dafür funktionieren die Algorithmen.

Goppold, Andreas: FORTH: Ein Programmiersystem ohne Grenzen

Den zwiespältigen Eindruck, den dieses Buch macht, kann man schwer begründen. Auf der einen Seite bemüht sich Goppold, für FORTH zu werben und die Unterschiede zu "gewöhnlichen" Programmiersprachen aufzuzeigen, zum anderen bleibt sein Buch ohne roten Faden, ohne klares Ziel. Zwischendurch wird ein wenig Sozialkritik geübt, etwas philosophiert und dazwischen Grundlagenwissen in FORTH vermittelt. Vielleicht mal aus der Bücherei ausleihen!



Der Apfel der Erkenntnis

Haydon, Glen: All about FORTH

Dieses Buch beschreibt vollständig die Implementierung von G. Haydon für Mountain View Press, wie sie durch die PC-SIG-Disk 031 repräsentiert wird.

MVP-FORTH Kernel Version 1.0305.02

Glen Haydon hat auch einen "FORTH guide" geschrieben, eine Einführung für Anfänger.

Hogan, Thom: FORTH ganz einfach

T. Hogan hat hier ein klares und verständliches Buch geschrieben, das gut übersetzt wurde und den Leser behutsam von Thema zu Thema führt. Am Ende jedes Kapitels bietet es eine Zusammenfassung und gibt dem Leser das Gefühl: "Nun weiß ich mehr über FORTH". Grundlegende Operationen, Kontrollstrukturen, Speichermanipulationen und Eingabe/Ausgabe werden anschaulich vermittelt; nur beim Erläutern der Stackvorgänge über eine halbe Seite hat er übertrieben, das wird bei R. Zech besser dargestellt. Allerdings behandelt Hogan den fig-Standard, der immer noch durch die Z80-Maschinen eine gewisse Gültigkeit hat.

Knecht, Ken: Einführung in FORTH

Wenn jemand dieses Buch erwirbt, um in FORTH eingeführt zu werden, so erlebt er eine bittere Enttäuschung - es sei denn, er betreibt die MMS-FORTH Version 1.9 auf dem TRS-80 Rechner mit dem Betriebssystem TRS-DOS/NEWDOS.

Dann allerdings besitzt er eine einfache Einführung in die Handhabung des Systems, die auch ex-

akte Hinweise über die Nutzung der 16KB-RAM und die Umgehung der Ladezeit von 4 Min. gibt. Allerdings gilt der Tip, vom Level II BASIC aus das FORTH neu zu starten, wohl nur für die oben genannte Zusammenstellung. Auch die Systembefehle (z.B. RBLK oder WBLK) werden nur auf dem TRS-80 funktionieren.

Hilfreich für den Computereinsteiger ist der ständige Vergleich von Routinen in FORTH und in BASIC - allerdings wird der Wechsel von BASIC nach FORTH selten sein. Einige Befehle wie PERFORM - PEND kannte ich vorher gar nicht und einige wie PRINT kollidieren sogar mit dem Standard, so daß dieses Buch nur der TRS-80/MMS-FORTH Besitzer nutzbringend verwenden kann.

Ken Knecht versucht Stackoperationen wie in BASIC einzuführen. Aber durch das Verwenden von absoluten Adressen ist die Portabilität der FORTH-Programme natürlich dahin.

Reymann, J: FORTH, GOLDMANN TB 13128

Diese Taschenbuchreihe wird aufgelöst und so ist dieses Buch über FORTH nur noch auf dem Wühltisch zu finden. Zu Recht, wie ich meine; denn es findet nur eine kurze, stichwortartige Beschreibung von FORTH statt. Weder stehen die Gedanken in einem Zusammenhang, noch hat sich der Übersetzer die Mühe gemacht, die Beispielprogramme ins Deutsche zu übertragen. Darüberhinaus sind die Beispiele für interessierte Anfänger zu schwer und zu unverständlich. Nicht zu empfehlen!

Roberts, S.D.: FORTH Applications

Dieses, in schönstem Englisch geschriebene Buch hat vom Ziel her Ähnlichkeit mit dem FORTH-Handbuch von P. Glasmacher. Während bei Glasmacher ganz bestimmte Funktionen wie das Umwandeln von Zeichenketten direkt als Quelltext vorliegen, hat Roberts hier eine Reihe verschiedener Themen aus unterschiedlichen Gebieten in FORTH zu lösen versucht.

Wie Roberts es noch 1985 niederschrieb, ist FORTH nicht lesbar (siehe Bild 1):

```
: N>P 0 SPAD C! 1 CNT I 32 SPAD
    C! BEGIN KEY DUP 13 = NOT
              WHILE ?C IF DUP EMIT SPAD
              CNT @ + C! 1 CNT +!
              ELSE DROP THEN REPEAT
    CNT @ SPAD !C 32 CNT @ SPAD
+ C! DROP ;
```

Bild 1

III. Die Problemfälle

Bücher, die nie erschienen oder unauffindbar sind:

Monadjemi, P.: FORTH für Fortgeschrittene

ist identisch mit

Monadjemi. P.: FORTH Tips & Tricks

Beide Bücher haben dieselbe ISBN - sie sollten zwar bei Data Becker erscheinen, sind aber nie auf den Markt gekommen.

Winfield, A. : Alles über FORTH

Da scheint es wohl in den Staaten ein Buch zu geben:

Allan Winfield: The Complete FORTH

Winzer, TH. :FORTH

Bei den letzten beiden bin ich mir nicht sicher; sollte jemand sie gelesen haben, gebt doch bitte Bescheid!

Das Büro der FORTH-Gesellschaft e.V. ist umgezogen

Aus verwaltungstechnischen Gründen ist die FORTH-Gesellschaft e.V. umgezogen.
Unter nachstehender Anschrift erreichen Sie uns in allen Angelegenheiten:

FORTH-Gesellschaft e.V.

Postfach 1110

D-8044 Unterschleißheim

Verwenden Sie bitte nur noch diese Anschrift.

IV. Atari ST

Redaktionelle Ergänzung

R. Aumiller/D. Luda: Atari ST 32FORTH Compiler

Diesem Handbuch liegt ein 32FORTH Interpreter auf Diskette bei. Man kann damit alle GEM-, VDI- und Betriebssystemroutinen nutzen (einschließlich LINE-A), sowie GEM-Applikationen schnell und problemlos programmieren. Die Schnelligkeit des Programmes erlaubt eine rasche Kompilation von Quelltexten. Einem professionellen Arbeiten mit 32FORTH am Atari ST steht nichts mehr im Wege: Alle 32FORTH Befehle werden genau beschrieben. Die Beispiele für die GEM-Programmierung ermöglichen bald die Entwicklung eigener GEM-Programme.

R. Aumiller/D. Luda : Programmieren mit FORTH Atari ST

Dieses Buch wendet sich an diejenigen, die schon über etwas Programmiererfahrungen in anderen Sprachen verfügen und nun komplexere Probleme bearbeiten wollen. Die Einführung in FORTH erfolgt schrittweise über einfache Programme bis hin zur fortgeschrittenen Programmierung. Dem Profi zeigt eine ausführliche Darstellung der Programmierung von FORTH unter GEM, wie vielfältig die Fähigkeiten des Atari ST genutzt werden können. Alle GEM-Routinen werden ausführlich beschrieben, und es wird gezeigt, wie man sie einsetzen kann. Dem Buch liegt eine Diskette mit über 50 Beispielprogrammen bei.

Mitgliederservice

Diese beiden Bücher können zu einem ermäßigten Preis über

D. Luda Software
Staudingerstr. 65
8000 München 83
Tel. 089/670 83 55

bezogen werden.

Inserentenverzeichnis:

Mitgliedschaftsantrag

Antrag auf Mitgliedschaft in der FORTH-Gesellschaft e. V.

Name: .....

Straße: .....

Ort: .....

Telefon: .....

FORTH-Gesellschaft e.V

Postfach 1110
D-8044 Unterschleißheim

Postgiro Hamburg
BLZ 200 100 20
Konto-Nr.: 56 32 11 - 208

Jährlicher Beitrag:

- ☐ DM 32,- (Schüler, Studenten, Rentner und Arbeitslose, nur mit Nachweis !)
- ☐ DM 64,- (Ordentliche Mitglieder, Auslandsadresse)
- ☐ DM 128,- (Fördernde Mitglieder, Firmen und Institutionen)

Außerdem unterstütze ich die FORTH-Gesellschaft e.V. mit einer Spende von DM: .....

Den Gesamtbetrag von DM .....

- ☐ möchte ich von meinem Konto abbuchen lassen.
- ☐ habe ich als Verrechnungsscheck beigelegt.
- ☐ habe ich am ..... auf ihr Konto überwiesen.

Datum: ..... Unterschrift: .....

(Bei Minderjährigen Unterschrift des Erziehungsberechtigten)

Einzugsermächtigung FORTH-Gesellschaft e.V., Postfach 1110, D-8044 Unterschleißheim

Bitte buchen sie den Mitgliedsbeitrag von meinem Konto ab.

Straße: .....

Name: .....

Ort: .....

Konto-Nr.: .....

BLZ: .....

Bank: .....

Unterschrift: .....

Auch im Original leere Seite

ADRESSEN

Lokale FORTH-Gruppen, die sich regelmäßig treffen:

2000 Hamburg 72 Klaus Schleisiek, Roter Hahn 42c (Laden), Bus 277 von U-Bahn Berne oder Barmbek, Tel. 040/6449412. Lokaler Koordinator: Peter Vollmann, Tel. 040/6440221.

6100 Darmstadt Andreas Soeder, Tel. 06257/2744. Treffen an der VHS an einem Mittwoch in der Mitte des Monats.

8000 München Heinz Schnitter, Tel. 089/3103385 und Christoph Krinninger 089/7259382. Treffen jeden Mittwoch im Monat 19 Uhr 30 im Vereinsraum 1 im Bürgerhaus Unterschleißheim am Rathausplatz (S-Bahnhaltepunkt S1 Unterschleißheim).

FORTH-Fachgruppen:

8000 München RTX 2000 Gruppe, Koordinator Martin Diez, Treff- und Zeitpunkt wie oben bei der lokalen münchner Gruppe.

6800 Mannheim FIS (FORTH Integriertes System) - Datenbank, Textverarbeitung, Kalkulation, Postadresse: Dr. med. Elemer Teshmar, Danziger Baumgang 97, 6800 Mannheim 31

Es möchten in ihrer Region eine Gruppe gründen:

7000 Stuttgart 31 Wolf-Helge Neumann, Huttenstr. 27, Tel. 0711/882638.

8500 Nürnberg 20 Thomas G. Bauer, Fichtestr. 31, Tel. 0911/538321.

Eine Fachgruppe will gründen:

7000 Stuttgart 80 Grafik/Arithmetik, Jörg Tomes, Anweilerweg 56, Tel. 0711/7802293.

8000 München 70 Btx u. FORTH, Christian Schwarz, Lindenschmitstr.30, 8000 München 70

Ansprechpartner zu bestimmten Interessengebieten:

Volksforth/Ultraforth:	Bernd Pennemann, Tel. 030/7923923 und Klaus Schleisiek, Tel. 040/6449412.
32-Bit Systeme:	Robert Jones, Tel. 02434/4579
Forthchips,- maschinen und RISC:	Roland Steck, Tel. 06151/661192
Künstliche Intelligenz:	Ulrich Hoffmann, Tel. 0431/677808
NC4000 Novix Chip:	Klaus Schleisiek, Tel. 040/6449412
Realtime relationale Netze:	Wigand Gawenda, Tel. 040/446941
Gleitkomma-Arithmetik:	Andreas Döring, Tel. 02631/52786

FORTH-Gesellschaft e.V.

FORTH GESELLSCHAFT e.V. - Postfach 1110 - D-8044 Unterschleißheim

Postgiroamt Hamburg, Kontonr.: 563211-208 BLZ 20010020