VIERTE DIMENSION

Volume VI/Nr.3 - September 1990

Themen

- Echtzeit'90
- LINKED ACTIONS
- FORTH Interactive Tutor
- RISC-FORTH
- BlocksWorld

FORTH MAGAZIN

7,50 DM

FORTH - Workshop
in Suhl (Thüringen)
3.- 4. Dezember 1990

EDITORIAL

Nun ist dieser heiße Sommer auch schon wieder fast vorbei. Es werden jetzt kühlere Tage und längere Nächte folgen, so daß sicherlich mancher seinen Computer öfters benutzen wird. Vielleicht entstehen dabei wieder so anregende Beiträge, wie sie diese 'Vierte Dimension' enthält.

Interessante und aufschlußreiche Nachrichten aus der FORTH-Gemeinde bilden den Anfang des Heftes. So können Sie beispielsweise erfahren, welchen Stellenwert FORTH auf der *Echtzeit'90* einnahm. Oder dem Artikel 'Brief aus der Provinz' entnehmen, welche Anstrengungen in Raum Moers unternommen werden, um FORTH attraktiv zu machen.

Äußerst aufregend ist die Möglichkeit, die FG-Mitgliedern bald offensteht, über ihr Modem mit aller Welt zu kommunizieren. Heinz Schnitter und Johannes Teich waren aktiv und haben der FORTH-Gesellschaft e.V. eine weltweite Adresse im EUnet verschafft, die auch allen Mitgliedern zur Verfügung steht.

Die bekannten Autoren J.Plewe, F.Stüss und J.Staben haben sich Gedanken über die Organisation der Worte in FORTH gemacht. Dabei ist ein längerer Artikel entstanden, dessen ersten Teil Sie in dieser 'Vierten Dimension' finden.

Artikel über eine computergestütze Lernhilfe für FORTH, ein RISC-FORTH und ein UNIX™-ähnliches Filesystem bilden den Abschluß des Heftes.

Rainer Aumiller
Denise Luda

Echtzeit'90 in Sindelfingen

IMPRESSUM

Titel:
FORTH MAGAZIN 'Vierte Dimension' ®
Zeitschrift der Mitglieder der FORTH-Gesellschaft e.V.
© 1990

Herausgeber:
FORTH-Gesellschaft e.V.

Redaktion und Satz©:
✉ D. LUDA Software©, Gustav-Heinemann-Ring 42, 8000 München 83, ☎ 089/670 83 55, FAX 089/679 22 71

Kontaktadresse:
Entweder direkt die Redaktion anrufen bzw. anschreiben, das FORTH-Büro in München, Postfach 1110, 8044 Unterschleißheim, ☎ 089/3173784 kontaktieren oder die FORTH-Mailbox (s.u.) 'Konferenz Vierte Dimension' benutzen.

Quelltextservice:
Der Quelltext von Beiträgen, die mit diesem Symbol gekennzeichnet sind, ist auf der Leserservice-Diskette zur jeweiligen Ausgabe oder in der FORTH-Mailbox ✆ 08841/5880 8N1 zu finden.

Quelltext Service

Autoren dieser Ausgabe:
Jörg Staben, Jörg Plewe, Frank Stüss, M. Schultheis, Andreas Findewirth, H.-G. Willers, Alexander Burger, Johannes Teich, Ulrike Schnitter, Heinz Schnitter, Friederich Prinz, Arndt Klingelnberg.

Erscheinungsweise:
Vierteljährlich

Redaktionsschluß:
Die zweite Woche im mittleren Quartalsmonat

Auflage:
ca. 1000 Stück

Druck:
Buch- und Offsetdruckerei Bickel Söhne, Frankfurter Ring 243, 8000 München 40

Bezugspreis:
Einzelheft DM 7,50, Abonnement 4 Hefte DM 40,-, bei Auslandsadresse DM 45,- inklusive Versand.

Vierte Dimension

I n h a l t

Leserbriefe und Zuschriften

FORTH-Autoren Forum in Btx

Interessierte FORTH-Autoren können ab sofort Ihre Kurzbeiträge (ca. 1-10K MSDOS-Dateien *.TXT, *.COM und *.BLK, in Ausnahmefällen auch länger) in Btx plazieren.

Die Autoren-Beiträge auf IBM-kompatiblen 5,25/3,5"-Disketten werden von mir als transparente Telesoftware im Postformat eingespielt. Ein Download ist via postzugelassenem Public Domain-Dekoder XBTX möglich, der von mir kostenlos abgegeben wird (gegen Einsendung einer formatierten Diskette in rücksendefähiger Verpackung mit selbstadressiertem Aufkleber und Rückporto). Auf dieser Diskette kann bereits der erste Beitrag eingereicht werden. Dieselbe Diskette wird an den Absender zurückgeschickt, sobald interessante Neuigkeiten für Ihn anliegen, oder falls der Beitrag nicht aufgenommen werden konnte, was hoffentlich nie vorkommt.

Ersteinsender erhalten neben XBTX.COM auch eine Datei mit dem Namen TELAUT.TXT (Teleautoren Betriebsanleitung) mit einer detaillierten Autoreninformation.

Die Beiträge in Btx werden mit einer Abrufgebühr belegt, die einerseits die Anbieterkosten abdecken und andererseits die Autorenleistungen abrufprozentual vergüten soll. Die Themen sind selbstwählbar.

FORTH-vereinsspezifische Fragen können ebensogut diskutiert werden wie FORTH-Software Beiträge.

Zum Ausbau und Optimierung des praktizierten Autoren-Modells gibt es als Dauerthemen:

- ✗ Freie kooperative Telearbeit im Autoren/Anbieter-Verbund
- ✗ Automated Freelancing
- ✗ Selbstbedienungs-Arbeitsplatz für Autoren

✆ Man erreicht mich per Btx durch Eingabe von: `*dr. pc#`
✆ Per Fax mit09875/471
✉ Per Post: Dr. Fridemar Pache, Feldgartenweg 14, 8802 Wolframs-Eschenbach

Kleinanzeige

Novix-Board DB1, wie neu und nicht benutzt + Software + Manual + Buch "Footstep" von Thing, komplett nur DM 720,- VB, ☎ 06131/368274 ab 10.00 Ludwig Richter

FORTH-Workshop

Vom 3.-4. Dezember (der ursprüngliche Termin im Herbst wurde wegen Zeitgleichheit mit der *electronica90* geändert) diesen Jahres wollen wir einen FORTH-Workshop in der DDR durchführen. Anbieter von FORTH-Systemen und Applikationen sollen die Möglichkeit haben, ihre Produkte vorzustellen und mit potentiellen Anwendern und Interessenten ins Gespräch kommen (Vortragende können sich *gebührenfrei* anmelden!).

Um uns einen Überblick über den Umfang der angebotenen Leistungen und Produkte zu verschaffen, teilen Sie uns Ihr Angebot bitte umgehend unter untenstehender Anschrift mit. Dort können auch weitere Informationen angefordert werden. Der Anmeldeschluß für Vortragende und Aussteller ist der 12.Oktober '90, der für sonstige Teilnehmer ist der 2. November '90.

✉ Kammer der Technik Suhl, Interessengemeinschaft FORTH, "Workshop", Postschließfach 190, DDR-6300 Ilmenau

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, Gustav-Heimann-Ring 42, 8000 München 83, Tel. 089/6708355, FAX 089/6792271 oder legen Sie sie in der FORTH-Mailbox München 'Konferenz Vierte Dimension' ab (8N1 Tel. 089/7259625).

Am liebsten hätten wir die Manuskripte auf einer Diskette 5 1/4" (360 Kbyte oder 1,2 Mbyte) im IBM-Format oder einer 3 1/2" Diskette (Atari-Format oder 720 Kbyte IBM-Format). Ist Ihnen das nicht möglich, können Sie auch normale Texte auf Papier einsenden. Bei Bildern sollte allerdings darauf geachtet werden, daß ein möglich guter Kontrast vorliegt. 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.

ECHTZEIT '90

von Arndt Klingelnberg

Am 19. bis 21. Juni fand in Sindelfingen die Ausstellung und der Kongress *Echtzeit'90* statt. REALTIME in Automation, Meßtechnik und Simulation mit dem Motto: 'Eine späte Antwort ist eine falsche Antwort'. Dieser besondere Bereich der modernen Datenverarbeitung fordert schnelle und noch wichtiger garantiert kalkulierbare Reaktions- und Antwortzeiten von Rechnern und Komplettsystemen.

Die FORTH-Gesellschaft e.V. und der PEARL e.V. veranstalteten den Kongress. Neben diesen zwei Programmiersprachen trat noch ADA deutlich hervor, dazu natürlich C, Pascal und Modula sowie auch besondere Realtime-Kernel. Mit dem Kongress wurde die Echtzeit-Isolation zwischen einzelnen Entwicklern und — notwendig — auch zwischen den Programmiersprachen-'Sekten' durchbrochen.

Die Siegersensation des Echtzeit-Programmier-Wettbewerbs: Als einziges Team hat die FORTecH-Gruppe aus der DDR die Aufgabe gelöst! Im Vordergrund der angejahrte CPM-Rechner Robotron.

Das Ferzenackl

Ein Programmier-Wettbewerb sollte als wichtiges FORTH-Ereignis in die Geschichte eingehen, fiel jedoch aufgrund der Turbulenzen des Umfeldes kaum auf: Die *DDR* und *FORTH* erzielten mit deutlichem Vorsprung den 1. Preis, obwohl das große Publikumsinteresse und auch die Tücken ihres individuellen Exemplars des 'Ferzenackl' dem Team eher hinderlich waren.

Wer bei der FORTH-Tagung '89 in Aachen schon die 5-Pixel Schwingschrift eines USA REALTIME-Wettbewerbs erlebt hatte, war gespannt. 9 Teams traten an, das 'Ferzenackl' zu bändigen. Eine Stahlkugel wurde von einem Schrittmotor in einem am Stahlband aufgehängten Körbchen transportiert und dann auf eine Wippe abgelegt. Eine Lichtschranke meldete die Wippen-Position an den Rechner zurück, da diese wiederum das Körbchen behinderte. Ein Elektromagnet mußte eingeschaltet werden, um das Körbchen zum Wiedereinfangen der Kugel zu fixieren.

10 Durchläufe ohne Handberührung waren gefordert. 2 Stunden und 5 Minuten erforderte die Bändigung dieses Prozesses. Das Publikum stand dicht gedrängt um einen riesigen, schweren Rechnerkasten und verfolgte die Kompilier-Iterationen: hier 2 Steps mehr; da einer weniger; doch etwas schneller in der 'Phase 5' weiter drehen; nein, auf der abfallenden Flanke der Lichtschranke muß getriggert werden. Zwischendurch klickte eine der Floppies, dann endlich: 13 Durchläufe unter dem Anfeuerungsgeraune des daumendrükkenden Publikums.

Damit gingen 3000 DM als 1. Preis an das FORTecH-TEAM Pfüller - Woitzek - Neuthe aus Rostock *auf* Z80 2.5 MHz COMFORTH. Die Reise mit Z80-Robotron, 12V-Netzteilblock und Benzinkanister auf der Trabbi-Rückbank hatte sich gelohnt.

Bereits deutlich abgeschlagen nach Zeit und Anzahl der erfolgten Kugel-Durchläufe ging ein drahtloses AEG-Telefon an das Trio um Jens Martin von der Uni Kaiserslautern *auf* Amiga 500 mit C. Platz 3 erreichte ein potentielles FORTH-Team, das dann jedoch Pascal einsetzte. Ein RTX-Evaluation-Board

ging an Dieter Peter (Brosius+Köhler) und Duchrow (Duchrow-Software) *auf* einem AT.

Ein 386iger mit 68020 Zusatzprozessor in einem 19"-Gehäuse unter Unix V mit RTK (RealTimeKernel) in C und Assembler, wie auch MS-C 5.0 und Quick-C 2.0 jeweils *auf* einemAT286 waren noch mit von der Partie. Ein zweites FORTH-Team hatte Hardwareprobleme, der Lötkolben konnte nicht mehr helfen.

Silicon-FORTH aus deutschen Landen

Nun ein Querschnitt durch Ausstellung und Kongress – Echtzeit-FORTH-spezifisch.

Der deutsche **FORTH-µP**: ein FORTH-RISC-Prozessor in CMOS-Technologie war der FRP 1600 von E-T-A aus Altdorf (bei Nürnberg), der als Funktionsmuster auf dem Stand der Firma Delta-t zu sehen war. Es ist ein RTX verwandter Prozessor, bei dem auf besonders schnelle Task-Wechsel und verbessertes Interrupt- und Flag-Handling Wert gelegt wurde. Die Multitask-Stacks liegen nun extern und können schnell umgeschaltet werden. Der Vorteil solcher Kontroller liegt in der Umgehung des von-Neumannschen Flaschenhalses, da getrennte Ports für Daten-Stack-Daten und -Adressen, Daten und Adressen sowie getrennte, unabhängige Ein-/Ausgabe zur Verfügung stehen. FORTH anstelle von bzw. *als* Assembler verspricht Komfort und Programmierschnelligkeit und damit automatisch bessere Software-Ergebnisse. Was die Ausführungsschnelligkeit angeht, so zeigen Benchmarks, daß ein solcher Prozessor bei 10 MHz vergleichbar ist zu einem 68020 bei 30 MHz. Der Vorteil des FORTH-µP's liegen in dem einfachen Aufbau des Chips: 33.000 Transitor-Funktionen gegenüber 190.000 beim 68020. Das erlaubt als Zukunftsaspekt den Einsatz verschiedenster Techniken wie: schnellste GalliumArsenid-Implementierungen, ASIC-Integration und Radiation-save Design.

Kongress-*PICK*

Nun ein paar *Pick*'s aus dem Kongress-*Stack*.

A. Krämer (Harris) betonte die Dauer und Vorhersagbarkeit von Aktionen (Interrupts, Taskwechsel): ein Cache ist ungeeignet, Pipelines bedingt und ein Stack gut geeignet (der RTX200 wird als Stack-orientierter Prozessor verkauft).

Ulrich Hoffmann diskutierte über Multitasking. Dieses kann nach dem Zeitscheiben-Verfahren (timesharing) ablaufen oder aber Prioritäts-gesteuert sein: Gerechtigkeit gegen Wichtigkeit. Ein *Sceduler* kontrolliert die Zustände der Tasks: 'nichtexistent', 'wartend', 'ablauffähig' und 'laufend'. Der *Sceduler* springt in einer doppelt-gelinkten sortierten Weckzeitliste von Kopf zu Kopf der Task-Prozesse. Die Latenzzeit während eines Taskwechsel ist besonders klein zu halten, da hier Interrupts nicht zugelassen werden können. Für ein System mit RTX2000 ergibt sich eine Interrupt-Latenzzeit von < 3 µs und eine Context-(Task)-Switchzeit von <40 µs + 2 µs je Stackelement.

Die sinnvolle Erweiterung des Rechensystems in Richtung auf den konkreten Prozess hin (Woitzel) und die Erweiterung der Sinne des Menschen durch Meßgeräte mit physiologisch angepaßten, also ausreichend schnellen Auswertungen (Reilhofer) sind FORTH-gemäße Problemstellungen. Online-Auswertungen verbunden mit dem Aha!-Effekt des Versuchsingenieurs stellen einen wesentlichen Schritt auf der Informationsfindung und -verdichtung (!) zu den Entscheidungsträgern in Mangagement und Politik hin. (Gerade in der Umweltproblematik ist Echtzeit-Rechenpower gefordert, nicht zuletzt für Simulationen und eine verbesserte Prozeßsteuerung, *der Verfasser*).

Auf Publikumseinwürfe hin erörterte Woitzel sehr einleuchtend was FORTH ist; eben nicht nur eine Programmiersprache (und auch keine Religion, *der Verfasser*): Der mit einer FORTH-Maschine zulässige Dialog hat in FORTH zu erfolgen. Nicht die Sprachen im engeren Sinne sind Objekte eines sinnvollen Vergleiches sondern die gesamte Programmier- und Ablaufumgebung mit allen Hilfsmitteln. Überhaupt gelang es ihm, das 'Phänomen' FORTH den vielen *nicht*-FORTH'lern in sinnvollen Termini der Informatik prägnant zu beschreiben. Sein Vortrag befaßte sich mit abgesetzter bzw. verteilter Compilierung. Rechner tauschen nicht mehr Quelltexte aus, sondern 'execution-tokens' (was wohl ein allgemeiner Begriff für die Code-Field-Adresse war).

Klaus Schleisiek-Kern erörterte das MUCK-System. Neben Ausführungen zu RTX2000, FORTH und Multitasking (siehe U. Hoffmann) waren hier die Angaben zum Rapid-Prototyping wichtig. Die Möglichkeit z.B. einen Laptop als Terminal und Massenspeicher zu verwenden und ohne ROM-Probleme die Anwendung auszutauschen oder zu überarbeiten, da diese im EEROM liegt, ermöglicht universelle Einsetzbarkeit und schnellste Software-Iterarionen. Aufgaben werden gelöst, bevor sie zu Marketing-Problemen werden. Nur zu oft ist die Software unzureichend, da sie umständlich in Generierung und Überprüfung und eben auch meist fest in ROMs eingebettet ist.

Heinz Schnitter stellte das OpenNetworkForth vor, ein bewußt offenes, transparentes und verteiltes FORTH-System, mit bis zu 255 Knotenrechnern, basierend auf dem Token-Passing Netzwerk ARCNET. Im Vortrag wurde die Notwendigkeit eines deterministischen Zugriffsverfahrens auf das Netzwerk für Echtzeitanwendungen diskutiert.

Objekte werden im ONF durch Datenstrukturen dargestellt und die Adressierung durch ein IMPORT/EXPORT-Modell gelöst. Mit EXPORT macht man ein FORTH-Wort dem Netzwerk bekannt, die hierdurch angelegte Vorwärtsreferenz kann durch IMPORT auf einem anderen Rechner ausgewertet werden. Fehler werden zum ERROR-Logger, einem spezialisierten Rechner, gesendet und dort protokolliert. Entsprechend den Anforderungen ist alles sehr flexibel gehalten, um wechselnden und erweiterten Anforderungen einfach gerecht werden zu können.

Ein Erfolg für FORTH

Über 2000 interessierte Besucher, davon 348 Kongressteilnehmer und 42 namhafte Aussteller kamen zur *Echtzeit90*, ein durchaus erfolgversprechender Auftakt. Den FORTH-Aktivlern sei Dank für diese ganz wesentliche Maßnahme, gerade auch für die FORTH-Public Relations. 1991 wird es weitergehen: am 11. bis 19. Juni gleichzeitig zur 'Meßtechnik Süd' wieder in Sindelfingen. (Kontakte: Ludwig Debringer, Agentur für technische Fachkongresse; ☎ 089 - 333 0 333).

✉ Arndt Klingelnberg
☎ 02104 - 6 16 48

KURZMITTEILUNG

Zimmer FORTH v.3.50a.k.

von Arndt Klingelnberg

Eine überarbeitete Version des **F-PC** wird gerade duplizierbereit auf Diskette gebannt. Neben der Original Version 3.50 ist als Komplettpaket eine überarbeite Version 3.50a.k., eine sofort lauffähige Demodiskette dazu und eine Diskette mit DOS-Utilities verfügbar. Weiterhin ein COMforth Demo (Rostock) und das JEDI Turbo-FORTH (France).

Zimmer FORTH zeichnet sich durch folgende Merkmale aus: Inkompatibilität zu volksFORTH (was aber eher volksFORTH angelastet werden muß), weitgehende Kompatibilität zu F83 (L&P), LMI, insbesondere zu JEDI, beim Assembler zu F83 (L&P) und zu MASM (!), komfortabler WS-angepaßter Editor (damit schreibe ich gerade Artikel für die 'Vierte Dimension'), Streamfiles, Hyper-Hilfe-System, Hard- und SoftwareFloatingPoint, 16-Bit-String-Package, Meta- und Target-Compiler, Nutzung von Extendend Memory und EMM sowie effektive Nutzung des DOS Memories durch getrennte Segmente für Code, Header, List ('beliebig' groß), Buffer, Editor etc., hohe Kompilier- und Ausführungsgeschwindigkeit, Kompatiblität bei Code-Definitionen mit gewissen Einschränkungen durch Segmentwechsel und *direct-threaded* Code.

Die überarbeitete Version bietet gelöste Bugs und ein weiter optimiertes Human-Interface, dazu gehört ein völlig überarbeitetes Pull-down-Menü- und Funktionstasten-Set (auf Original umschaltbar!), das nun endlich deutsche Umlaute im Editor erlaubt, dazu erweiterte und noch komfortablere Hilfen, eine verbesserte Mouse-Nutzung, Line-Editor mit History, weitere erschlossene Editierhilfen etc.

Eine Einschränkung für manche (für mich ein Vorteil): bis auf einige Hilfstexte von mir in gemischtsprachiger Schreibweise ist alles in Englisch gehalten. Total umfaßt das Ganze ca. 5 MB, die weitgehend gezippt sind, das sind also normal über 10 MB. Darin enthalten ist allerdings die Original- und die komplette überarbeitete Version. Eine Harddisk ist sehr empfehlenswert. Allerdings betreibe ich es auch auf einem Portable mit 2*720 kB, was den Komfort und die Möglichkeiten, speziell bei System-Generierungen einschränkt. Es sollte zumindest ein Zugriff auf eine HD-Installation möglich sein.

Die Distribution inkl. Kostenerstattung wird noch geklärt. Die Auslieferung erfolgt vorzugsweise auf 1M44 Scheiben (am kostengünstigsten und mit dem eindrucksvollsten Demo natürlich).

Anfragen an:
Arndt Klingelnberg (CCD2018//F1290)
e-mail mailbox box:geo1:klingelnberg
fax/modem ++49 +2404 - 6 30 39
(may be not connected ALLtime)
voice ++49 +2404 - 6 16 48
mail/visiting strassburgerstr. 12 ; D-5110 alsdorf ; F.R.Germany

Die nächste 'Vierte Dimension' erscheint im Dezember '90

BRIEF AUS DER PROVINZ

von Friedrich Prinz

Quelltext Service

Liebe FORTH-Freunde,

nachdem wir hier in Moers unsere 'Handicaps' abschütteln konnten und wieder voll und ganz in das FORTH-Leben eingestiegen sind, ist es wohl wieder einmal Zeit für einen 'Brief aus der Provinz'.

Zu Anfang des Jahres, spätestens aber ab Anfang Frühling '90, haben wir uns hier etwas schwer getan. Einige Mitglieder der örtlichen FORTH-Gruppe waren beruflich und/oder schulisch sehr stark eingespannt. Das Moerser Arbeitslosenzentrum, in dem wir unsere 'Clubräume' haben, wurde renoviert und umgebaut. Wir waren gezwungen unsere Treffen zunächst 14-tägig, später sogar nur noch monatlich zu organisieren.

Seit dem Anfang der Sommerferien NRW ist hier aber wieder alles 'in der Stunde'. Unsere Treffen finden wieder wöchentlich statt, an jedem Samstag in den früher schon beschriebenen Räumen des MALZ. Hier halten wir auch wieder FORTH-Kurse ab. Am 04.08.90 beginnt ein neuer Einsteigerkurs, eine Woche später führen wir unseren Kurs für Fortgeschrittene weiter.

Zum Einsteigerkurs haben wir bereits mehr als 10 Anmeldungen vorliegen. Wieviele Leute sich darvon entscheiden werden, FORTH zu ihrem Hobby, und ein wenig auch zu ihrer Philosophie zu machen, kann natürlich jetzt noch niemand sagen. Nach den guten Erfahrungen aus unseren lokalen Anfängen glaube ich aber, daß auch daraus wieder einige potentielle Mitglieder für die FORTH-Gesellschaft dabei sind. Wir haben in der örtlichen Presse kräftig die Werbetrommel für FORTH gerührt und unsere Kursangebote in verschiedenen Tages- und Wochenzeitungen veröffentlichen lassen. Dabei wurden wir ganz besonders vom Arbeitslosenzentrum unterstützt. Unterstützung erfahren wir auch durch den DPWV, den Deutschen Paritätischen Wohlfahrtsverband, der in diesem Herbst bereits zum zweiten Male einen unserer Kurse bezahlt. Genaugenommen bezahlt der DPWV dem Arbeitslosenzentrum als dem offiziellen Ausrichter die Unterrichtsstunden des Kurses für Fortgeschrittene - das MALZ vergütet mir, als dem Kursleiter, die angefallenen Stunden. Im vergangenen Jahr haben

wir für dieses Geld, nach entsprechender Diskussion in der Gruppe, einen LMI-Compiler, PC/FORTH+ als Komplettsystem für die Gruppe angeschafft. Dabei haben wir, dankenswerter Weise, von der Firma Flesch einen Ausbildungsrabatt bekommen. Was wir mit dem in diesem Jahr 'einkommenden' Geld machen werden, haben wir noch nicht diskutiert. Selbstverständlich werden wir wieder etwas anschaffen, was wir für unsere Gruppe brauchen können.

Auch außerhalb der Gruppe sind die einzelnen Mitglieder recht aktiv in Sachen FORTH. Michael Major, der zur Zeit im Arbeitslosenzentrum ein Praktikum für sein Studium absolviert, hat seinen Rechner in das MALZ geschleppt und bringt dort den Leuten MSDOS, Textverarbeitungen unter GEM und natürlich FORTH näher. Martin Wissusek, der wöchentlich 70 km aus Heiden anreist, hat für die Gruppe Buttons gefertigt. Davon lege ich Euch einige zur Begutachtung bei. Bernd Feuerer, seines Zeichens Drukker, wird in den nächsten Wochen Briefbögen für die FORTH-Gruppe Moers anfertigen. Ulrich Prinz, der zwei Jahre lang als Geschäftsführer des MALZ gearbeitet hat und als 'alter Gewerkschafter' über ausgezeichnete Kontakte zu verschiedenen Einzelgewerkschaften des DGB verfügt, hat uns in diesem Jahr erstmals zum 1. Mai mit dem DGB in Verbindung gebracht. Wir waren auf der Maikundgebung mit einem Zelt und zwei Rechnern vertreten, an denen nach dem offiziellen Teil der Kundgebung vor allem die anwesenden Kinder bis zum frühen Abend großen Spaß hatten. Im kommenden Jahr wollen wir das ausbauen und versuchen auf diesem Wege weiter 'Werbung' für FORTH zu machen.

Wir machen also alles in allem viel 'Kleinkram', keine großen, weltbewegenden oder sonstwie spektakulären Sachen. Wir tun halt, was uns Freude macht - und wovon wir glauben, daß es der Gruppe und FORTH nutzt. Den Spaß an der Sache zu behalten, ist deshalb auch unser oberstes Ziel. Die Mitglieder der Gruppe pflegen auch privat zum Teil recht intensive Kontakte. An besonders heißen Tagen verlegen wir die Kursstunden häufig in einen Biergarten an den Rhein und um die bis dahin hoffentlich bereits geknüpften Kontakte zu unseren 'Neuen' zu intensivie-

wenn forth zu klein ist:

(für alle Atari ST)

bigFORTH hat eine lange Entstehungsgeschichte hinter sich. Seinen Ursprung hat es im PD-System volksFORTH. Ziel der Entwicklung war ein FORTH-System, das in seiner Leistungsfähigkeit nicht hinter modernen und professionellen Compilern für "konventionelle" Sprachen zurücksteht. Dieses Ziel kann man als erreicht ansehen, wie ein Blick auf einige der Features sicher zeigt:

- **Mächtiger Compiler:**
  32-Bit-Stacks, erzeugt optimierten 68000-Native-Code, relokatibles System (ohne Einschränkungen!), weitgehend FORTH-83-kompatibel, unbeschränkter Adreßraum, nutzt Screen- und Stream-Files, relokatible Turnkey-Applikationen (denen man FORTH nicht mehr ansieht) sind problemlos machbar...

- **Vielseitige Tools:**
  Assembler, Disassembler, Multitasker, sourcefähiger Decompiler (decompiliert alle Optimierungen), Source-Level-Debugger (Single-Step und Trace, beliebig viele Breakpoints), Post-Mortem Dump und Returnstack-Trace bei allen Fehlern (Ausgang über ABORT"), resetfest (für den Ausstieg aus der Endlosschleife), Multiwindow-Editor (natürlich unter GEM), beliebiger anderer Editor nutzbar...

- **Umfangreiche Libraries:**
  Komfortables Fileinterface, Druckertreiber, sämtliche TOS-Funktionen (GEMDOS, BIOS, XBIOS, GEM-AES, GEM-VDI und Line-A-Grafik), Floating-Point-Arithmetik, leistungsfähiges Memory Management, High-Level-GEM-Libraries, Turtle Graphic und mehr...

- **Transparenz:**
  Sämtliche Sourcen (einschließlich Kernal) auf Diskette, Target-Compiler wird mitgegeben...

Die ausführliche deutsche Dokumentation (über 250 Seiten Handbuch) wird Anfängern sicher ihre ersten Schluckbeschwerden bei dem großen Bissen beseitigen, aber auch alte Hasen werden ihren Nutzen daraus ziehen...

Und das alles nur für
DM 200.- (incl. Mehrwertsteuer und Versand)

Für Vorsichtige gibt es eine frei kopierbare Demoversion (ohne Sourcen&Handbuch), für eine Schutzgebür von
DM 10.- (incl. Versand)

Bestellungen bitte schriftlich an:
Bernd Paysan
Stockmannstr. 14
D-8000 München 71

ren, wollen wir kurz vor Weihnachten ein gemeinsames Weihnachtsfest feiern.

Selbstverständlich steht bei all diesen Dingen FORTH immer im Vordergrund. Mittlerweile sind einige der ehemaligen Eleven recht 'fit' in FORTH geworden, was sich zum Teil in einigen 'Tools' niederschlägt, die von unseren Gruppenmitgliedern geschrieben worden sind. Einige dieser 'Werke' schicke ich Euch auf einer Diskette mit. Ich überlasse es der VD-Redaktion zu beurteilen, ob Ihr diese Tools (Schaltungen Verdeutlichen, Disketten-Archiv, DEBUG für LMI-FORTH, Speichermonitor) veröffentlichen wollt (*Anm.d.R.: die Quelltexte sind über den Quelltext-Service erhältlich*). Grundsätzlich stellen aber alle Autoren ihre Quelltexte dem von Euch angesprochenen PD-Pool zu Verfügung.

✉ Friederich Prinz,
Hombergerstraße 335,
4130 Moers 1

Testbericht: bigFORTH

Umsteigen vom Käfer auf Daimler möglich

von M. Schultheis

Nach langem Warten tritt bigFORTH jetzt als 32-Bit-Erweiterung des weitverbreiteten volksFORTH für den Atari ST an. Das screen-orientierte System ist eine der modernsten Implementationen, die zur Zeit bei uns zu einem Hobby-Preis erhältlich sind. Der Autor Bernd Paysan hat verschiedene Aktivitäten geschickt gebündelt und marktreif gemacht. Wer hier wieder ein Public-Domain-Produkt erwartet, der sei daran erinnert, daß der Preis für das bigFORTH wirklich nur ein Anerkennung und eine bescheiden Aufwandsentschädigung für das Gebotene ist.

Das Handbuch zum bigFORTH ist sehr klar gegliedert und umfaßt neben der Beschreibung der ca. 2000 Worte, einen FORTH-Kurs für Anfänger, ein Tutorial für Umsteiger und viele allgemein nützliche Informationen.

Das System befindet sich auf zwei doppelseitigen Disketten und enthält auch alle Sourcen, sogar mit Targetcompiler, und vielen Libraries zu allen wichtigen TOS-Schnittstellen. Vervollständigt wird das System durch eine Turtle-Graphik, einen Debugger und ein Gleitkommapaket.

An dieser Stelle kann daher nur ein subjektiver kurzer Eindruck über bigFORTH vermittelt werden. Das bekannte `VIEW` zusammen mit einem Shadow-Screen auf dem SW-Monitor erreicht fast Hypertext-Qualität. Der praktische volksFORTH-Editor wurde verbessert. Sehr angenehm ist z.B. die Darstellung von zwei vollen Screens aus beliebigen Files, die mit der Atari File-Selection-Box bequem ausgewählt werden können. Im bigFORTH entfaltet das verbesserte volksFORTH-Multitasking seine wahren Qualitäten, die bei 64kByte meist rasch ihre Grenzen fanden. Die neuen Pipes zur Kommunikation zwischen mehreren Tasks sind Stand der Technik. Der volksFORTH-Heap wurde beibehalten und hilft die Programmgröße durch Benutzen temporärer Worte, trotz der Optimierungen, in Grenzen zu halten. Eine eigenständige Speicherverwaltung ist für Multitasking ausgelegt und ergänzt dieses große System.

Den *Streamfilern* sei verraten, daß alle notwendigen Worte bereits als Source mitgeliefert werden und der Autor eine entsprechende Editor-Anbindung als Update angekündigt hat. Das bigFORTH ist aus Subroutinen und makrogenerierten Teilen aufgebaut. `DO - LOOP` benutzt Register für die innere Schleife. Eine sog. Peephole- Optimierung verbessert die Laufzeit nochmals durch die Verhinderung unnützer Register-Stack-Register-Befehle an der Grenze zwischen zwei FORTH-Worten. Die Programm-Ausführungszeiten bei 32-bit-Worten liegen bei nur etwa 30 bis 40% im Vergleich zum volksFORTH mit 16-bit-Worten. Aus diesen Gründen ist natürlich kein reines F83-FORTH mehr möglich, aber der Decompiler erkennt meistens trotzdem die richtigen Worte. Bei der Definition von Adressen z.B. für das Wort `PERFORM` sind besondere Vorkehrungen zu treffen, die ausführlich erklärt werden.

Der Puffer für nur eine Kommandozeile ist etwas sparsam ausgefallen und zwingt zum öfteren Tippen gleicher Sequenzen. Die Verwendung der Funktionstasten mildert dies aber ab. Die Eigenwilligkeiten des volksFORTH beim Fileinterface sind weiterhin vorhanden, und leider nicht durch Standardworte z.B. aus dem Unixbereich ergänzt worden. Aber das kann vielleicht die erste Übung für den stolzen bigFORTH-Besitzer sein, wenn er sich sein Traumsystem aufbaut (siehe auch Artikel BlocksWorld in diesem Heft).

✉ M.Schultheis
Meringerzell 15
8905 Mering

FORTH-Gesellschaft e.V. intern

Auswertung der schriftlichen Abstimmung

von Ulrike Schnitter

Liebe Mitglieder der FORTH-Gesellschaft e.V., die Auszählung der Stimmzettel der schriftlichen Abstimmung am 25.07.1990 ergab folgendes Ergebnis:

- Bei der Briefwahl wurden insgesamt 78 Stimmzettel zurückgeschickt, davon war 1 Stimmzettel ungültig.
- Entlastung des Direktoriums für das Vereinsjahr 1989.
  73 Ja 0 Nein 4 Enthaltungen
- Wiederwahl der Direktoren: Christoph Krinniger, Johannes Reilhofer und Heinz Schnitter.
  72 Ja 0 Nein 5 Enthaltungen

Damit ist das Direktorium entlastet und die Wahl des Direktoriums bestätigt.

Wahlausschluß: Johannes Teich, Josef Witzl

Neue DFÜ-Aktivitäten

von Heinz Schnitter und Johannes Teich

Die FORTH-Gesellschaft e.V. weitet ihre Aktivitäten auf dem Gebiet Datenkommunikation aus. Erfahrungen mit unserer Münchner und Murnauer FORTH-Box haben gezeigt, daß unter den Mitgliedern der FG eine rege Nachfrage besteht. Einziger Hinderungsgrund allzu eifrig die Box anzurufen sind die Telefonkosten, da für die meisten Mitglieder Gebühren der Fernzone 3 anfallen. Um allen Mitgliedern die Teilnahme an der Datenkommunikation zu ermöglichen, haben wir uns ein neues Konzept ausgedacht.

Wir werden deshalb ab September/Oktober 1990 möglichst viele Großstädte der Bundesrepublik mit Informationen versorgen. Die Verteilung der Informationen übernimmt infiNet, ein aktiver Knotenrechner des Zerberus-Netzes mit Standort München. Die Gebühren für die Verteilung der Nachrichten auf die einzelnen Rechner übernimmt die FORTH-Gesellschaft. Dieser Weg ist günstiger und sicherer als ein eigenes Netz aufzuziehen, da man hier auf eine bestehende und funktionierende Infrastruktur zurückgreifen kann. Außerdem braucht die FG keine Investitionen zu tätigen, die sich in der Regel auf ca. 3.000,- DM pro Knotenrechner belaufen. Hinzu kämen noch die Gebühren für Telefon und Unterhaltskosten.

Was bietet das neue FORTH-Netz?

Die Mailbox in Murnau bleibt bestehen und ist an dem *Backbone unido* in Dortmund angeschlossen; d.h. die FORTH-Gesellschaft e.V. ist Mitglied im EUnet und erhält eine weltweite Adresse. EUnet ist der europäische Teil eines weltweiten Netzes von Rechnern, in der Regel mit dem Betriebssystem UNIX™, die sich des Kommunikationsprotokolls UUCP bedienen. Das amerikanische USENET ist ebenfalls ein Bestandteil des Netzes und bedient Nordamerika. Die weltweite Adresse der FORTH-Gesellschaft e.V. heißt:

```
forthev.UUCP
```

forthev ist auch der Name der Box in Murnau. Alle Mitglieder der FG haben die Möglichkeit die EUnet-Dienste über die Murnauer Mailbox in Anspruch zu nehmen:

- ✗ Elektronische Post per Electronic Mail
- ✗ Lesen und Schreiben der News.

Stichworte:

- ✗ Datenkommunikation,
- ✗ EUnet,
- ✗ UUCP,
- ✗ Zerberus-Netz

E-Mail ist innerhalb Deutschlands kostenlos, europa- und weltweit wird für die ein- und ausgehende Post eine geringe Gebühr berechnet.

Das News-System stellt ein verteiltes weltweites Konferenzsystem dar. Wir bieten die Newsgroup **comp.lang.forth** in unserer Box an. Diese Newsgroup enthält Nachrichten aus fast allen FORTH-Netzen der USA. So kann man hautnah die Diskussionen um das ANS-FORTH in der Box mit verfolgen, ja es besteht sogar die Möglichkeit aktiv daran teilzunehmen.

Damit dieses Vorhaben ein Erfolg wird, brauchen wir Ihre Hilfe. Falls Sie Interesse haben, von diesem Service Gebrauch zu machen, bitten wir Sie, eine Zerberus-Box in Ihrer Nähe zu suchen und mit dem dortigen Sysop zu vereinbaren, daß er die vier Bretter der FORTH-Gesellschaft übernimmt. Die FG wird dann über infiNet *Ihre* Box mit dem Datenaufkommen versorgen und übernimmt die Kosten. Wichtig ist hierbei, daß eine gewisse Mindestzahl an Mitgliedern dafür Interesse zeigt.

Anschriften von Zerberus-Boxen, die dazu bereit sind, können beim FORTH-Büro erfragt werden. Die FORTH-Box in München wird noch bis Ende 1990 in Betrieb sein, auch dort kann man sich informieren. Grundsätzlich werden beide Boxen der FORTH-Gesellschaft über den Stand unseres Projektes berichten.

Zur Gestaltung Ihrer Adresse machen wir Ihnen folgenden Vorschlag: Angenommen Donald Duck möchte an der Münchner Zerberus-Box infiNet einen Account; es wäre günstig, folgende Adresse zu wählen:

```
D.Duck_FG@infiNet
```

Es sind einschließlich **_FG** 20 Buchstaben erlaubt. Mit dieser Adresse weiß der Sysop, daß D.Duck Mitglied der FG ist und beim FORTH-Büro überprüfen muß, ob die Mitgliedsnummer ok ist. Außerdem hat diese Adresse den Vorteil, daß D.Duck an der Murnauer Box eine weltweit adressierbare Anschrift: **D.Duck@forthev.UUCP** beantragen könnte. Auch für ein zukünftiges Gateway zwischen dem Zerberus- Netz und unserer Box wäre diese Adressierung günstig.

Die Bretter im Zerberus-Netz sind folgendermaßen organisiert:

- **forthev/info**: öffentlich, readonly; für Infos, Hinweise auf was es sonst noch gäbe, wenn man doch nur Mitglied wäre.
- **forthev/forum**: öffentlich, read/write; Diskussionen
- **forthev/news**: nur für Mitglieder, readonly; comp.lang.forth usw.....
- **forthev/files**: nur für Mitglieder, readonly; Quellen, FORTH-Systeme, Kurse usw.....

Wir bitten alle Mitglieder diesen Service in Anspruch zu nehmen. Bei der nächsten Mitgliederversammlung werden wir diese Einrichtung diskutieren, beurteilen und darüber abstimmen ob wir sie beibehalten, erweitern oder aufgeben sollen.

Zum Inhalt der News

Es handelt sich um Mitteilungen, Diskussionen und Anfragen zum Thema FORTH. Gelegentlich sind ganze Programme, ja sogar komplette FORTH-Systeme darin zu finden.

Einen gewichtigen Teil nimmt der Fortgang der Normungsarbeiten für den ANS-FORTH-Standard ein. Die Ergebnisse der im vierteljährlichen Turnus stattfindenden Treffen des *X3J14*-Teams werden hier veröffentlicht und eifrig kommentiert, gelobt, angegriffen und verteidigt. Applikations-Programmierer mögen mit Ungeduld auf die endgültige Fassung warten - aber aus dem Ringen darum ist mehr über das Wesen von FORTH zu erfahren als aus dem zu erwartenden Ergebnis.

An den Diskussionen sind Vertreter der verschiedensten Richtungen beteiligt: die Puristen, die allen Änderungsvorschlägen mit Mißtrauen begegnen (und kämen sie vom Erfinder selbst); die Modernisten, die FORTH am liebsten umkrempeln möchten; die Minimalisten, denen der Grundwortschatz schon zu üppig ist, die Portabilisten, die FORTH mit C und UNIX™ vermählen wollen, sowie die Steuerungs-Experten, denen es vor allem auf Schnelligkeit ankommt und die verlangen, daß der Standard die Architektur der FORTH-Chips und "FORTH im ROM" berücksichtigt. Die Diskussionen sind erfreulich sachlich und fair, kompetent und oft auch sprachlich exzellent.

Aber neben dem Standard gibt es genügend andere Themen: FORTH-Literatur (Brodie's zweites Buch läuft aus!), verfügbare Systeme, interessante Anwendungen. Oder Anfragen aus den Reihen des (auch weiblichen) Nachwuchses, die sehr entgegenkommend beantwortet werden. Unter dem Titel "Distant Drums" wurde der Brief eines Schülers aus der DDR zitiert, der den Zugang zum Netz ausgerechnet über Australien gefunden hatte. Auch ein "Russian Forthman" (Sergei Baranoff) meldet sich zuweilen.

Eine Warnung muß allerdings ausgesprochen werden: die Menge der Informationen ist beachtlich. Es hat also wenig Sinn, die News *auf Halde* zu legen, um sie sich in der stillen Jahreszeit vorzuknöpfen. Seit Beginn dieses Jahres wird in die Newsgroup comp.lang.forth das GEnie ForthNet und FIGI-L, ein Listserver der FORTH Interest Group International, eingespeist, was zu einer Verdoppelung des Angebotes führte, aber auch etliche bekannte Namen aus der FORTH-Welt zutage förderte.

Zu den eifrigsten Teilnehmern im ForthNet zählen das British Columbia FORTH Board und das LMI FORTH Board. Wer Aufsätze und Bücher von Ray Duncan kennt, die FORTH nicht erwähnen, kann sich hier überzeugen, daß er ein leidenschaftlicher FORTHler geblieben ist. Weitere Namen, die immer wieder auftauchen: Mitch Bradley, Doug Philips, Robert Berkey, John Wavrik, Jack Woehr, Philip Koopman, Peter da Silva, Frank Sergeant, William Bouma, Dennis Ruffer, Wil Baden, Jack Brown, Charles Eaker, Martin Tracy - um nur einige zu nennen. Allerdings: Chuck Moore schweigt. Aber man darf vermuten, daß er sich die Lektüre nicht entgehen läßt.

✉ Heinz Schnitter,
Nelkenstr. 52,
8044 Unterschleißheim
✉ Johannes Teich,
Hauptmann-Bauer-Weg 16,
8110 Murnau

LINKED ACTIONS

Teil I

von Jörg Plewe, Frank Stüss, Jörg Staben

Quelltext Service

Stichworte

- ✗ objektorientierte Programmierung,
- ✗ Listen,
- ✗ Rekursion

Ziel des Beitrags

Grundsätzlich zeigt dieser Beitrag, wie man das Verhalten eines Wortes ändern kann, ohne den Umweg über eine Neudefinition und damit ein Neukompilieren zu gehen. Darüberhinaus möchte der Beitrag noch einmal das Eingabeverhalten des volksFORTH darstellen. Und, vielleicht das Hauptziel, es sollte mal wieder auf der ganzen Klaviatur eines FORTH-Compilers gespielt werden, um der Langeweile marktbeherrschender Compiler etwas entgegenzusetzen.

Im einzelnen werden, in der Reihenfolge ihres Auftretens, angesprochen:

- objektorientierte Programmierung
- Listen als Datenstrukturen
- Vorwärtsreferenzen
- Wiederverwendbarkeit von Code
- Rekursion
- Schnittstelle zur Maschinensprache

Gibt es denn in FORTH überhaupt Probleme?

Das Problem kristallisierte sich beim Einsatz des Wortes **BYE** heraus, denn viele Programme führen beim Beenden notwendige De-Initialisierungen aus. Um dem Betriebssystem einen sauberen Bildschirm zu hinterlassen, ergänzen viele Programme das Wort BYE in dieser Weise:

```
: bye  ... page bye ;
```

Allerdings schätzen nur die wenigsten FORTH-Anwender die Statuszeile des volksFORTH auf der DOS-Ebene und schon ändert sich BYE erneut:

```
: bye  status off  bye ;
```

Nun müssen auch noch einige andere Werte zurückgesetzt werden, dies macht vielleicht S_BYE, womit die nächste Redefinition von BYE ins Haus steht:

```
: bye  s_bye bye ;
```

Damit dürfte dann der letzte Rest an Übersicht verloren gegangen sein, was BYE nun alles macht, wo doch ein VIEW BYE nur das zuletzt definierte zeigt. Nicht ganz so schlimm ist es beim Booten eines Programms, wenn Initialisierungsroutinen ausgeführt werden. Hier ist **'COLD** das Wort, in das bestimmte Aktionen eingehängt werden. Meist fängt es ganz harmlos an [1]:

```
Create logo
," STABEN Software Service S-S-S"
:Does> count type ;

' logo Is 'cold
```

Dann müssen aber doch noch einige Werte auf Null gesetzt werden, dies macht bestimmt. VAR_INIT. Und schon erfährt **'COLD** die erste Änderung:

```
: var_init
   queue off  counter off ... ;

' var_init Is 'cold
```

Soweit, so gut. Aber weil vergessen wurde, **LOGO** in **VAR_INIT** einzutragen, taucht während des Programmlaufs nirgendwo der Schriftzug vom *Software Service* auf. Von diesem Aspekt sind vor allem Zeiger auf Worte betroffen, deren Inhalt ständig umgesetzt wird. Damit geht das vorherige Verhalten des Wortes vollständig verloren, statt lediglich geändert zu werden.

Ähnliche Probleme treten auch mit dem Eingabezeilen-Editor CED im volksFORTH auf. Beim Laden des CED wird (DECODE neu gesetzt und der Eingabezeilen-Editor damit aktiviert. Setzt nun ein später geladenes Wort wiederum (DECODE neu, ohne die Erweiterungen zu berücksichtigen, so wird der Eingabezeilen-Editor schlichtweg ausgehängt. Überhaupt ist (DECODE das Wort, an dem die Problematik des Änderns von Aktionen deutlich wird.

Leben und Leiden von (DECODE

(DECODE - *sein Leben ...*

Darf ich Ihnen zuerst einmal (DECODE vorstellen? Sie finden es im Feld KEYBOARD, das die Eingabeworte des volksFORTH enthält. (DECODE wertet ein Zeichen an einer bestimmten Position in einem Pufferspeicher aus, ob dadurch eine bestimmte Aktion ausgelöst wird, wie bei Backspace **#bs** oder Carriage Return **#cr** oder ob das Zeichen lediglich ge-echo-t in diesem Pufferspeicher hinterlegt wird. Wie sensibel der Bereich um (DECODE ist, zeigt die Tatsache, daß das *originale* (DECODE des volksFORTH dies zuläßt:

```
pad c/l  Ascii A (decode  type
```

Das *neue* (DECODE des CED dagegen quittiert eine solche Eingabe mit einem Systemstillstand, da in dem Wort INS die Position im Puffer vom Variableninhalt SPAN subtrahiert und das Ergebnis einem CMOVE übergeben wird. Bei einem negativen Argument für CMOVE reagiert das FORTH dann halt FORTH-typisch. Zusammen mit KEY finden Sie (DECODE in (EXPECT wieder; dieses nimmt lediglich Zeichen

in einem Pufferspeicher entgegen. Somit muß nach einer Modifikation von (DECODE auch (EXPECT neu definiert werden, da sonst immer das *alte* (DECODE ausgeführt wird. Aus dem gleichen Grund muß dann auch der Eingabevektor KEYBOARD erneut definiert werden.

Gesetzt der Fall, Sie haben mit viel Mühe ein Wort **HELP** entworfen, das dem Benutzer eine zutiefst beeindruckende Hilfe bietet und mit **-&59 Constant #F1** die F1-Taste als Opfer auserkoren, dann werden Sie liebevoll in (DECODE eine zusätzliche Zeile einfügen **... #F1 case? IF help exit THEN ...** und (EXPECT erneut definieren, damit (EXPECT von dem neuen Verhalten von (DECODE Kenntnis nimmt.

Abschließend schreiben Sie den Inputvektor KEYBOARD erneut hin, damit das neue (DECODE und das neue (EXPECT ausgeführt werden und achten dabei auf die vorgeschriebene Reihenfolge von KEY und KEY? etc.

```
Input: keyboard
       (key (key? (decode (expect ;
```

Nach dieser handwerklichen Meisterleistung werden Sie frohgemut feststellen, daß ab jetzt die Funktionstaste F1 unterstützt wird.

... *und sein Leiden*

Das Ganze funktioniert aber nur solange, bis der sehr schöne und eigentlich unverzichtbare Kommandozeilen-Editor CED kommt. Dieser bringt sein eigenes (DECODE und (EXPECT mit und zwingt diese beiden in brutalster Form dem System auf:

```
' (decode  ' keyboard  6 +  !
' (expect  ' keyboard  8 +  !
```

Somit sind Ihre Schöpfungen aus dem Feld der Eingabeworte sang- und klanglos verschwunden und damit auch die Unterstützung der Funktionstaste F1.

Endlich!! Da ist das Problem: Es gibt häufig Situationen, in denen man das Verhalten eines Wortes elegant erweitern oder reduzieren möchte, ohne die bestehende Arbeitsweise zu beeinträchtigen.

Die bisherigen Möglichkeiten

Redefinition

Die nachfolgenden Techniken werden vielen unter den Lesern schon bekannt sein. Sie sollen hier aber dennoch vorgestellt werden.

Prinzipiell gibt es - wie immer in FORTH - mehrere Möglichkeiten, das Verhalten einer Definition zu ändern oder zu erweitern. Die einfachste dürfte die Redefinition sein. Soll z.B. WORDS um eine zusätzliche Aktion wie ein Löschen des Bildschirmes erweitert werden, läßt sich

```
: key  ( -- char)
   : words ( --)  cls words ;
```

definieren, wobei die Meldung WORDS EXISTS geflissentlich überlesen wird.

Patchen

Eine andere Möglichkeit ist das 'Patchen' einer Definition, indem der Zeiger auf den ausführbaren Code (die Code Field Adresse CFA) umgesetzt wird. Das Laufzeitverhalten des Wortes wird damit komplett geändert:

```
' words >body @  Constant old_words
\ sichert altes WORDS

: new_words       rdrop cls ;

' new_words  ' words >body !
\ nun machts WORDS nur noch CLS,
\ aber kein WORDS
```

Der grundlegende Unterschied ist, daß beim Redefinieren alle vorherigen Aufrufe von WORDS unberührt bleiben, d.h., die älteren Aufrufe von WORDS arbeiten noch *normal*. Dagegen ändert das Patchen auch die bisherigen Aufrufe von WORDS, indem immer NEW_WORDS ausgeführt wird. Ein

```
old_words  ' words >body !
       \ WORDS macht jetzt
       \ wieder WORDS, aber kein CLS
```

macht das Umsetzen der ersten CFA rückgängig.

Tip: Auf einem System mit Decompiler (SEE etc.) wird nach einem SEE WORDS diese Änderung der ersten CFA korrekt dekompiliert und damit anschaulich, was da passiert. Versucht man, das Laufzeitverhalten zu ergänzen, so käme so etwas in Frage, aber besonders schön ist es nicht:

```
' words body @  Constant old_words

: new_words    cls   old_words
execute  ;
' new_words    ' words body !
```

Zusätzliches EXIT

Ist dem Programmierer von vornherein klar, daß ein Wort sein Verhalten im Laufe seines Daseins ändern muß, so kann man für weitere Aktionen Platz reservieren.

Dies wird mit nur einer varianten Aktion in dem Wortpaar NAME und 'NAME vorgeführt:

NAME endet mit **... exit ;** Durch das EXIT vor dem Semikolon, das ebenfalls ein EXIT ist, wird das Ende des Wortes künstlich herbeigeführt und das Semikolon so nie erreicht. Durch simples Abzählen (NAME enthält das EXIT an sechster Position) macht 'NAME das freie Plätzchen in NAME verfügbar, wovon SHOWLOAD und SHOW im Editor regen Gebrauch machen, indem mit

```
... ['] show 'name ! ...   oder
... ['] exit 'name ! ...
```

einmal die zusätzlich auszuführende Aktion SHOW oder dann wieder das EXIT an dieser Position in NAME eingetragen werden.

Defer / Is

Weiß man dagegen vorher nicht, was einem Wort alles widerfahren kann, hält man sich alle Möglichkeiten offen, indem man es als DEFER-Wort anlegt, dessen aktuelles Verhalten durch **Is** immer wieder neu bestimmt werden kann:

```
Defer words    ' old_words Is words
               ' new_words Is words
```

Auch hier wirkt eine Änderung auf alle bisherigen Aufrufe, aber die Teile, die ALT_WORDS und NEW_WORDS gemeinsam sind, befinden sich doppelt im Speicher. Weiterhin hat diese Lösung den Nachteil, daß genau wie beim Patchen das Verhalten des Wortes nicht modifiziert (wie bei NAME/'NAME), sondern jedesmal komplett umgesetzt wird.

Tom Zimmer's Defers/Undefers

Da jeder irgendwann mal die gleichen Probleme löst wie alle anderen vor ihm, hat Tom Zimmer im F-PC natürlich auch schon eine Lösung für das Problem, daß der Funktionsumfang eines Wortes mit steigenden Anforderungen wachsen muß. Seine Technik des "background task chaining" benutzt ein DEFERred Wort als Ausgangsbasis. Wird nun ein neues, erweitertes Wort definiert, so bindet DEFERS den alten Funktionsumfang ein. Als Beispiel mag das KEY, ein *User deferred word*, dienen. Das Wort KEY soll nun dahingehend erweitert werden, daß zur Zeicheneingabefunktion KEY noch eine Zeitanzeige oben rechts im Bildschirm ausgeführt wird.

```
DEFER BGSTUFF
' NOOP IS BGSTUFF

: (KEY)
   DEFERS BGSTUFF
   [ the words to do (KEY) ]  ;

' (KEY) IS BGSTUFF   \ BGSTUFF = (KEY)

: TIME
   DEFERS BGSTUFF    \ führt (KEY) aus
   #OUT @   #LINE @ \ sichert cursor
                     \ Position
      64 0 AT .TIME \ Zeitanzeige
                     \ oben rechts
      AT   ;         \ cursor -
                     \ alte Position

' TIME IS BGSTUFF    \ BGSTUFF =
                     \ (KEY) + .TIME
```

Mit dieser Technik kann man zwar das Verhalten eines Wortes dynamisch ändern, aber man muß mit den Defer/Defers/Undefers jonglieren und übersichtlich ist es - vor allem durch den allerersten Rückverweis - auch nicht. Also: Eine *total* andere Lösung muß her!

Das neue Konzept: FORTH Worte als verteilte Listen

Unser Ansatz ändert deshalb das Konzept der FORTH-Worte grundlegend. *Gewöhnliche* FORTH-Worte sind Listen von CFA's, die normalerweise hintereinander liegen. Hier dagegen werden die CFA's der FORTH-Worte in verketteten Listen gehalten, die es erlauben, diese Listen komfortabel zu ändern, zu erweitern, zu kürzen und selbstverständlich auch den Inhalt anzuzeigen. Solche Listen sind in FORTH nichts Ungewöhnliches - nicht nur die normalen FORTH-Worte, sondern auch die Vokabulare von FORTH sind Listen. Für die Entdecker unter den Lesern: VOC-LINK könnte der Anker einer Liste sein und VOCS muß notwendigerweise irgendwie auf diese Liste zugreifen. Es zeigt sich halt immer wieder: Der FORTH-Compiler selbst ist *die* exemplarische Anwendung von FORTH.

HICKS

Entschuldigung, es muß OOP heißen!

Damit überhaupt jemand diesen Beitrag liest, hatten wir ganz, ganz weit vorne dem geneigten Leser auch die publicity-trächtige objektorientierte Programmierung OOP in Aussicht gestellt. Versprochen ist versprochen und gehalten ist gehalten: Schon im ersten Abschnitt dieses Beitrages tauchte ganz kurz ein Objekt auf - das LOGO vom Software Service. Denn wie in der Literatur [2] klar beschrieben, faßt man in der OOP *Datenstrukturen* ***und*** *Algorithmen zu Objekten, also zu neuartigen Einheiten zusammen.* Mit dieser **Kapselung** ist schon das erste Planziel auf dem hehren Weg zur besseren Software-Entwicklung erreicht. Die weiteren Planziele der *Vererbung* und der *Polymorphie* werden wir hoffentlich auch noch erfüllen.

Wo kann man denn in FORTH 'was erben?

Die Vererbung ist eine der Grundfunktionen von FORTH; am nachfolgend angeführten definierenden Wort (defining word) für Logo's sieht man, daß alle *Instanzen* vom Typ LOGO: die gleiche Datenstruktur und die gleiche Zugriffsprozedur haben.

```
: Logo:  Create  ,"
         Does>   count type ;

  Logo:  Jörg    F - 68K "
  Logo:  Frank   DFF - FORTH "
```

Aber so wie eine Schwalbe keinen Sommer macht, eröffnet eine einzige Zugriffsmethode noch nicht die Möglichkeiten objektorientierter Programmierung. Doch die Einführung mehrerer Methoden eines Objektes - wie hier z.B. ein **put** für das erneute Zuweisen eines Strings - ist bereits ausreichend diskutiert worden [4],[5]. Damit ist die Polymorphie, die Vielgestaltigkeit von Methoden und Objekten für uns in dieser 'Vierten Dimension' kein Thema.

Im Vordergrund steht hier die Vererbung, deren Möglichkeiten im ersten Ansatz für die Listenwörter intensiv durchgespielt worden sind.

Unser erster Ansatz: defining defining words [siehe Quelltext 1]

Aus reiner Lust an der Bequemlichkeit sollte ein definierendes Wort entworfen werden, das sich seinerseits auf definierende Worte stützt, denn für alle generierten Listen sind die Aktionen gleich.

Aber fangen wir erst mal vorne an: volksFORTH ist ein System, wo man tunlichst die Verschwendung von Bytes vermeiden möchte; deshalb greift man gerne zu headerlosen Worten, die durch ein | vor der Definition gekennzeichnet sind. Damit kann man die Namen nicht mehr benötigter Worte nach dem Compilieren mit **clear** löschen und damit den von den Namen belegten Speicherplatz wieder freigegeben. Weil dieser Headerless-Mechanismus beim Verwirklichen komplexerer Ideen Komplikationen mit sich bringen kann, hat Frank dieses **?|** in die Welt gesetzt, so daß man mit **headers off/on** wählen kann, ob mit Headern compiliert werden soll:

```
Variable headers
headers off
: ?| ( -- )
   headers @ not IF | THEN ;
```

Claus Vogt dagegen redefiniert das |, um es wirkungslos zu machen:

```
\ headers no-headers      clv 01jun89
' | Alias old|
Defer |   ' old| Is |
: headers
   ?head off   ['] noop Is | ;
: no-headers
   ?head on    ['] old| Is | ;
```

Wie schon gesagt, in FORTH gibt's halt immer mehrere Möglichkeiten...

Code-sharing beim Anzeigen und Ausführen

Beim Entwickeln des Codes für das Ausführen und die Darstellung der Listen über eine Schleife fiel auf, daß bei-

de Aktionen sich nur in dem unterscheiden, was mit dem gefundenen Listenelement geschieht.

Da in der letzten Zeit gerade das sichere Erstellen und die Pflege von Quellcode im Mittelpunkt des Interesses stehen, wird der Code für den Zugriff nun doppelt genutzt.

```
?| Defer action
?| : all_links        ( firstlink --)
    BEGIN
       ?dup
    WHILE  dup cell+  action  @ REPEAT ;
\ perform ist als  @ execute
\ im Grundwortschatz enthalten.
?| : .id  ( cfa)
     @ >name .name  2 spaces ;
?| : link-perform
     ['] perform Is action
     all_links ;
?| : .links
     ['] .id      Is action
     all_links ;
```

Da ist es doch eine Schande, wenn man System-Worte wie WORDS oder VOCS benutzt, die ja auch nicht viel anders funktionieren können. Immer wieder der gleiche Code im System. Kann man denn da gar nichts machen?

Kaum zu glauben: EVALuate

In diesem ersten Entwurf stützt sich das definierende Wort **ACTION:::** auf drei, auch selbst definierende Worte **defines: performs: lists:** , daher auch die drei Doppelpunkte im Namen. Diese Colon's sollen anzeigen, daß ACTION::: drei Namen erwartet und diese den jeweiligen Aktionen *Einhängen*, *Ausführen* und *Anzeigen* zuordnet.

```
: action:::
   ( <add> <perform> <show> --)
      defines:  last @ ( nfa)
      name> listcfa !
      performs:  lists:      ;
actions::: cold: cold .cold
\           ^     ^     ^
\           |     |     das ist der
\           |     |     Anzeiger
\           |     das ist der
\           |     Ausführer
\          das ist der Einhänger
actions::: test:  test .test

test: vocs   test: cr   test: order
test: cr
```

Es ist aber noch *nicht* möglich, so etwas zu schreiben, weil der Stack nicht neutral gehalten wird:

```
test: key  test: emit
```

Dieses kleine Manko können wir Ihnen noch zumuten, aber daß Sie sich jedesmal drei ähnliche Namen ausdenken und diese vor allem noch selbst hinschreiben müssen - das kann so nicht bleiben!! Ein Blick über den Gartenzaun der Programmiersprachen zeigt sofort, daß - nein, nicht TurboPASCAL - sondern LISP die Lösung schon kennt. Mit dem Wort EVAL ist es möglich, daß Sie dem Wort **ACTION:** nur noch *einen* Namen übergeben, direkt aber alle *drei* für die Liste notwendigen Namen ins FORTH-Wörterbuch eingetragen werden. EVAL wertet einen übergebenen String aus und sichert vorsichtshalber den Texteingabepuffer TIB auf dem Returnstack. So können wir uns im Zwischenspeicher PAD die entsprechenden Varianten eines Namens zusammenbasteln und EVAL führt dann die definierenden Worte mit diesen Namen aus.

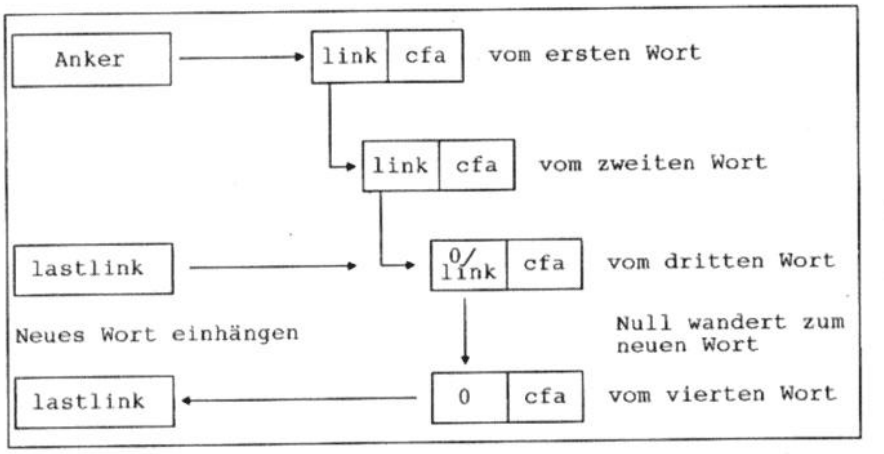


Bild 1: Versuch einer graphischen Darstellung

```
\ definer lister performer
?| : definer     " defines: "
                 ">pad  listname>pad
                 Ascii :  pad append
                 pad eval ;
?| : lister      " lists: "
                 ">pad  Ascii
                 pad append
                 listname>pad
                 pad eval ;
?| : performer   " performs: "
                 ">pad  listname>pad
                 pad eval ;
```

Die Worte in den Gänsefüßchen " " dürfen *nicht* headerless sein und EVAL ist mit *Multipush/Multipull* auch nicht gerade ein Muster an Transparenz, aber das soll das große Erstaunen nicht mindern:

```
\ Das große Staunen:  ACTION:
: Action:      ( <string> --)
     here listname !
     bl word c@ 1+ allot
       \ allgemeiner Name ab HERE
     definer
     last @ name> listcfa !
     performer
     lister  ;

Action: wow
```

Sehen Sie selbst: Im Dictionary stehen drei neue Wörte zu Ihrer Verfügung - Sie können **wow** sofort eine Aufgabe zuweisen und sich mit **.wow** den Listeninhalt ansehen:

```
wow: words  wow: cr
```

Dieser erste Ansatz hat seinen Reiz in der Zugriffssicherheit und der Geschlossenheit der Lösung, hat aber den Nachteil, daß sich das definierende Wort zwei Zeiger merken muß. Wenn Sie die Listenstruktur genauer betrachten (siehe Bild 1), sehen Sie, daß die Liste durch zwei Werte beschrieben wird - den Anfang ANKER und das Ende LASTLINK. Der Anker beim ersten eingehängten Wort gesetzt und später nicht mehr verändert, d.h. neue Worte werden hinten angehängt (siehe Bild 1).

Wie legt man Worte zum Testen an?

In der Testphase braucht man immer einige Worte, die nichts machen. Auch ein schlichtes **bell** ist hilfreich - aber hier reicht schon eine Bildschirmausgabe, damit man 'was sieht. Solche dummy actions legt ein Hobbyist mit Hilfe des volksFORTH-Editors über F5/F7 an und ändert halt die Namen und die auszugebenden Meldungen:

```
: eins        ." Action 1   "  ?cr ;
: zwei        ." Action 2   "  ?cr ;
: drei        ." Action 3   "  ?cr ;
```

Jemand mit FORTH-Erfahrung langt da doch ganz anders zu:

```
: cw:
  Create ( n -- n+1 ) dup c, 1+
  Does> ( -- )
    c@
    ."  Dies ist Wort Nr. "
    . ?cr  ;

0  cw: w1  cw: w2  cw: w3  ( count )
\
\ COUNT liegt auf dem Stack
\
              Create  ctable ( count )
              c,
               ] w1 w2 w3 [
```

Und da sich in FORTH die Parameter so schön von Wort zu Wort durchschlängeln, läßt sich direkt eine Tabelle mit diesen Prozeduren anlegen, die wie ein String ihre Größe im ersten Byte enthält. Zum Vergleich nochmal die Tabelle ohne Größenangabe.

```
Create: table  w1 w2 w3 ;
```

Quelle(n)

[1] Für die, die ihr :DOES> verlegt haben:

```
\ :Does>  für  Create
\  <name> :Does> <action> ;
| : (does>
   here >r [compile] Does> ;
: :Does>
   last @ 0= Abort" without reference"
   (does> current @ context !
   hide 0 ] ;
```

[2] c't 5/90; S.278 ff., Listen vererben

[3] 'Vierte Dimension'; T. Rayburn, METHODS>

[4] Zeitschrift: Computer Language; G.W. Shaw, FORTH shifts gears

[5] "Die Programmiersprache" FORTH; R. Zech, S.15

[6] "Object-oriented Forth"; D. Pountain

Weiterführende Informationen:

✗ S. Lipschutz, Datenstrukturen; S.73 ff. (Datenfelder)
S. Lipschutz, Datenstrukturen; S.123 ff. (Verkettete Listen)

Autoren

✉ Jörg Plewe,
Großenbaumer Str.27
4330 Mülheim/Ruhr,
☎ 0208-423514

✉ Frank Stüss,
An der Turnhalle 6
6369 Schöneck 2,
☎ 06187-91503

✉ Jörg Staben,
Hagelkreuzstr.23
4010 Hilden,
☎ 02103-55609

Fortsetzung folgt in der nächsten 'Vierten Dimension'

```
\ headerless words                                          jps 12jul90

  Variable headers

  headers off

  : ?| ( -- )  headers @ not IF | THEN ;


  : cell+  2+ ;                    \ Transparenz


\ code sharing beim Ausführen und Anzeigen                  jps 12jul90

?| Defer action

?|      : all_links        ( firstlink --)
                          BEGIN  ?dup WHILE
                                 dup cell+  action
                          @ REPEAT ;

?|      : .id  ( cfa)      @ >name .name  2 spaces ;

?|      : link-perform     ['] perform Is action  all_links ;
?|      : .links           ['] .id     Is action  all_links ;


\ listcfa  performs:  lists:                                jps 12jul90

?| Variable listcfa

?| : name>list   listcfa @ >body ,  ;

   : performs:
                Create  name>list
                Does>   @ @ link-perform  ;

   : lists:
                Create  name>list
                Does>   @ @ .links  ;


\ defines:   action:::                                      jps 12jul90

  : defines:
      Create  0 ,  0 ,
      Does> dup >r
            @ 0= IF   here dup  r@ !  r> cell+ !
                 ELSE r> cell+ dup @ here swap !
                                     here swap !
                 THEN  0 ,  ' ,  ;

  : action:::   ( <add> <perform> <show> --)
      defines:  last @ ( nfa) name> listcfa !
      performs:  lists:
  ;


\ Validierung I                                             jps 12jul90

  action::: kalt: kalt .kalt

  kalt: words  kalt: cr
  kalt: vocs   kalt: cr
  kalt: order  kalt: cr
  kalt  ( ruft words cr      )
        (      vocs cr       )
        (     order cr auf   )
 .kalt  ( zeigt Listeninhalt )
```

```
\ EVALuate  ( hat sich bei PUSH einiges ausgeborgt)  jps 12jul90

?| Create: multipull
          r> ( adr)  r> ( len)   rp@ 2dup + rp!
          -rot cmove ;

?| : multipush  ( adr len --)   \ die oben in MULTIPULL stehen!
         even                   \ auf'm PC nicht nötig
         r> -rot 2dup rp@ over - dup rp!    \ Platz auf R-Stack
         swap cmove   >r ( len)  >r ( adr)  \ String dahin
         multipull  >r >r ;     \ und wieder weg!

: eval   ( <string> -- ??? )    #tib push  >in push  blk push
         tib #tib @ multipush   >in off  blk off
         count dup >r  tib swap cmove  r> #tib !
         interpret  ;


\ string Bastelei                                           jrg 12jul90

?| Variable listname

\ : append   ( char string --)
\    under   count + c!   dup c@ 1+ swap c! ;
\ : attach   ( addr len  addr.to --)
\     >r under   r@ count + swap move   r@ c@ + r> c! ;

?| : listname>pad
       listname @ count  pad attach ;

?| : ">pad    ( string --)
       dup c@ 1+  pad swap cmove  ;


\ definer  lister  performer                                jps 12jul90

?| : definer    " defines: "
                ">pad  listname>pad  Ascii :  pad append
                pad eval ;

?| : lister     " lists: "
                ">pad  Ascii . pad append  listname>pad
                pad eval ;

?| : performer  " performs: "
                ">pad  listname>pad
                pad eval ;

\ Die Worte in den Quotes " " dürfen NICHT headerless sein !


\ Das große Staunen:  ACTION:                               jps 12jul90

  : Action:     ( <string> --)
      here listname !
      bl word c@ 1+ allot      \ allgemeiner Name ab HERE
      definer
      last @ name> listcfa !
      performer
      lister
  ;


  clear ( headers)

\ Validierung II

  Action: warm
```

Quelltext 1

FIT

FORTH Interactive Tutor

Konzept einer computergestützten Lernhilfe für FORTH

von Andreas Findewirth

Einleitung

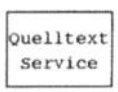
Ostwestfalen gehört ohne Zweifel zur FORTH-Diaspora. Zwischen Bielefeld, Gütersloh, Detmold, Herford und Minden sind die Freunde dieser Sprache dünn gesät. Eine FORTH-Regionalgruppe kam aus den Gründungsschwierigkeiten bisher nicht heraus, von FORTH-Kursen etwa gar an der Volkshochschule kann man nur träumen. In einer solchen Situation stellt sich die Frage nach methodischen Lehr- und Lernhilfen für FORTH mit besonderer Intensität.

Wer FORTH lernen will, muß dies also nur zu oft der Not gehorchend allein im stillen Kämmerlein zu Hause tun. Brauchbare Lehrbücher gibt es ja durchaus [1,2,3,4] und preiswerte Public-Domain-FORTH-Systeme wie das volksFORTH der FORTH-Gesellschaft halten die finanzielle Belastung gering. Aber oft vermißt man beim Selbststudium ein brauchbares Feedback; das schlichte `HAEH?` des volksFORTH wirkt da zunächst eher etwas demotivierend und wenig aussagekräftig. Oder noch schlimmer, der Rechner verschwindet wortlos im FORTH-Nirwana und es bleibt nur noch der Reset-Knopf als Ausweg.

Wie soll also ein geeignetes *Anfänger-FORTH*-Lernsystem aussehen?

- *Robustheit* - Typische Fehler müssen aufgefangen und mit sinnvollen Fehlermeldungen kommentiert werden. Auch exotische Fehler sollten nicht gleich zum Totalabsturz führen.

- *Selbsterklärungsfähigkeit* - Das System soll prinzipiell ohne Handbuch und weiterführende Literatur zu handhaben sein. (Die Verbreitung einer Lernhilfe mit dieser Eigenschaft über PD dürfte den Bekanntheitsgrad von FORTH deutlich verbessern.)



- *Adaptives Benutzerinterface* - FIT soll sich den individuellen Vorkenntnissen und den Lernfortschritten des Benutzers anpassen.

Im folgenden möchte ich versuchen, Struktur und Funktionen eines solchen FORTH-Lernsystems zu skizzieren (draft proposal!). Ich verstehe dabei meine Überlegungen lediglich als Diskussionsgrundlage; in die tatsächliche Realisierung eines FIT sollte selbstverständlich der gesammelte Sachverstand möglichst vieler FORTH-Kenner einfließen.

Stichworte

- ✗ Computer-gestütztes Lernen
- ✗ FORTH-Übungs-System
- ✗ Systemtransparenz
- ✗ Systemrobustheit
- ✗ Adaptive Benutzerschnittstelle

Komponenten

"Das Ziel [des computergestützten Trainings] besteht vielmehr darin, komplexe Zusammenhänge einfach, verständlich und beliebig oft darstellen zu können und bei Bedarf zu wiederholen. Die Vermittlung von Lerninhalten soll Spaß machen; Lerntempo und Wiederholung einzelner Lektionen bleiben dem Schüler anheim gestellt. [...] Kann ein Schüler bestimmte Fragen oder Aufgaben nicht beantworten, leitet ihn das Programm auf alternative Lernwege. Das heißt, derselbe Sachverhalt wird in unterschiedlicher Weise noch einmal erklärt. Auf diese Weise paßt sich das Lernprogramm dem Wissensstand des Lernenden an." Zitat aus [5]

Ein Lernprogramm im o.g. Sinne möchte ich nur als eine von zwei Komponenten eines FIT sehen. Ich werde diesen Teil, der den Lernstoff präsentiert und durch Fragen etc. den Erfolg der Wissensvermittlung prüft, als den Präsentationsteil bezeichnen, kurz FIT/P.

Zum Vermitteln einer Programmiersprache, insbesondere einer Dialogsprache wie FORTH, gehört natürlich das praktische Ausprobieren des Erlernten. D.h., es wird ein unter didaktischen und methodischen Gesichtspunkten aufgebautes, einfaches, aber leistungsfähiges und robustes FORTH-System für Übungszwecke (learning-by-doing) benötigt, im folgenden kurz als FIT/L bezeichnet.

Um Mißverständnissen vorzubeugen, sei hier noch einmal betont: Es geht nicht darum, mit einem FIT-Programm FORTH-Kurse und die Bemühungen von Lehrerinnen oder Lehrern überflüssig zu machen. Kein Dialog mit Maschinen kann zur Zeit (und womöglich wird er es nie können) die Erfahrung des Lernens im zwischenmenschlichen Kontext ersetzen. Mit FIT soll vielmehr dem Lernwilligen beim Fehlen sonstiger Angebote die Möglichkeit geboten werden, autonom und dialogorientiert Kenntnisse über FORTH zu erwerben und dabei interaktiv, individuell und effizient zu lernen.

FIT/P - Die Präsentation

Die Frage nach der Stoffgliederung stellt sich bereits dem Autor eines Lehrbuchs. Die Funktionsweise des Stack, Worte zur Durchführung von arithmetischen Berechnungen oder logischen Vergleichen, die Anwendung von Definitionsworten, die Speicheraufteilung usw. lassen sich jeweils einzeln und für sich recht gut erklären. Aber in welcher Reihenfolge soll der Stoff präsentiert werden, um das Verständnis des FORTH-Systems insgesamt - wie jedes nicht-triviale System ist es mehr als die *Summe* seiner Teile - zu ermöglichen? (Nur wenige Bücher widmen sich so ausführlich der *Philosophie* von FORTH wie Brodie's *Thinking FORTH*, aber Ansätze vermitteln fast alle.)

Nun, sicherlich das wichtigste Kriterium für die Stoffgliederung sind die Vorkenntnisse des Lernenden, sein methodisches Rüstzeug und seine Lernabsichten. Ein *absolute beginner* muß sicherlich ganz anders an sein FORTH-System und seinen Rechner herangeführt werden als beispielsweise ein Student der Informatik mit Grundkenntnissen in Datenstrukturen, der FORTH zu Vergleichszwecken gegenüber C oder Pascal kennenlernen will.

Lehrbuchautoren können diese Fälle nur insoweit berücksichtigen, indem sie im Vorwort oder der Einleitung Hinweise zum Durcharbeiten des Materials geben. Für Lernprogramme stellt sich die Lage anders dar. Die einzelnen *Häppchen* Lernstoff - im folgenden als Präsentationselement, kurz PE, bezeichnet - können in verschiedenen

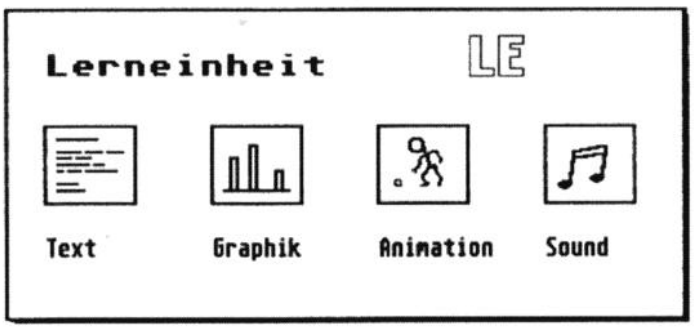


Bild 1: PE - Präsentationseinheit

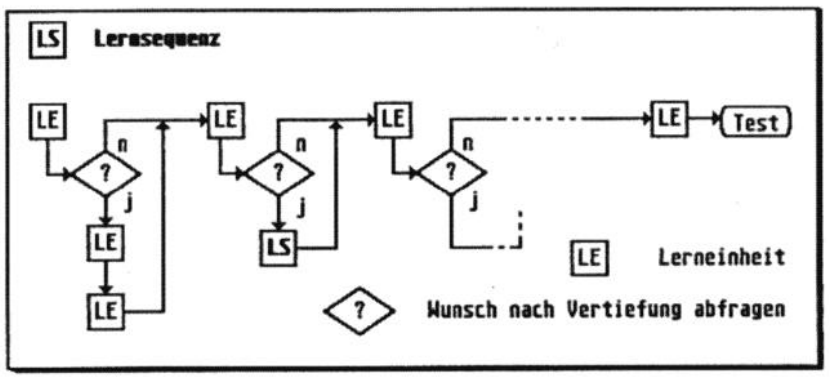


Bild 2: PS - Präsentationssequenz

Gliederungshierarchien dargestellt werden. Die Abfolge der Lernschritte - im folgenden als Präsentationssequenzen, kurz PS, bezeichnet - wird dem individuellen Fortschritt des Lernenden und seines Bedürfnissen dynamisch angepaßt. Innerhalb einer Präsentationssequenz besteht die Möglichkeit, jeweils auf Wunsch den Stoff zu vertiefen und weitere PE's oder PS's anzufordern. In der Regel schließt jede PS mit einem Test bzw. einer Übung ab.

Einzelheiten können den Abbildungen 1 (PE), 2 (PS) und 3 (Test/Übung) entnommen werden. Dabei scheint es mir nicht erforderlich, zur Realisierung auf ein spezielles, externes Autorensystem zurückzugreifen. Die Erstellung von FIT/P in FORTH selbst als Anwendungsprogramm bietet eine Reihe von Vorzügen: bessere Portierbarkeit (das Lernsystem folgt der Sprache, die es vermittelt) und eine gute Transparenz auch des Lernsystems für den fortge-

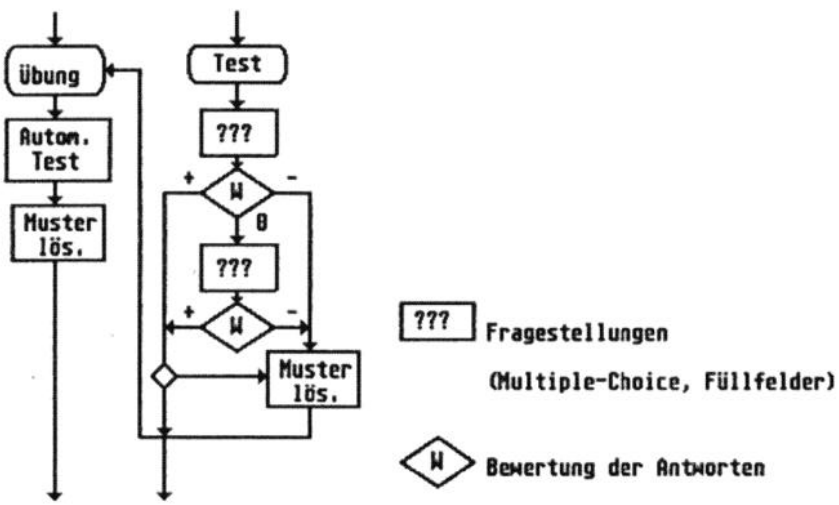


Bild 3: Test/Übung

schrittenen Benutzer (Wandlung vom Tutor zur Fundgrube, Tools etc.). Vielleicht könnte hier auch das gelegentlich als unbefriedigend empfundene Screen-Konzept bei der Text-Präsentation von Lernstoff noch einmal zu Ehren kommen.

Üblicherweise kann nicht der ganze Stoff eines Lernprogramms in einer einzigen Sitzung am Computer bewältigt werden. Wie sollen diese Sitzungen strukturiert sein ? Bei der ersten Sitzung muß die Grundstruktur der Stoffpräsentation erstmals erstellt werden. Das Lernprogramm befragt also den *Schüler* über seine Kenntnisse (Selbsteinschätzung) und Interessen (Lernabsichten). Ein Einstiegstest präzisiert die Selbsteinschätzung und liefert die Daten, um die Abfolge der Lernsequenzen festzulegen. Jede Folgesitzung beginnt mit einem kleinen Einstiegstest, der die Präsentationsfolge des Stoffes aktualisiert. Dann folgt in Abhängigkeit vom Einstiegstest eine Stoffwiederholung, aus der sich heraus die Vermittlung von neuem Lernstoff entwickelt. Nach einer bestimmten Zeit oder auf Wunsch des Lernenden wird die Sitzung beendet und mit einem kleinen Test sowie Hinweisen für das Selbststudium beendet.

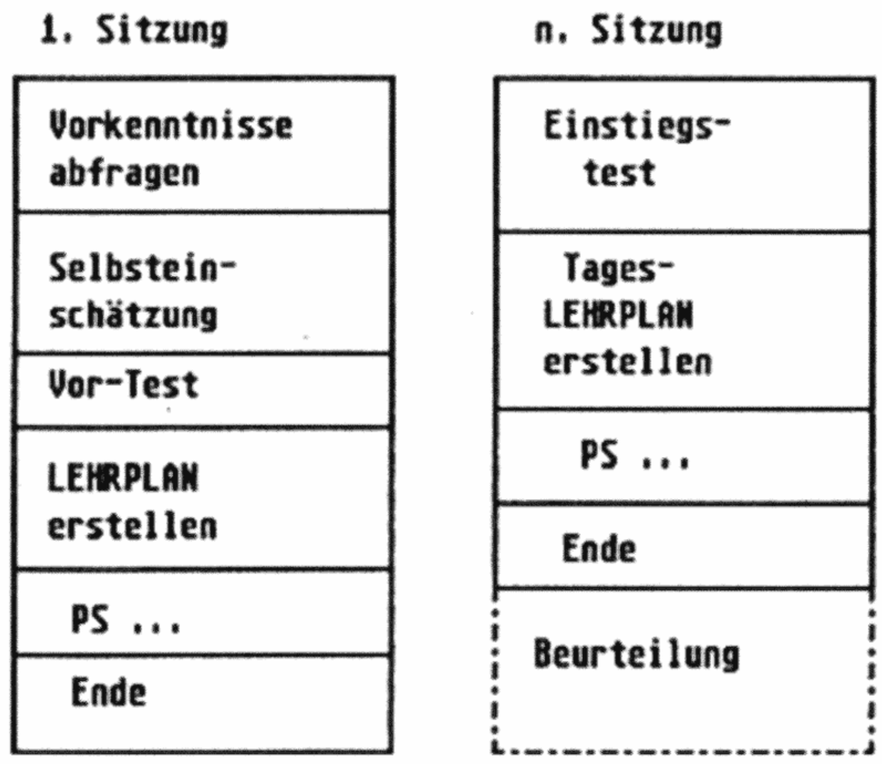


Bild 4: Struktur von Sitzungen

Die Testergebnisse beeinflussen wiederum den Inhalt der nächsten Sitzung (siehe Bild 4).

Ein Lernprogramm soll aber nicht nur gelenktes Lernen (wie zuvor konzipiert) ermöglichen, sondern auch selbstbestimmtes Lernen. Es muß beispielsweise für den *Schüler* die Möglichkeit bestehen, an jeder Stelle vertiefende Informationen anzufordern oder Querverweisen zu folgen (Hypertext-Konzept, vgl. auch LEIBNIZ in [6]).

Auch soll das System auskunftsfähig sein, also zum Beispiel die Lernziele angeben können oder auf Wunsch anzeigen, wie die Testergebnisse die Strukturierung des dargebotenen Lernstoffes beeinflußt haben. Auch die Tests bzw. Übungen sind ambivalent zu sehen. Zum einen beeinflussen die Ergebnisse die Stoffpräsentation und sind damit Bestandteil des gelenkten Lernens. Zum anderen vermitteln sie dem *Schüler* Erfolgs- und Mißerfolgserlebnisse. Wer an sich selbst Wissenslükken entdeckt, wird diese schließen wollen (selbstbestimmtes Lernen) und gezielt vertiefenden Stoff anfordern.

Forderung	**Autor und Fundstelle in VD**
Leichte Stringverwaltung Funktionen zur Handhabung von Dateien Floating-Point-Arithmetik Zuladbare mathematische Funktionen	Wolfgang Mües, 3/87 S.15
Objektorientiertheit	R.Jones (D.Luda), 2+3/88
Ein-/Ausgabe großer Integer-Zahlen Schnelle mathematische Funktionen (ASM)	4/88, S.47
Kontextsensitive Hilfen in Forth	Jörg Staben, 4/89 S.5
Eingabeprozedur für Zahlen Mathematische Funktionen - Gleitkommaoperatoren - Winkelfunktionen Maskenhandhabung, Fenstertechnik	Josef Rieger, 4/89 S.6
Wortwahl, Eindeutschung Einfache Grafik- u. Soundaufrufe Nicht-screenorientierter Editor Hilfetexte (abschaltbarer) Fehlerabfangroutinen "Luxus"-Versionen von DUMP und .S	M. Holzapfel, 4/89 S.15
Aktivität	
Kleinkram Laden	Finn Berlev, 3/87 S.49
Swopperatoren	R. Kretzschmar, 2+3/88
Stack-Handhabung	Frank Stüss, 2/89 S.23
Neue Datentypen in Forth	K. Scheller, 3/89 S.26
SWORDS	A. Soeder, 4/89 S.7
Streamfile-Interface	Jörg Plewe, 1/90 S.6

Tabelle 1: Übersicht Forderungen und Aktivitäten

FIT/L - Das Übungssystem

In der 'Vierten Dimension' waren schon öfter Beiträge zu lesen, in denen bestimmte Eigenschaften für ein Einsteiger-FORTH gefordert oder vorgestellt wurden, zuletzt vor allem in [7]. Eine Kurzübersicht enthält Tabelle 1.

Ich möchte all diese Anregungen wie folgt systematisieren:

- Systemsicherheit
- Durchsichtigkeit und Selbstbeschreibung
- Erweiterte Hilfe- und Auskunftsfunktionen
- Ausreichendes Sortiment bekannter Datenstrukturen
- Ausreichendes Sortiment von Ein-/Ausgabeverfahren
- Mehr Komfort bei FORTH-Besonderheiten

Fehlerquelle	Gegenmaßnahme
Parameter-Stack wächst ins Dictionary hinein. (z.B. bei nicht ausgewogener Stack-Bilanz in einer Programmschleife).	Regelmäßige Stack-Tests in Programmschleifen mit ?STACK .
Return-Stack wächst in die User-Area hinein. (z.B. wegen zu großer Rekursionstiefe).	Return-Stack-Tests in rekursiven Worten mit ?RSTACK.
Nicht abbrechende Schleifen.	Zeit-/Lauflimit für Schleifen.
"Absturz" nach Manipulation des Return-Stacks mit >R, R>, RDROP usw.	Kontrollierte Verwendung der Return-Stack-Worte mit ?PAIRS erzwingen.
Irrtümliches Überschreiben von Speicherbereichen (z.B. versehentlich falsch BASE 2 ! statt 2 BASE ! Folge: Zerstörung der Kaltstartwerte der User-Area) .	Schreibschutz für ausgewählte Speicherbereiche installieren bei Verwendung von !, C!, 2! Variablen usw. müssen benutzbar bleiben.

Tabelle 2: Übersicht Fehlerquellen und Gegenmaßnahmen

Die Systemsicherheit erscheint mir besonders wichtig; aber auch nur mit entsprechendem Aufwand zu realisieren. Ein *sicheres* FORTH-System sollte im Idealfall

- ✗ Fehlerquellen vorab erkennen und anzeigen,
- ✗ zur Laufzeit Fehlersituationen auffangen, eindeutig benennen und kontrolliert beenden,
- ✗ nach Abstürzen in *exotischen* Situation wenigstens Information über die Fehlerursache liefern.

Praktisch ist es natürlich bei einem so offenen System wie FORTH nicht machbar, wirklich alle möglichen Fehlersituationen beim Entwurf zu analysieren und entsprechende Sicherheits-Checks einzubauen. Andererseits sind die FORTH-Neulinge in der Regel keine *Verbrecher*, deren Ziel es ist, mit allen Tricks die Sicherheits-Checks zu überwinden.

Ich halte folgende Vorgehensweise für praktikabel: Vorschläge für Sicherheits-Checks werden in der 'Vierten Dimension' publiziert und dann von den Lesern auf Herz und Nieren geprüft. Was ihre Kritik und ihren Einfallsreichtum übersteht, ist brauchbar.

Tabelle 2 enthält eine Übersicht wichtiger Fehlerquellen und möglicher Gegenmaßnahmen, vgl. auch den beigefügten Quelltext.

Durchsichtigkeit und Selbstbeschreibungsfähigkeit ist FORTH bereits von Hause aus mehr zu eigen als anderen Programmiersprachen. Da sind z.B. Worte wie `WORDS`, `ORDER`, `DUMP`, `VIEW` oder Tools wie der Disassembler, Decompiler und Tracer im volksFORTH, die Einblicke ins System gestatten. Weitere Überlegungen in dieser Richtung finden sich u.a. in Anhang von [3] und im 6. Kapitel von [4].

Ein *Anfängersystem* sollte mit Worten, die einfach zu handhaben sind und einen intensiven Einblick in das Innere von FORTH gestatten, *verschwenderisch* ausgestattet sein. Wer mit FORTH beginnt, sollte nicht gleich vor der Wahl stehen, auf solche Hilfen zu verzichten oder sie selbst programmieren zu müssen. Das Gebiet ist derart reichhaltig und umfangreich, daß man trotzdem daraus noch Übungsaufgaben entwickeln kann. Dabei können die fer-

Konstruktionsmuster	Programmanweisung	Daten-Typ	Operatoren	Hinweis
atomares Element	Zuweisung	skalarer Typ - enumeration - integer - cardinal - real/float - (complex) ,dto. - boolean - character - set/bag	DEC, INC ,... + - * / ... dto. dto. and or not ... Element von	[1] [2]
Aufzählung	zusammengesetzte Anweisung	record/struct		[1]
Wiederholung um bekannte Anzahl	DO-LOOP-Anweisung	array/row (string)	concat ...	[3]
Auswahl	IF-ELSE-THEN CASE-Konstrukt	varianter record union		
Wiederholung um unbekannte Anzahl	BEGIN, WHILE, UNTIL, REPEAT	sequence, stream/file	open, append ...	
Rekursion	Prozedur (Wort)	rekursiver Typ - lineare Liste - Bäume		[4] [5]
allgem. Graph	GOTO-Anweisung	"verpointerte" Struktur		

ANMERKUNGEN:

[1] Vgl. VD 3/89 S.26ff
[2] Vgl. VD 1/89 S.13ff
[3] Vgl. VD 4/89 S.24ff
[4] Typische Datenstruktur in FORTH
[5] Sehr nützlich zum Suchen und Sortieren

Neben der Einteilung nach Struktur/Typ/Art der Daten ist für die Handhabung in der Praxis die **Speicherklasse** von großer Bedeutung. In C spricht man z.B. von *static* und *automatic*, was etwa der Einteilung in globale und lokale Daten von Prozeduren entspricht. Vgl. hierzu auch VD 1/88 S.25ff.

Tabelle 3: Datenstrukturen, Elementaroperationen und Kontrollstrukturen

tig vorgegebenen Auskunftsworte zusätzlich als Anschauungsmaterial wertvolle Dienste leisten.

Erweiterte Hilfe- und Auskunftsfunktionen sind gewissermaßen der Import von Dienstleistungen aus dem FIT/P in das FIT/L. Syntax und Grundprinzip von FORTH sind sehr einfach gehalten, ein effektives Arbeiten erfordert jedoch die Kenntnis des recht umfangreichen Wortschatzes. (Ähnlich: einfache Grundstrukturen in C, umfangreiche Funktionsbibliotheken; einfaches Botschaftskonzept in Smalltalk, umfangreiche Klassen- und Methodenbestände). Der FORTH-Wortschatz muß also methodisch und systematisch erschlossen werden (Wort-Datenbank), so daß gezielt nach Worten gesucht werden kann, deren Namen nicht bekannt ist, über deren Funktionszweck jedoch Vorstellungen bestehen. Dies unterstützt auch die Wiederverwendbarkeit einmal entwickelten Codes (Worte) und damit auch die Kompaktheit von FORTH.

Ein ausreichendes Sortiment bekannter Datenstrukturen erleichtert den Umgang mit FORTH außerordentlich. Ich will an dieser Stelle die althergebrachte Diskussion um FORTH und Floating-Point-Zahlen nicht unnötig neu beleben. Bekannt ist, daß es ohne Gleitkommazahlen geht, in [3] z.B. wird der Leser noch im ersten Kapitel mit Zweier-Komplement-Zahlen und Skalierungsbefehlen für die Integer-Arithmetik konfrontiert. Mir erscheint es jedoch sinnvoller, für *absolute beginners* die arithmetischen Interna zunächst zu verstecken. Es soll gerechnet werden können, auch mit Kommazahlen, frei nach der Devise *Bring es zum Laufen.* In einem zweiten Schritt kann man dann die Zahlendarstellung im Rechner und eine Analyse des Rechenaufwandes einbeziehen.

Es sollen also im Lernsystem alle gängigen Datenstrukturen mit ihren Elementaroperationen verfügbar sein. Im Laufe des Lernprozesses ist immer noch Gelegenheit für den *Schüler*, selbst solche Datenstrukturen oder Modifikationen davon im Rahmen von Übungsaufgaben selbst zu implementieren. Auch hier können dann die fertigen Lösungen als Musterbeispiele herangezogen werden. Eine Übersicht enthält Tabelle 3, die sich an [8] orientiert.

Für Ein- und Ausgabeprozeduren gilt sinngemäß das zu den Datenstrukturen Gesagte. Einfache Befehle wie `INPUT` oder `PRINT` in Basic gestatten die rasche Entwicklung kleiner interaktiver Programme und fördern dadurch den Lernprozess. Ähnliches gilt für Grafikbefehle wie beispielsweise in Logo.

Mehr Komfort bei FORTH-Besonderheiten wie der Stack-Handhabung ist wünschenswert, steht die Sprache doch im Ruf, etwas für Bit-Pfriemler oder Technik-Freaks mit selbstgelötetem Mikrocomputer zu sein. (In letzter Zeit scheint mir FORTH allerdings etwas *salonfähiger* geworden zu sein.) Aber zugegeben: *Noisy words* im Quelltext, all die `DUP`'s, `SWAP`'s, `DROP`'s, `ROT`'s usw., wirken auf den ersten unvoreingenommenen Blick doch reichlich seltsam. *Produktiv* sind sie auch nicht, jonglieren sie doch nur die Parameter im Speicher ein bißchen hin und her. Ist das also der Preis, den man für das Parameter-Passing mit unbenannten Parametern bezahlen muß ? (Oh Pascal, du hast es besser !?!)

Zu diesem Thema gibt es schon länger Überlegungen, ich nenne hier nur [9]; aber der Stein der Weisen scheint noch nicht gefunden.

Ausblick

Wahrscheinlich ist es gar nicht möglich, *das* Lernsystem für Anfänger zu konzipieren, welches allen Anforderungen gleichermaßen genügt. Dafür ist vielleicht die Zielgruppe nicht homogen genug oder FORTH viel zu flexibel. Dennoch möchte ich die Idee eines interaktiven FORTH-Tutors (FIT) weiter verfolgen und fortentwickeln; das Arbeitsfeld habe ich in diesem Beitrag abzustecken versucht. Es ist sehr umfangreich und ich bin daher für jede Hilfe dankbar und für konstruktive Kritik empfänglich.

Noch eine kleine Schlußbemerkung: Einem künftigen Lernsystem würde ich ganz bestimmt das unangenehme `HA-EH?` austreiben wollen, obwohl man diese Eigentümlichkeit des volks-FORTH immerhin als gelungene Eindeutschung einer FORTH-Systemmeldung ansehen kann. Wenn ein höfliches FORTH-System mit einer Eingabe nichts anfangen kann, sollte es sich z.B. mit `PARDON?` melden. Das kommt zwar aus dem Französischen, ist aber auch in Deutschland bekannt und im englischen Sprachraum ebenfalls gebräuchlich (pardon: apologize, e g for not hearing or understanding what sb says).

```
hallo PARDON?
bye OK
```

Quellenangaben

[1] Programmieren in FORTH - Vom Einstieg bis zum Standard (org.: Starting FORTH), Leo Brodie, Hanser München-Wien/Prentice-Hall London 1984

[2] FORTH - Ein Programmiersystem ohne Grenzen, Andreas Goppold/Roger Bouteiller, Edition Aragon 1985

[3] FORTH 83 - Eine gründliche Einführung in die FORTH-Version, Ronald Zech, Franzis München 1987

[4] Einführung in FORTH 83 (volks-FORTH), Manfred Mader, Heim-Verlag Darmstadt 1989

[5] Pauken per PC - Computer-based-Training CBT, Bernward Bodenstedt, Chip 6/90 S. 320ff, Vogel-Verlag Würzburg

[6] Software-Engineering auf Personal Workstations, Andreas Goppold, VD Vol.IV, 2/3 Sept. 1988 S. 27ff

[7] Das "Greenhorn-Module ", Martin Holzapfel, VD Vol.V, 4 Dez. 1989 S.15f

[8] Algorithmen und Datenstrukturen, Niklaus Wirth, Teubner Stuttgart 1983

[9] Swopperatoren - Stackoperatoren mit System, Rolf Kretzschmar, VD Vol.IV, 2/3 Sept. 1988 S. 10ff

✉ Andreas Findewirth
Im Großen Vorwerk 48
4900 Herford

FORTH Interactice Tutor

Screen # 0

```
\\                          SECURITY.SCR                    21jun90fi

      Quelltexte zu einem Beitrag in der Vierten Dimension

                              F I T
                              ---
                        Forth Interactive Tutor
     - Konzept einer computergestützten Lernhilfe für Forth -


                        Andreas   Findewirth
                        Im Großen Vorwerk 48
                        4900   H E R F O R D

                        Telefon  05221/23504
```

Screen # 1

```
\ Loadscreen ...                                            21jun90fi

page  0 list cr

Onlyforth                       here


1 4  +thru


here  ' memlimit 2+  !          clear

                                here swap -  cr
+safety                         40 u.r  .(  Bytes allokiert.) cr

     &7 list cr
```

Screen # 2

```
\ Schreibschutz für Speicher                                19jun90fi

here | Constant memlimit

| : var?   ( adr - flag )  2- @  [ ' blk @ ] Literal  = ;
| : user?  ( adr - flag )  up@ udp uwithin ;
| : mem?   ( adr - flag )  memlimit u< ;

| : ?protect  ( adr - adr )
     dup mem? IF dup var? not abort" Keine Variable!" exit THEN
     dup user? not abort" Keine User-Variable!" ;

| : (!   ( w  adr - )  ?protect ! ;
| : (c!  ( 8b adr - )  ?protect c! ;
| : (2!  ( d  adr - )  ?protect 2! ;
```

Screen # 3

```
\ Hilfsworte f. Kontrolle Returnstack u. Schleifen          19jun90fi

: ?rstack ( - )  rp@ udp -  $40 <  abort" Zu viele Aufrufe!" ;

| : 4?  ( w - )  4 ?pairs ;


| 2Variable time0

Variable timelimit              &12000 timelimit !   \ = 60 sec

| : #)  ( - )  ?stack  $4BA. 12@  time0 2@  d- dabs drop
     timelimit @  u> abort" Zeitlimit Schleife!" ;

| : (#  ( - )  $4BA. 12@  time0 2! ;
```

Screen # 4

```
\ Ein-/Ausschalter für Sicherheits-Checks                   19jun90fi

| Variable >checks

| Create -sec  | noop noop  noop noop noop noop  ! c! 2! [

| Create +sec  | ?stack ?rstack (# #) 4    4?    (! (c! (2! [

: +safety  ( - )  +sec >checks ! ;
: -safety  ( - )  -sec >checks ! ;            -safety

| : >exec ( n - )  Create c,  Does> c@ >checks @ + @ execute ;

| : Make_exec ( - )    0 &9 bounds  DO  I 2*  >exec  LOOP ;

Make_exec   ?stack ?rstack  ?(# ?#)  >4< >4<?  ! c! 2!
```

Screen # 5

```
\ Einige Worte anpassen                                     20jun90fi

: recursive  ( - )
     compile ?rstack  [compile] recursive ; immediate restrict

: BEGIN    compile ?(#  [compile] BEGIN ;      immediate restrict
: DO       compile ?(#  [compile] DO ;         immediate restrict
: ?DO      compile ?(#  [compile] ?DO ;        immediate restrict
: LOOP     compile ?#)  [compile] LOOP ;       immediate restrict
: +LOOP    compile ?#)  [compile] +LOOP ;      immediate restrict
: UNTIL    compile ?#)  [compile] UNTIL ;      immediate restrict
: REPEAT   compile ?#)  [compile] REPEAT ;     immediate restrict

: >r       compile >r  >4< ;                   immediate restrict
: r>       >4<?  compile r>  ;                 immediate restrict
: rdrop    >4<?  compile rdrop ;               immediate restrict
```

Screen # 6

```
\  "Musterfehler" zum ausprobieren                          20jun90fi

: uhr  ( - )  page  25000 0  DO  1 1 at  I dup .  LOOP ;

| : fak  ( w1 - w2 )  recursive
     ?dup 0=  IF  1  ELSE  dup 1- fak  *  THEN ;

: unlimited  ( - )  -1 fak .s cr ;

: forever  ( - )  page
   0  BEGIN  1 1 at  dup .  1 +  stop? ?exit  REPEAT ;

: peng  ( - )  base $10 ! ;

: murks  ( - )  base @ dup >r dup . r> r> .s ;
```

Screen # 7

```
\ Anmerkungen                                               21jun90fi

Die vorgestellte Worte sind ein erster Versuch, etwas Sicher-
heit gegen fehlerhafte Eingaben zu bieten. Sie decken dabei
in etwa den Bereich ab, der in Tabelle 2 des VD-Artikels
aufgelistet wird.

Screen 6 enthält einige Worte zum Ausprobieren. MURKS wird schon
beim Laden bemeckert, die anderen Worte werden kompiliert und
können aufgerufen werden.

Für den ernsthaften Einsatz müpten einige Passagen erheblich
schneller werden und sollten dann in Assembler codiert werden.
Der Returnstack ist gegen Überlaufen zur Zeit nur bei direkter
Rekursion geschützt, aber nicht gegen indirekte Rekursion z.B.
mit deferred Worten.
```

Screen # 8

```
\\ MEMLIMIT VAR? USER? MEM? ?PROTECT (! (C! (2!             21jun90fi

MEMLIMIT Obergrenze des geschützten Dictionary-Bereichs

VAR? liefert true, wenn adr die pfa einer Variablen ist.
USER? liefert true, wenn adr in der Userarea liegt.
MEM? liefert true, wenn adr im geschützten Bereich liegt.

?PROTECT bricht ab, wenn adr weder auf den Inhalt einer
  Variablen noch einer Uservariablen zeigt.


(!    "Gesicherte" Worte
(C!
(2!
```

Screen # 9

```
\\ ?RSTACK 4? TIME0 TIMELIMIT #) (#                         21jun90fi

?RSTACK bricht ab, wenn auf dem Returnstack der Platz knapp wird

4? bricht ab, wenn w nicht 4 ist (unstructured).


TIME0 speichert Startzeit bei Programmschleife.

TIMELIMIT enthält maximale Zeit für Programmschleife in 5 ms

#) testet am Schleifenende Stack und Zeitlimit, verwendet
   für die Zeitmessung die 200Hz-Atari-Systemvariable.

(# initialisiert die Startzeit am Beginn der Schleife.
```

Screen # 10

```
\\ >CHECKS -SEC +SEC +SAFETY -SAFETY >EXEC MAKE_EXEC        21jun90fi

>CHECKS zeigt auf gültigen Testwortvektor.

-SEC Testwortvektor bei ausgeschalteter Sicherheitsprüfung

+SEC Testwortvektor bei eingeschalteter Sicherheitsprüfung

+SAFETY schaltet die Sicherheitsprüfung ein.
-SAFETY schaltet sie aus.

>EXEC erzeugt ein Wort, das die n/2-te Komponente eines
  Testwortvektors ausführt.

MAKE_EXEC erzeugt alle benötigten Worte in einem Rutsch.
```

Screen # 11

```
\\ Redefinitionen                                           21jun90fi

RECURSIVE wird um Test auf Überlauf des Returns erweitert.


BEGIN, DO, ?DO initialisieren den Zähler für die Zeitlimit-
  Überwachung bei Programmschleifen.

LOOP, +LOOP, UNTIL, REPEAT werden um Test auf Stacküberlauf
  und Zeitlimitüberschreitung erweitert.


>R und R> bzw. RDROP müssen paarweise auftreten und dürfen nicht
  mit Kontrollstrukturen verschränkt werden.
```

RISC-FORTH

von H.-G. Willers, Gartenstr. 11, 8047 Karlsfeld

Stichworte

- ✗ RISC-Prozessor
- ✗ i860
- ✗ FORTH-Implementation

"No Risc, no Fun", diese Devise eines Herstellers von RISC-Computern, läßt sich sinngemäß auf FORTH übertragen:
"No Risc, no FORTH".

Getreu diesem Motto wurde ein FORTH auf einem modernen Risc-Prozessor, dem i860 von Intel implementiert (siehe Info-Kasten i860 [1]). Ein RISC-Prozessor lässt sich grob anhand von 4 Merkmalen charakterisieren:

- ✗ Ausführung einer Instruktion pro Taktzyklus
- ✗ einheitliches Befehlsformat
- ✗ Load/ Store Architektur
- ✗ Aufwandsverlagerung: Compilezeit statt Laufzeit

Die Ausführung von einer Instruktion pro Taktzyklus wird durch den Einsatz von Cache-Speichern und die Einbeziehung einer Instruktions-Pipeline gewährleistet. Während ein Opcode aus dem Speicher (Cache) gelesen wird, wird gleichzeitig der zweite Befehl in der Pipeline dekodiert, zur selben Zeit werden beim dritten Befehl Operanden aus Registern (oder dem Speicher/Cache) geholt, während beim vierten Befehl nun arithmetische Operationen ausgeführt werden u.s.w.

Dieses Idealbild von einer Instruktion pro Takt ist in Realität natürlich nicht immer erfüllbar, und wird durch Störungen des Pipeline-Flußes, Cache-Misses und längere Instruktionsdauer (z.B. bei Multiplikation o.ä) hervorgerufen

Ein einheitliches Befehlsformat erleichtert die Dekodierung von Befehlen ganz außerordentlich, z.B. werden nicht illegale Adressierungsarten eines Maschinenbefehls als Grundlage eines anderen Befehls gewählt (siehe 68000), was den inneren Aufbau eines RISC-Prozessors leichter implementierbar macht.

Load/Store-Architekur bedeutet, daß Speicherzugriffe im allgemeinsten Fall nur Register-Indexed erfolgen, also in der Art von:

```
load   reg1, index(reg2)
```

wobei hier Register1 mit dem Inhalt der Speicherzelle, deren Adresse die Summe aus Index und Register2 ist, geladen wird.

RISC-Prozessoren leben in erster Linie von guten Compilern, welche auf die zugrunde liegende Maschinenarchitektur zugeschnitten sind. Dadurch kann optimal schneller Code erzeugt werden.

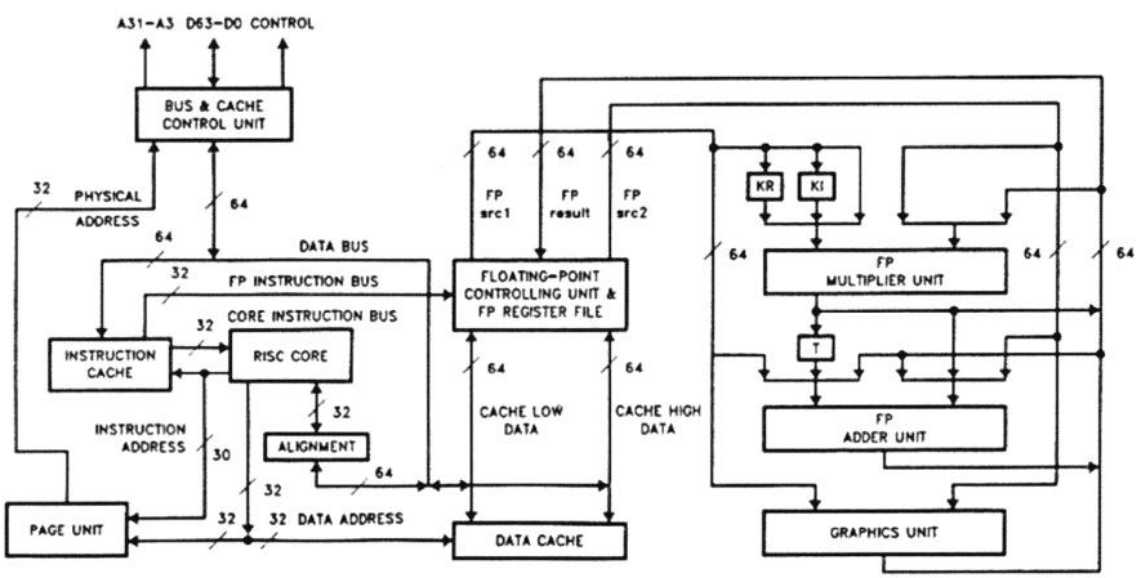


Bild 1: Block Diagramm

Es gibt natürlich Mischformen bei den einzelnen RISC-Prozessoren, sowie zusätzliche Forderungen für einen *echten* RISC-Prozessor (wie z.B. eine Implementierung ohne Mikrocode), aber diese Dinge sind sicher genauso philosophisch zu betrachten, wie die Frage, welcher Mikroprozessor nun wirklich der Beste ist.

Ans Eingemachte

Als Basis für das Projekt diente Allan Pratt's CFORTH. Dies ist ein FORTH, das in der Sprache C geschrieben ist, und unter UNIX lauffähig ist. Ein spezieller Preprozessor erzeugt dabei ein Speicherabbild des Wörterbuches. Der innere Interpreter und die Primitive sind in C geschrieben. CFORTH arbeitet mit Zellen zu 16 Bit und ist nach FIG-Standard geschrieben.

Im Rahmen der Implementierung wurden der Adreßinterpreter und Teile der Primitive in i860-Assemblercode umgeschrieben. Der Preprozessor wurde geändert, sodaß er die Wortheader und die Adressen in einem Format ausgibt das kompatibel mit dem i860 Assembler ist. Ein Lauf des C-Preprozessors faßt alle beteiligten Dateien zusammen, die danach assembliert und zu einem ablauffähigen FORTH gelinkt werden. Alle beteiligten Programme (Preprozessor, C-Compiler, Assembler und Linker) laufen dabei unter einem 386er UNIX ab.

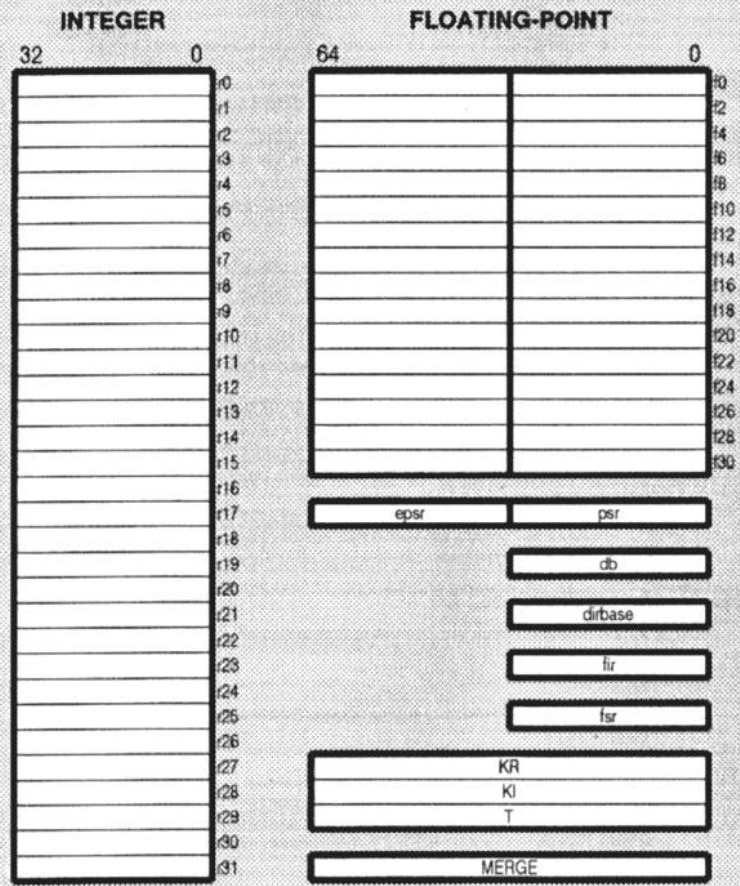
Von der Struktur her ist das i860 FORTH noch ein FIG-FORTH. Ein Großteil der Worte wurde jedoch auf F83 umgeschrieben. Das i860 FORTH ist ein 32-Bit FORTH, das heißt, daß alle Zahlen auf dem Stack mit 32 Bit Breite angelegt werden. Ebenso ist der ganze Speicherbereich des i860 direkt zugänglich.

Das folgende Listing zeigt den inneren Interpreter des i860-FORTH:

```
//
//--        address interpreter for
//     indirect threaded FORTH machine
//
//     all jumps to next are done by
//       bri    ZNEXT
//     to allow for a debugging
//     version of the
//     address interpreter.
//
next::
        ld.l    0(ZIP), ZW
        addu    4, ZIP, ZIP
        ld.l    0(ZW), ZTEMP1
        bri     ZTEMP1
        addu    4, ZW, ZW
                // ZW points to PFA
```

i860 64-Bit Microprocessor

Der i860 besitzt 32 Integer-Register mit jeweils 32 Bit Breite (siehe Abbildung). Darüber hinaus besitzt der i860 32 Gleitkommaregister zu 32 Bit, die sich für doppeltgenaue Zahlen auch als 16 Register mit 64 Bit paaren lassen. Der i860 hat ein Instruktions-Cache von 4 KByte und ein Daten-Cache von 8 KByte.



Info-Kasten i860

Dieses `Next` ist analog zum FIG-FORTH implementiert und wird in 6 Taktzyklen des i860 abgearbeitet. Interessant ist dabei die Inkrementierung von ZW (Code Field Pointer). Sie geschieht im sogenannten 'Branch Delay Slot'. Dabei wird bei einer Programmverzweigung immer noch ein Befehl zusätzlich abgearbeitet, in unserem Falle die Inkrementierung von ZW.

Das Register ZNEXT zeigt direkt auf 'next', so daß mit

```
bri     ZNEXT
```

direkt zum inneren Interpreter gesprungen werden kann. Auf diese Weise kann durch 'Umbiegen' von `ZNEXT` eine Debug-Version einfach erzeugt werden, die z.B. immer einen Ausdruck der Register der FORTH-Maschine auf dem Bildschirm ausgibt.

Doppelpunktdefinitionen werden mit dem Primitiv `DOCOL` eingeleitet:

```
//
//      *************
//      *   DOCOL   *
//      *************
//
_DOCOL::
  st.l ZIP, -4(ZR) // store IP
  mov  ZW, ZIP     // temp
```

```
bri   ZNEXT       // do it again sam
addu -4, ZR, ZR   // dec. return
                  //stack ptr.
```

Der Instruction-Pointer wird dabei auf dem Return-Stack abgespeichert und das nächste FORTH-Wort ausgeführt. Die Ausführungszeit beträgt dabei 4 Taktzyklen.

Am Ende einer Doppelpunktdefintion wird der gerettete Wert des IP wieder hergestellt:

```
//
//    **********
//    *   ;S   *
//    **********
//
p_semis::
      ld.l    0(ZR), ZIP
             // get return
      bri     ZNEXT
      addu    4, ZR, ZR
             // adjust return stack
```

Diese Operation wird in 3 Taktzyklen des i860 ausgeführt.

Die Angabe der Taktzyklen geht immer von der Annahme aus, daß der auszuführende Code im Instruction-Cache steht. Bei einer Größe des I-Cache von 4kB kann davon ausgegangen werden, daß alle in Assembler ausgeführten Programmteile im I-Cache geladen sind.

Auf Grund der Cache-Implementierung für das Daten-Cache (Write-Back Cache) wird sichergestellt, daß Lese- und Schreiboperationen am Daten- und am Return-Stack zu keiner Belastung des externen Buses führen. Leser und Schreiber im Daten-Cache werden in einem Takt abgewickelt.

Benchmark

Durchgeführt wurde der Benchmark, wie er in [2] beschrieben wurde:

```
: nip      swap  drop ;
: bench
  1 100000 0 DO  1  nip  LOOP
  drop ;
```

Die Ergebnisse:

System	Zeit
i860 FORTH indirect threaded i860 33MHz	0.23s
FPC 3.50 20MHz Tandon 386 mit 64k Cache	0.96s
FFORTH ohne Makro [2] ATARI ST, 8MHz	2.7s
FFORTH mit Makro [2] ATARI ST, 8MHz	0.88s
32FORTH [2] ATARI ST, 8MHz	6.0s
volksFORTH [2] ATARI ST, 8MHz	4.7s
4xFORTH [2] ATARI ST, 8MHz	1.2s
FORTHmacs [2] ATARI ST, 8MHz	4.4s

Die Zeiten für den i860 und den 386er wurden vom Autor mit 10^7 Durchläufen bestimmt und auf 100000 Durchläufe rückgerechnet.

Im Vergleich dazu beträgt die Zeit für 100000 leere Loops beim i860 nur 0.05s.

Ausblick

Die für ein Indirect Threaded FORTH schon sehr guten Ausführungszeiten beim i860 lassen sich durch Maßnahmen, wie sie in [2] und [3] beschrieben wurden noch weiter verbessern.

Eine Umstellung auf *Subroutine Threading* bringt sicher einen Faktor 2 in der Rechenleistung. Dabei wird der innere Interpreter des FORTH-Systems durch eine Folge von (z.B. für den 68000)

```
jsr       dies
jsr       das
jsr       jenes
jsr       ....
```

Instruktionen ersetzt, was einen wesentlich geringeren Overhead an Rechenzeit bedingt.

Macro-Inlining (direktes Einsetzen des auszuführenden Codes anstelle eines '`jsr ...`') erhöht zwar die Größe des übersetzten Programms, bringt jedoch auch hier bei kurzen Worten (`SWAP, DROP, DUP` ...) noch einmal ca. einen Faktor 2 an Rechenleistung.

Eine 'richtige' F83-Implementierung mit einem Metacompiler als nächste Stufe soll diese Geschwindigkeitssteigerung nachweisen.

Quellen

[1] Dokumentation zum i860 der Firma Intel

[2] J. Plewe: "Schnelles FORTH für den MC68000", Vierte Dimension, Vol. VI, Nr. 1

[3] L. Chavez: "A Fast FORTH for the 68000", Dr. Dobb's Journal, October 1987 (beschreibt die Implementierung von Mach2 für den Macintosh)

✉ H.-G. Willers
Gartenstr. 11
8047 Karlsfeld

BlocksWorld

Ein hierarchisches Filesystem in FORTH

von Alexander Burger

Quelltext Service

Das Schöne an FORTH ist, daß man vom System her direkt dazu ermuntert wird, alles nach eigenen Wünschen und Vorstellungen zu gestalten.

1986 bekam ich die Laxen/Perry Implementation [1] von FORTH 83 in die Finger. Ich hatte schon immer davon geträumt, mich näher mit der 6809-CPU zu beschäftigen, und beschloß daher, ein Stand-Alone-System mit dieser CPU und dem modifizierten Laxen/Perry-System zu bauen. Das Ganze paßte auf eine Europa-Karte (CPU, 32K RAM, 32K EPROM, ein serieller und zwei parallele Chips). Der Meta-Compiler von Laxen/Perry wurde umgeschrieben, auf daß er ROM-fähigen 6809-Code erzeugte, und an einem der Parallel-Chips wurde ein SCSI-Interface in Software installiert. Auf der daran angeschlossenen Harddisk und den zwei 3.5" Floppies ließen sich nun per SCSI-Kommando bequem individuelle Blocks (zu je zwei 512-kByte Sektoren) lesen und schreiben. Wie aber sollte man nun diese großen Datenmengen unter Kontrolle behalten?

Traditionelle FORTH-Systeme verwalten ihre Massenspeicher nach einem sehr simplen Schema: Ihre Vorstellung von der Welt außerhalb ihres direkt adressierbaren Speichers beschränkt sich auf eine geordnete Folge von 1-kByte Blocks, auf die nach der Methode der sogenannten *Virtuellen Speicherverwaltung* über Blockpuffer zugegriffen wird. Obwohl dadurch der Filestruktur etliche Beschränkungen auferlegt werden, lassen sich auch deutliche Vorteile erkennen: Kurze Zugriffszeiten und eine einfache Implementation. Die eigentlichen Probleme liegen darin, daß man sich die Blocknummern seiner Files auf der Disk merken und Kommandos wie `126 load` oder `221 edit` eingeben muß. Zur Erleichterung des Lebens existieren Utilities, die z.B. die jeweils ersten Zeilen aufeinanderfolgender Blocks ausgeben und so bei der Datensuche helfen. Daher benutzen die meisten kommerziellen und public-domain FORTH-Systeme heutzutage das Filesystem der Hostmaschine, wobei sie manchmal sogar noch die alte Blockstruktur oben draufsetzen (so auch Laxen/Perry). In diesem Falle wird auf einzelne Blocks durch Angabe von Filename und Blocknummer zugegriffen. Leider handelt man sich damit aber die Nachteile beider Welten ein: Schlechte Performance und Abhängigkeit von einem vorgegebenen Betriebssystem - und die Unflexibilität der Blockfiles.

Stichworte

- ✗ SCSI,
- ✗ Files,
- ✗ Directories,
- ✗ Subdirectories,
- ✗ Pathnames

Bei einem Stand-Alone System fällt diese Möglichkeit weg. Ich beschloß, ein simples hierarchisches Filesystem zu entwickeln, das die Block-Philosophie beibehält, aber Directories und Subdirectories als einzelne Blocks und Files als sequentiell aufeinanderfolgende Blocks unterhält. Die Kommandos `pwd`, `ls`, `cd`, `rm` und `mkdir` wurden von UNIX™ entliehen, obwohl sie natürlich bei weitem nicht die Mächtigkeit der Originale erreichen. Ich nannte es *BlocksWorld*; der Name hat aber nichts mit dem berühmten KI-Programm zu tun. Im Herbst 1986 lief der erste Prototyp, zunächst cross-kompiliert auf einem CP/M-Rechner, dann stand-alone. Seitdem war das System viel in Benutzung und entwickelte sich weiter.

Directories und Subdirectories

Alle angeschlossenen Harddisks und Floppies sind logisch in 1024-Byte-Blocks unterteilt. Die Blocknummern zählen von 1 bis zu einem Device-abhängigen Maximalwert, der auf meinem System 719 für die beiden Floppies und 22139 für die 20-MByte Harddisk beträgt.

Der erste Block auf jedem Device enthält das Root-Directory. Ein Directory besteht immer aus einem einzelnen 1024-Byte-Block und kann maximal 64 Einträge enthalten. Jeder Eintrag spezifiziert entweder ein File oder ein Subdirectory.

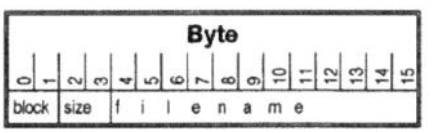


Bild 1:Struktur eines Directory-Eintrages

Bild 1 zeigt die Struktur eines Directory-Eintrages. Er besteht aus drei Feldern: Einem 2-Byte *Block Field*, einem 2-Byte *Size Field* und einem 12-Byte *Name Field*. Das Block-Feld enthält die Nummer des ersten Blocks des Files, oder Null wenn dieser Directory-Eintrag nicht belegt ist. Das Size-Feld gibt die Größe des Files in Blocks an, wobei Null ein Subdirectory kennzeichnet. Filenamen sollten nicht länger als 12 Bytes sein und dürfen aus allen ASCII-

Zeichen außer Blanks und Slashes ('/') bestehen. Für Namen mit weniger als 12 Bytes wird der Rest mit Blanks aufgefüllt, und zu lange Namen werden einfach abgeschnitten.

Die ersten beiden Einträge jedes Directories sind immer . und .. Sie dienen als Verweise auf sich selbst und auf das Parent-Directory. Wenn ein neues Directory mit `mkdir` kreiert wird, werden diese beiden Einträge automatisch erzeugt. Im Root-Directory befindet sich außerdem immer ein spezielles File `heap`. In ihm stecken alle freien Blocks im Filesystem, und er kann geöffnet und benutzt werden wie ein normales File (meist für temporäre Daten oder als Scratch). Sobald aber Blocks für ein neues File mit `mkdir` oder `mkfile` angefordert werden, muß er sie herausrücken und sich dabei entsprechend verkleinern lassen.

Byte															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0001		0000		.											
0001		0000		.	.										
0002		02CE		h	e	a	p								
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0

Bild 2: Ein Root Directory nach der Initialisierung

Bild 2 zeigt Block 1 eines frisch initialisierten Filesystems auf einer 720-kByte 3.5-Zoll-Floppy. Gemäß 6809-Konvention findet sich das höherwertige Byte eines 16-bit-Wortes an erster Stelle. Die Block-Felder des .-Eintrages und des ..-Eintrages sind beide 1, d.h. sie zeigen auf sich selbst (Das Root-Directory ist als einziges sein eigenes Parent-Directory). Ihre Size-Felder

Byte															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0001		0000		.											
0001		0000		.	.										
0003		02CD		h	e	a	p								
0002		0000		u	s	r									
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0

Bild 3: Root Directory nach der Erzeugung des "usr"-Directories

sind Null, weil es sich um Directory-Verweise handelt. Das Heap-File beginnt bei Block Nummer 2 und hat die Größe 718 (hex 0C2E), eins weniger als das Gesamtfassungsvermögen von 719 Blocks. Alle übrigen 61 Directory-Einträge sind leer und enthalten daher die Blocknummer Null.

Nun geben wir den Befehl `mkdir usr` ein, erzeugen also ein neues Directory mit dem Namen `usr`. Bild 3 zeigt das geänderte Root-Directory: Das Heap-File beginnt nun bei Block Nummer 3 und hat noch 717 (hex 02CD) blocks übrig, während ein neuer Eintrag `usr` ein Directory in Block 2 anzeigt. Dieses neue Directory (in Bild 4) zeigt mit dem .-Eintrag auf sich selbst (2) und mit dem ..-Eintrag auf sein Parent-Directory, das Root-Directory in Block 1. Alle anderen Einträge sind leer.

Byte															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0002		0000		.											
0001		0000		.	.										
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0
0000		0000		0	0	0	0	0	0	0	0	0	0	0	0

Bild 4: Das "usr"-Directory in Block 2

Files

Ein File ist einfach eine Anzahl zusammenhängend aufeinanderfolgender Blocks. Je nach Anwendung kann er FORTH Source-Screens, ASCII-Text oder binäre Daten enthalten. Sobald ein File geöffnet ist, kann jeder seiner Blocks mit einem einzigen Diskzugriff erreicht werden, da sich seine logische Adresse als Summe seiner relativen Nummer und des Startblocks des Files errechnen läßt.

Devices und Pfadnamen

Jedes einzelne BlocksWorld Filesystem residiert auf einem eigenen SCSI-Device und wird durch eine Nummer ("Devicenumber") identifiziert. Bei meinem 6809-System besitzt die Harddisk die Device-Nummer 0 und die Floppies sind 1 bzw. 2. Da 16-Bit-Zahlen für die Blocknummern verwendet werden, ist die maximale Größe eine Filesystems 65535 Blocks (Block Null wird nicht benutzt: *reserved* auf deutsch). Harddisks mit größerer Kapazität als 64 MByte müssen in mehrere logische Devices aufgeteilt werden. Ein Pfadnamen identifiziert ein gegebenes File. Wenn er mit einer Nummer (d.h. der Devicenumber) beginnt, heißt er *absoluter* Pfadname. Das Kommando

```
cd 1/usr/simul ok
```

macht beispielsweise `simul` zum aktuellen Directory. Ein *relativer* Pfadname, z.B.

```
cd ../languages ok
```

ist also einer, der nicht mit einer Nummer beginnt. Es spezifiziert den Zugriffspfad relativ zum aktuellen Directory.

Im Gegensatz zu UNIX-ähnlichen Systemen versteht BlocksWorld Pfadnamen nur im Zusammenhang mit dem `cd`-Kommando. Um mit einem File zu arbeiten, muß zuerst mit `cd` sein Directory aufgesucht und es dann mit `open <file>` geöffnet werden. Das ist nicht so unbequem wie es sich zunächst anhören mag, weil `open` einen Dictionary-Eintrag für das File kreiert und somit auf das File später einfach durch Exekutieren seines Namens zugegriffen werden kann, auch wenn das aktuelle Directory mittlerweile geändert wurde.

File Control Blocks

Informationen über offene Files finden sich in *File Control Blocks* (FCB's). Ein FCB (siehe Bild 5) ist einem Directory-Eintrag ähnlich, besitzt aber zusätzlich ein *Dev-Field* für die Devicenummer. Ein File wird somit vollständig beschrieben durch Device-

Byte																	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
dev		block		size		f	i	l	e	n	a	m	e				

Bild 5: Struktur eines File Control Blocks

nummer, Blocknummer und Filesize. Viele Files auf verschiedenen Devices können gleichzeitig offen sein, ihre Anzahl ist nur durch den vorhandenen Speicher begrenzt.

Löschen von Files

Wie Ihnen vielleicht aufgefallen sein wird, gibt es bei dem hier beschriebenen Filesystem ein Problem: Wie kann man ein nicht mehr benötigtes File oder Directory löschen? Man kann zwar einfach eine Null in das Block-Feld im Directory-Eintrag schreiben (und somit den Eintrag als *frei* markieren), wird damit aber nicht die vom File belegten Blocks zur Wiederverwendung freigeben (Wie oben beschrieben, werden Blocks für neue Files stets vom *heap* Systemfile abgezwackt).

Die *richtige* Lösung läge wohl bei der Verwendung von File-Management-Tabellen und komplizierteren Algorithmen (und mithin der Einführung von fragmentierten Files). Hier jedoch wird eine ziemlich brutale Methode angewandt: Alle Blocks im Filesystem, die sich oberhalb des gelöschten Files befinden, werden verschoben und schließen so die Lücke. Danach müssen noch alle Directory-Einträge, die auf verschobene Files zeigen, aktualisiert und das *heap*-File entsprechend vergrößert werden.

Nach nunmehr mehrjähriger Benutzung kann ich sagen, daß das eigentlich gar keine so schlechte Lösung ist. Wenn das File noch ziemlich jung war, muß nicht viel bewegt werden (oder gar nichts, wenn es das zuletzt kreierte File war), und die Verschiebeoperation geht recht schnell mit dem SCSI Copy-Kommando, das direkt von Device zu Device kopiert. Alte Files hingegen löscht man recht selten. Außerdem wird man in vielen Fällen die Blocks durch einfaches Umbenennen des Files wiederverwenden können.

Die Implementation

Das Listing zeigt die 17 Source-Screens, die hier relevant sind, zusammen mit ihren Shadow-Screens (Shadows enthalten ausschließlich Kommentare und werden zur Rechten der zu ihnen gehörenden Sources ausgedruckt). Einige von ihnen (2, 3, 4, 5 und 17) sind noch Relikte der ursprünglichen Laxen/Perry-Implementation. Sie haben sich aber in einigen Details geändert und sind darum hier aufgeführt. Nicht aufgeführt sind die (systemabhängigen) Primitiv-Funktionen für den SCSI-Zugriff:

- ✗ `scRead (bufHead – )`: Liest einen 1024-Byte-Block vom SCSI-Device in den Blockpuffer
- ✗ `scWrite (bufHead – )`: Schreibt einen 1024-Byte-Block vom BlockPuffer zum SCSI-Device
- ✗ `scCopy (srcBlk srcDev dstBlk dstDev – )`: Kopiert einen Block direkt von Device zu Device.

Command Line Flags

Screen 1 zeigt ein Werkzeug, das sich auch bei anderen Anwendungen als nützlich erwiesen hat. Es emuliert das Verhalten von Command-Line-Flags und dient zur Modifikation bestimmer Befehle. Das `ls`-Kommando zum Beispiel akzeptiert die Flags: `-l` (long) und `-t` (tree), siehe Bild 6. Ein Aufruf

Bild 6: Beispiele für Command-Line-Flags

```
ls
    heap    usr/    etc/    sysgen/  ok

-l ls
       703    18 heap
         2     0 usr/
         3     0 etc/
       166     0 sysgen/
ok

-t ls
    heap
    usr/
        simul/
            Flight
        languages/
            Basic
    etc/
        Test
        Tools
    sysgen/
        Meta
        Kernel
        rom
        Prommer
ok

-t -l ls
       703    18 heap
         2     0 usr/
             4     0 simul/
                 5    40 Flight
            45     0 languages/
                46    40 Basic
         2     0 etc/
            86    40 Test
           126    40 Tools
       166     0 sysgen/
           167    64 Meta
           231   400 Kernel
           631    32 rom
           663    40 Prommer
ok
```

von `ls` ohne Flag schreibt die Filenamen in eine Zeile, wobei es Directories am Ende mit einem Slash '/' markiert werden. Mit der `-l`-Option zeigt `ls` auch den Startblock und die Größe für jedes File an. Die `-t`-Option läßt `ls` rekursiv den Directory-Baum absuchen und druckt alle Files und Directories unter dem aktuellen Directory aus. Wie Bild 6 zeigt, kann man auch die Kombination von `-t` und `-l` aufrufen (Das Ausgedruckte wird aber etwas unleserlich).

Es können maximal 16 verschiedene Flags definiert werden. Natürlich können verschiedene Kommandos die selben Flags verwenden. Ich habe beispielsweise ein `ps`-Kommando, das alle gegenwärtig aktiven Tasks anzeigt und auch die `-l`-Option akzeptiert, oder eine File-Backup-Utility, die mit `v` nach dem Kopieren auch verifiziert.

Kommandos, die Command-Line-Flags verwenden wollen, können die einzelnen Flags mit `-x?` testen und sollten nach Beendigung ihrer Aufgabe alle Flags mit `clrFlags` löschen.

Blockzugriffe

Screen 2 definiert ein paar oft benötigte Konstanten und die Block-Puffer-Arrays und -Variablen. Die Screens 3 bis 5 implementieren die virtuelle Speicherverwaltung. Die Worte `buffer` und `block` (Screen 4) sind die wichtigsten Werkzeuge für den Filezugriff. Sie verhalten sich genau wie in der Laxen/Perry-Version, erwarten also eine Blocknummer auf dem Stack und geben eine Pufferadresse zurück. Anders `(buffer)` und `(block)`: Sie brauchen jetzt eine Device- und eine Blocknummer als Argumente, um den Zugriff auf jeden beliebigen Block im System zu ermöglichen.

Initialisierung

Screen 6 kreiert vier FCB's für den internen Gebrauch, und sechs kurze Worte, die den Zugriff auf die einzelnen FCB-Felder erleichtern.

Die doppeltgenaue Variable `dir#` in Screen 7 enthält die Device-und Blocknummer des aktuellen Directorys. Ihr Initialwert ist Block Eins auf Device Null, in meinem Falle das Root-Directory auf der Harddisk.

Das Wort `directory` ist sehr nützlich für den schnellen Directory-Wechsel. Es wird in der Form

```
directory <dirname> ok
```

benutzt. Ein Wort `<dirname>` wird kreiert, das sich das gegenwärtig aktuelle Directory merkt und so später aus einem beliebigen anderen Directory durch einfaches Executieren von `<dirname>` eine Rückkehr ermöglicht.

Screen 8: `setName` liest einen Filenamen bis zu einem vorgegebenen Delimiter (das mag ein Blank oder ein Slash sein). `setName` wird von `setFcb` zur Initialisierung eines File Control Blocks verwendet, aber auch von den Worten `ren` und `from`. `dirEnter` durchsucht das Directory in einem Blockpuffer nach einem unbenutzten Eintrag und installiert die Daten aus dem FCB dort. Wird kein freier Platz gefunden, erfolgt ein Abbruch.

`fallot` implementiert den oben beschriebenen File Allocation Mechanismus, indem es das `heap`-File verkleinert und die Blocknummer des ersten so freigewordenen Blockes zurückgibt.

`mkdir` ist das Top-Level-Kommando zur Erzeugung neuer Directories, es erwartet den Directory-Namen im Input-Stream, reserviert einen Block auf der Disk und initialisiert ihn als Directory.

Mit `mkfile` (Screen 10) wird wird ein neues File kreiert. Das Kommando

```
40 mkfile Test ok
```

erzeugt das File *Test* mit einer Größe von 40 Blocks im aktuellen Directory, und füllt alle Blocks mit Blanks.

`mkdir` und `mkfile` überprüfen nicht, ob ein File gleichen Namens bereits im aktuellen Directory existiert. Falls dem so ist, bleibt eines der beiden Files solange unerreichbar, bis der andere umbenannt wurde.

Zur Initialisierung eines Filesystems dient `mkfs`. Die beiden Kommandos

```
cd 1 719 mkfs ok
```

richten ein neues Filesystem auf der Floppy in Device 1 ein. Das Kommando `mkfs` ist gefährlich. Es überschreibt das Root-Directory und fragt vorher nicht nach einer Bestätigung. Wenn es einmal fälschlich eingegeben wurde, kann aber ein sofortiger Druck auf den Resetknopf doch noch Rettung bedeuten, weil `mkfs` von sich aus keinen `flush` ausführt.

Suchen und Finden

Screen 11 zeigt einige Worte zum Suchen und Ausdrucken von Filenamen. `pwd` in Screen 12 zeigt den absoluten *Path to Working Directory* - zum aktuellen Directory also - an, beginnend mit der Device-Nummer, gefolgt von den durch Slashes getrennten Directory-Namen:

```
pwd 0/usr/tools/math ok
```

Die eigentliche Arbeit wird dabei von `(pwd)` erledigt, einer rekursiven Routine die - entweder die Device-Nummer ausdruckt, wenn ihr Argument gleich 1 (Root-Directory) ist - oder anderenfalls ihren eigenen Pfadnamen ausdruckt, indem sie sich selbst mit ihrer Parent-Directory-Nummer aufruft, dann einen Slash und schließlich den aktuellen Directory-Namen anfügt.

`ren` dient zum Umbenennen von Files und wird so benutzt:

```
ren <oldName> <newName> ok
```

Zur Änderung des aktuellen Directorys schreibe man

```
cd <Pfadname> ok
```

`cd` schneidet sich die Directory-Namen aus dem Pfadnamen heraus und ruft wiederholt `(cd)` auf. Wird irgendwo in dem String von Directory-Namen eine Zahl entdeckt, wird die Suche im Root-Directory des durch sie bestimmten Devices fortgesetzt. Siehe auch oben die Erklärung von absoluten und relativen Pfadnamen.

Screen 14 enthält das `ls`-Kommando. Auch hier wird die Arbeit an eine rekursive Funktion - `(ls)` - übergeben. Sie extrahiert alle Files im aktuellen Directory und druckt ihre Namen. Wenn das `-l` Flag gesetzt ist, erfolgt der Ausdruck im *langen Format*, bei

dem auch die Startadresse und Filegröße mit angezeigt werden. Das `-t` Flag läßt `(ls)` rekursiv Unterdirectories abarbeiten. Zu beachten ist, daß `(block)` bei jedem Schleifendurchgang neu aufgerufen wird. Es wäre nicht genug, lediglich einen Pointer in das Directory im Puffer zu halten, weil der Puffer bei sehr tief verschachtelten Subdirectories überschrieben worden sein kann.

`rm` ist in Screen 16 definiert. Wenn das zu entfernende File ein Directory ist, sollte es leer sein, anderenfalls wird sich `rm` beschweren. Das aktuelle Directory und das *heap*-File im Root-Directory werden auf den neuen Stand gebracht, die Größe des Files bestimmt und alle Blocks oberhalb mit der `scCopy` Funktion nach unten verschoben. Die rekursive Hilfsfunktion `adjDir` (Screen 15) durchsucht alle Directories und korrigiert Directory-Einträge, die auf verschobene Files oder Directories verweisen (Schätzen Sie einmal, wie oft das Filesystem auf meiner Harddisk ruiniert wurde, bis diese Funktion endlich fehlerfrei (?) war). `rm` entfernt Files ohne Ansehen der Person. Es ist nicht ratsam, es beispielsweise auf *heap* im Root-Directory anzuwenden.

Screen 17 schließlich zeigt die BlocksWorld-Versionen der Laxen/Perry Funktionen zum Definieren und Öffnen von Files.

Resümee

Das hier beschriebene Filesystem hat sich in der FORTH-Umgebung als durchaus nützlich und effektiv erwiesen.

Es bleiben aber noch viele Punkte, in denen das System verbessert werden könnte. Es wäre zum Beispiel nett, einen Time-and-Date-Stamp in jedem Directory-Eintrag zu haben, oder die Block- und die Size-Fields auf 32 Bit Breite zu vergrößern, um Filesysteme größer als 64 MBytes zuzulassen. Pfadnamen sollten nicht auf das `cd`-Kommando beschränkt sein, und man müßte Werkzeuge wie `fsck` (zur Überprüfung des Filesystems) oder `cp` (zum Kopieren ganzer Subdirectory-Bäume) entwickeln. Für Leute, die lieber mit narrensicheren Systemen arbeiten (wohl keine FORTH-Programmierer) wäre ein bißchen mehr Error-Checking vonnöten.

Quellenangaben

[1] Laxen/Perry Public Domain F83-FORTH, FORTH Interest Group, Orange County Chapter

Vollständige Quelleexte erhältlich bei:
✉ Alexander Burger
Max-Reger-Str.18
D-8050 Freising
☎ 08161-83264
Fax 08161/81538

1

```
 0 \ Command flags
 1 variable flags
 2 : clrFlags ( -- )   flags off  ;
 3 : flg! ( n -- )   flags @  or flags !  ;
 4 : flg? ( n -- f )   flags @  and  0<>  ;
 5
 6 : -l ( -- )  1 flg! ;          : -l? ( -- f )  1 flg? ;
 7 : -t ( -- )  2 flg! ;          : -t? ( -- f )  2 flg? ;
 8 : -u ( -- )  4 flg! ;          : -u? ( -- f )  4 flg? ;
 9 : -v ( -- )  8 flg! ;          : -v? ( -- f )  8 flg? ;
10
11 : access ( n file -- blk dev )
12     @  2dup 4+ @ over <= swap 0< or
13     if  scr off  ." File access error"  quit  then
14     2@  >r + r>  ;
15
```

2

```
 0 \ Devices                          BLOCK I/O
 1  4 constant #buffers          1024 constant b/buf
 2 64 constant f/dir
 3 12 constant b/fname          b/fname 4+ constant b/dir
 4 b/dir 2+ constant b/fcb
 5 32768 constant limit
 6 #buffers 1+ 8 * 2+ constant >size
 7 limit b/buf #buffers * -  constant first
 8 first >size - constant init-r0
 9 first  >size - constant >buffers
10 first  2-  constant >end
11 : buffer#     ( n -- adr )   8* >buffers +    ;
12 : >update     ( -- adr )   1 buffer# 6+  ;
13
14
15
```

3

```
 0 \ Devices                          BLOCK I/O
 1 forth definitions
 2 : capacity ( -- n )
 3     file @  4+ @  ;
 4 : latest? ( blk dev  -- n | a f )
 5     2dup  1 buffer# 2@  d=
 6     if  2drop  1 buffer# 4+ @  false rdrop then ;
 7 : absent? ( blk dev  -- a f )
 8     latest?  false #buffers 1+ 2
 9     do  drop 2dup i buffer# 2@ d=
10         if  2drop i leave  else  false then
11     loop  ?dup
12     if  buffer#  dup >buffers 8 cmove  >r >buffers dup 8+
13         over r> swap - cmove>  1 buffer# 4+ @ false
14     else  >buffers 2!  true  then ;
15
```

21

```
04jul87abu \ Command flags                                  04jul87abu
           flags   holds the flags set recently
           clrFlags ( -- )    Clear all flags
           flg! ( n -- )    Set one or more flags
           flg? ( n -- f )   Test one or more flags

           Several predefined flags

           access
               Translate locical block number in file to physical block-
               and device-number. Perform a range check.
```

22

```
24jan87abu \ Devices                    BLOCK I/O                28feb87abu
           These variables are used by the BLOCK I/O part of the system.
           Unlike FIG Forth the buffers are managed in a true least
           recently used scheme.  The are maintained in memory as an array
           of 8 byte entries, whose format is defined at left.  Whenever
           a block is referenced its pointer is moved to the head of the
           array, so the most recently used buffer is first. Thus multiple
           references are very fast.  Also we have eliminated the need for
           a null at the end of each block buffer so that the size of a
           buffer is now exactly 1024 bytes.
           The format of entries in the buffer-pointer array is:
              0-1 is Device Number        2-3 is Block number
              4-5 is Address of Buffer    6-7 is Update Flag
           buffer#   Return the address the the nth buffer pointer.
           >end      Return a pointer to just past the last buffer packet.
           >update   Return a pointer to the update flag.
```

23

```
24jan87abu \ Devices                    BLOCK I/O                24jan87abu

           capacity ( -- n )
               The number of blocks in the current file
           latest? ( blk dev  -- n | a f )
               Check if the wanted block is the first in the list. If so,
               return directly from the calling word.
           : absent? ( blk dev  -- a f )
               Search through the block/buffer list for a match. If it is
               found, bring the block packet to the top of the list and
               return a false flag and the address of the buffer.  If the
               block is not found, return true, indicating it is absent,
               and the second parameter is garbage.
```

4

```
 0 \ Devices                          BLOCK I/O                 24jan87abu
 1 : update ( -- )    >update on   ;
 2 : discard ( -- )    1 >update !  ;
 3 : missing ( -- )
 4     >end 2- @ 0< if  >end 2- off  >end 8 - scWrite  then
 5     >end 4- @  >buffers 4+ ! ( buffer )  1 >buffers 6+ !
 6     >buffers dup 8+ #buffers 8* cmove>    ;
 7 : (buffer) ( blk dev -- a )   pause  absent?
 8     if  missing  1 buffer#   4+ @  then  ;
 9 : buffer ( n -- a )    file access  (buffer)  ;
10 : (block) ( blk dev -- a )
11     (buffer)  >update @ 0>
12     if  1 buffer#  dup scRead  6+ off  then  ;
13 : block ( n -- a )    file access  (block)  ;
14 : inBlock ( n -- a )    inFile access  (block)  ;
15
```

24

```
\ Devices                          BLOCK I/O                 24jan87abu
update   Mark the most recently used buffer as modified.
discard  Mark the most recently used buffer as unread.
missing   Writes the least recently used buffer to disk if it
    was modified, and moves all of the buffer pointers down by
    one, making the first one available for the new block.  It
    then assigns the newly available buffer to the new block.
(buffer)  assigns a buffer to the specified block on the given
    device. No disk read is performed. Leaves the buffer address
buffer  assigns a buffer to the specified block.
    No disk read is performed. Leaves the buffer address.
(block) Leaves the address of a buffer containing the given
    block on the given device. Reads the disk if necessary.
block   Leaves the address of a buffer containing the given
    block. Reads the disk if necessary.
inBlock  like block, but for the inFile.
```

5

```
 0 \ Devices                          BLOCK I/O                 24jan87abu
 1 : empty-buffers ( -- )
 2     first limit over - erase
 3     >buffers #buffers 1+ 8* erase
 4     first 1 buffer#   #buffers 0
 5     do   dup on  4+  2dup !   swap b/buf + swap  4+
 6     loop   2drop   ;
 7 : save-buffers ( -- )
 8     1 buffer#   #buffers 0
 9     do   dup @ 1+
10       if  dup 6+ @ 0< if  dup scWrite  dup 6+ off  then
11         8+ then   loop   drop   ;
12 : flush ( -- )
13     save-buffers  empty-buffers  ;
14 defer load
15
```

25

```
\ Devices                          BLOCK I/O                 24jan87abu
empty-buffers
    First wipe out the data in the buffers.  Next initialize the
    buffer pointers to point to the right addresses in memory
    and set all of the update flags to unmodified.


save-buffers
    Write back all of the updated buffers to disk, and mark them
    as unmodified.  Use this whenever you are worried about
    crashing or losing data.

flush     Save and empties the buffers. Used for changing disks.

load  Interpret a screen as if it were typed in.
```

6

```
 0 \ Devices                          BLOCK I/O                 04jul87abu
 1 dos definitions
 2 create workFcb   b/fcb allot
 3 create .fcb    0 ,  0 ,  0 ,
 4 ascii . c,  b/fname 1- bl fill
 5 create ..fcb   0 ,  0 ,  0 ,
 6 ascii . c,  ascii . c,  b/fname 2- bl fill
 7 create fromFcb  b/fcb allot
 8
 9 : f@blk ( fcb -- n )  2+ @ ;
10 : f!blk ( n fcb -- )  2+ ! ;
11 : f@size ( fcb -- n )  4+ @ ;
12 : f!size ( n fcb -- )  4+ ! ;
13 : f+blk ( n fcb -- )  tuck f@blk + swap f!blk ;
14 : f-size ( n fcb -- )  tuck f@size swap - swap f!size ;
15
```

26

```
\ Devices                          BLOCK I/O                 08jan87abu
workFcb     General purpose scratch fcb
.fcb
  The fcb for self reference
..fcb
  The fcb to refer to the parent directory
fromFcb    The standard input file

f@blk ( fcb -- n ) Get block number from the fcb
f!blk ( n fcb -- ) Store the block number in fcb
f@size ( fcb -- n ) Get the file size from the fcb
f!size ( n fcb -- ) Store the file size in fcb
f+blk ( n fcb -- ) Increment the block number in fcb by n
f-size ( n fcb -- ) Decrement the file size in fcb by n
```

7

```
 0 \ File system initialization                          04jul87abu
 1 2variable dir#    1 dir# 2+ !
 2 : device ( -- dev )    dir# @  ;
 3 : heapFcb ( -- fcb )
 4     1 device (block)    [ b/dir 2* 2- ] literal + ;
 5 : parent ( n -- n' )  device (block)  b/dir + @ ;
 6 : rdDir ( -- a )  dir# 2@ (block) ;
 7
 8 forth definitions
 9 : directory ( -- )
10     dir# 2@ 2constant   does> 2@  dir# 2! ;
11
12
13
14
15
```

27

```
\ File system initialization                             04jul87abu
dir#  Contains current directory
device ( -- dev )  return the current device number
heapFcb ( -- fcb ) Return the address of the heap entry
    ( note that there is no 'device' field to access )
parent ( n -- n ) Return the number of the parent directory
rdDir ( -- a ) Read the current directory


directory
    Define a name which, when executed, switches back to this
    directory
```

8

```
 0 \ File creation                                        24jan87abu
 1 dos definitions
 2 : setName ( adr del -- )
 3     over b/fname blank
 4     word  count dup b/fname > if drop b/fname then
 5     0 ?do  count  2 pick c!  swap 1+ swap  loop  2drop ;
 6 : setFcb ( del blkNo size fcb -- )
 7     device over !  rot over f!blk  tuck f!size
 8     6+ swap setName ;
 9 : dirEnter ( fcb a -- )
10     swap 2+ swap    dup b/buf +     begin
11         over @ 0=  if drop  b/dir cmove update exit then
12     swap b/dir + swap 2dup = until  abort" Directory full" ;
13
14
15
```

28

```
\ File creation                                          03feb87abu
setName ( adr del -- )
  Parse a file name delimited by del and put it at adr


setFcb ( blkNo size fcb -- )
  Init a fcb

dirEnter ( fcb a -- )
  Enter a new file name into the directory
  starting at a
```

9

```
 0 \ File system intitialization                          10jan87abu
 1 : fallot ( size -- blkNo )
 2     heapFcb 2dup f@size > abort" Not enough disk space"
 3     dup f@blk -rot  2dup f+blk f-size update ;
 4 : iniDir ( .blk ..blk -- )
 5     ..fcb f!blk  dup  .fcb f!blk device (block)  dup b/buf
 6     erase  .fcb  over dirEnter  ..fcb swap dirEnter ;
 7 : (mkfile) ( del blkNo size -- )
 8     workFcb setFcb  workFcb rdDir dirEnter ;
 9
10 forth definitions
11 : mkdir ( -- )
12     1 fallot  bl over 0 (mkfile)  dir# 2+ @  iniDir  ;
13
14
15
```

29

```
\ File system intitialization                            24jan87abu
fallot ( size -- blkNo )
    Allocate space for a file of size blocks in the current
    file system
iniDir ( .blk ..blk -- )
    Init a directory entry

(mkfile) ( del blkNo size -- )
    File creation primitive


mkdir ( -- )
    Create a new directory
```

10

```
0 \ File system intitialization                          03feb87abu
1 : mkfile ( n -- )
2     1 ?enough  dup fallot  swap  2dup  bl -rot (mkfile)
3     0 do  dup device (buffer)  b/buf blank update  1+ loop
4     drop ;
5 : mkfs ( n -- )
6     1 ?enough    1-
7     1 1 iniDir  1 device (block)  b/dir 2* +  2 over !
8     2+ tuck !
9     2+ 26725 ( 'he' ) over !  2+ 24944 ( 'ap' ) over !
10    2+ 8 blank  update ;
11
12
13
14
15
```

11

```
0 \ Directory functions                                  24jan87abu
1 : .file ( fcb -- )
2     6+ b/fname 0 do count dup bl = if drop leave then
3     emit loop  drop ;
4 : file? ( -- )  file @ .file ;
5
6 dos definitions
7 : (findFile) ( blk dev -- fcb )
8     (block) dup b/buf +  begin
9       over @ if
10        over 4+  workFcb 6+ b/fname compare
11        0= if drop 2- exit then
12      then    swap b/dir + swap
13    2dup = until abort" File not found" ;
14
15
```

12

```
0 \ Directory functions                                  24jan87abu
1 : findFile ( blk dev del -- fcb )
2     0 0 workFcb setFcb (findFile) ;
3 : .dir ( n -- )
4     dup parent  device (block)
5     begin 2dup @ <> while b/dir + repeat
6     2- .file drop ;
7 : (pwd) ( dir# -- )
8     dup 1 = if drop  device (.) type
9     else  dup parent recurse  ascii / emit  .dir then ;
10
11 forth definitions
12 : pwd ( -- )
13    cr  dir# 2+ @  (pwd) ;
14
15
```

13

```
0 \ Directory functions                                  24jan87abu
1 : ren ( -- )
2     dir# 2@ bl  findFile  6+  bl setName update ;
3 dos definitions
4 : (cd) ( del -- )
5     >in @ over word  number?
6     if  drop 1 swap dir# 2!   2drop
7     else  2drop  >in !  dir# 2@ rot findFile
8         dup f@size abort" Not a directory"
9         f@blk  dir# 2+ !  then ;
10 forth definitions
11 : cd ( -- )
12    0  >in @  bl word c@  swap >in !
13    1 ?do  here i+ c@  ascii / = if 1+ then loop
14    0 ?do  ascii / (cd)  loop  bl (cd) ;
15
```

14

```
0 \ List directory contents                              24jan87abu
1 dos definitions
2 : (ls) ( dir# level -- ) ( -l -t )
3     4+ >r   b/dir 2*  f/dir 2 do
4       over device (block)  over +
5       dup @ if  2-      j ( level ) spaces
6         -1? if  dup f@blk 6 u.r  dup f@size 6 u.r space  then
7         dup .file  dup f@size 0= if  ascii / emit then
8         -1? -t? or if  cr  then
9         dup f@size 0=  -t?  and if dup f@blk j recurse  then
10      then
11    drop  b/dir +  loop  rdrop 2drop ;
12 forth definitions
13 : ls ( -- ) ( -l -t )
14    cr  dir# 2+ @  0  (ls)  clrFlags ;
15
```

15

```
0 \ Remove a file                                        27feb87abu
1 dos definitions
2 2variable rm# ( blk size )
3
4 : adjDir ( blk -- )
5     0  f/dir 0 do
6       over device (block)  over +  dup @  ?dup if
7         rm# @ u> if  rm# 2+ @  over -! update  then
8         dup 2+ @ 0=  i 1 > and if  dup @ recurse  then
9       then  drop  b/dir + loop  2drop ;
10
11
12
13
14
15
```

30

```
\ File system intitialization                            24jan87abu
mkfile ( n -- )
    Create a file with a size of n blocks and init it with
    blanks

mkfs ( n -- )
    Create a file system of size n.  Init the 'heap' file
```

31

```
\ Directory functions                                    24jan87abu
.file ( fcb -- )
    Print a file name

file? ( -- )  Print the name of the current file


(findFile) ( d -- fcb )
    Find the file with the name in workFcb in the
    current directory
```

32

```
\ Directory functions                                    24jan87abu
findFile ( d del -- fcb )  Parse a file name and try to find it

.dir ( n -- )
    Print the name of the directory in block n


(pwd) ( dir# -- )
    Print the path to the directory dir#


pwd ( -- )
    Print the path to the current directory
```

33

```
\ Directory functions                                    24jan87abu
ren ( -- )
    Rename a file

(cd) ( -- )
    Single step in directory changes


cd
    Change the current directory
```

34

```
\ List the directory contents                            21jan87abu

(ls) ( dir# level -- ) ( -l -t )
    Recursive routine that lists the contents of a directory.
    If the -l flag is set, the first block and the size of
    the file is also printed.  If the -t flag is set, sub-
    directories will be displayed recursively.


ls ( -- ) ( -l -t )
    List the contents of the current directory
```

35

```
\ Remove a file                                          03feb87abu

rm# ( blk size )
    contains first block and size of the file to remove
adjDir ( blk -- )
    Adjust all directory entries that point into the moved area
```

16

```
0 \ Remove a file                                        21jan87abu
1 forth definitions
2 : rm ( -- )
3     dir# 2@ bl findFile  dup f@size  0= if  ( directory )
4       f@blk device (block)  b/dir 2* +  f/dir 2 do
5         dup @ abort" Not empty"
6       b/dir + loop
7     then  drop
8     dir# 2@ (findFile)  2+ dup 2@  rot off update
9     swap ?dup 0= if 1 then  ( Directory size is 1 )
10    dup heapFcb 4+ +!  update  swap rm# 2!   flush
11    heapFcb f@blk  rm# 2@ +  ?do
12      i device  i rm# 2+ @ -  device   scCopy
13    loop   1 adjDir  ;
14
15
```

17

```
0 \ Define and Open files                                24jan87abu
1 : file: ( -- fcb )
2     >in @ create >in !  here  bl 0 0 here  b/fcb allot setFcb
3     does> !files  ;
4 : openFile ( fcb -- fcb )
5     dup  workFcb b/fcb cmove  dir# 2@ (findFile)
6     2+ 2@  2 pick  device over !  2+  2!  ;
7 : ?define ( -- fcb )
8     >in @  defined
9     if  nip >body @  else  drop  >in ! file:  then openFile ;
10 : define ( -- )    ?define  drop  ;
11 : open ( -- )    ?define  !files  ;
12 : from ( -- )
13     fromFcb 6+ bl setName  fromFcb openFile inFile !  ;
14
15
```

36

```
\ Remove a file                                          10jan87abu

: rm ( -- )
  Remove the file with the name following in the input stream
  by shifting all files behind it and adjusting the directories
  where necessary.  This may take some time, if the file
  is rather old!
```

37

```
\ Define and Open files                                  08jan87abu
file: ( -- fcb )
    Define the next word as a file by allocating an fcb in the
    dictionary and parsing the next word as a file name.
openFile ( fcb -- fcb )
    Actually open the file by filling in dev and blk fields.

?define ( -- fcb )
    Define the next word as a file if it does not already exist.
    Leave the address of the file control block.
define ( -- )
    Define the following word as a file name without opening it.
open ( -- )
    Open the following file and make it the current file.
from ( -- )
    Open the following file and make it the current input file.
```

BÜCHERECKE

von Werner Horn und Horst Finsterbusch

Claus Kühnel
"FORTH auf dem Kleincomputer"
Militärverlag, Berlin 1990,
96 Seiten
ISBN 3-327-00941-4,
Preis DM 5,60

Die Broschüre wendet sich vorrangig an die Benutzer von Kleincomputern des Types KC87.

Der Autor versucht, dem Leser die Erkenntnis zu vermitteln, daß es sich bei FORTH nicht einfach um eine weitere Programmiersprache handelt, sonder dem Anwender ein komplettes Programmierwerkzeug zur Verfügung steht.

Die Einführungen zur Handhabung des FORTH-Systems und zur Sprachvermittlung fallen bedingt durch den Umfang der Broschüre knapp aus, sind aber unterstützt durch treffend gewählte Beispiele didaktisch gut formuliert.

Der Leser wird hierbei kaum mit der inneren Struktur von FORTH, Implementationsfragen und der Compilersteuerung konfrontiert.

Vermittelt werden der FORTH-83-Standard sowie ausgewählte Unterschiede zum FIG-FORTH-Standard. Die FORTH-Worte werden in der üblichen Form eines alphabetisch geordneten Glossariums zur Verfügung gestellt. Leider haben sich hier einige Druckfehler eingeschlichen.

Mit Bezug auf den Kleincomputer KC87 werden Ausführungen zur Arbeit mit dem Massenspeicher, zur Kommunikation mit dem Drucker bzw. der Benutzerschnittstelle, zur Bedienung des Editors sowie der Speicheraufteilung gemacht.

Einige kurz kommentierte, vollständig abgedruckte Quell-Listen eines Zeileneditors, Stringverarbeitungs- und Gleitkommapaketes füllen weitere Seiten.

Ob der Autor sein selbstgestecktes Ziel erreichen kann, FORTH als "komplettes Programmierwerkzeug" von anderen Programmiersprachen abzuheben, ist infolge thematischer Auslassungen bzw. der am KC87 gegebenen Benutzeroberfläche zu bezweifeln.

Dem Anspruch, einen Einstieg in FORTH zu ermöglichen, wird die Broschüre durch ihr gutes didaktisches Konzept und der klaren Ausdrucksweise des Autors sehr wohl gerecht.

Vack, Gert-Ulrich:
"Programmieren mit FORTH"
1.Auflage - Berlin:
Verlag Technik, 1990
336 Seiten, 85 Bilder,
17 Tafeln, 38 Programme
ISBN 3-341-00518-8,
Preis DM 35,-

Der Autor behandelt im vorliegenden Buch FORTH als Einheit von Sprache, Betriebssystem und Programmiermethodik, nimmt eine allgemeine Einordnung vor und geht auf die historische Entwicklung von FORTH ein.

Es werden die Grundlagen der FORTH-Programmierung in verallgemeinerter Form dargestellt sowie Konzepte der Implementierung aufgezeigt. Anschließend wird der Wortschatz (Sprach- und Funktionsumfang) in einem Glossarium vermittelt. Die FORTH-Worte sind nach Funktionen geordnet. Ein alphabetisches Wortregister im Anhang des Buches ergänzt die gewählte Darstellungsart. Interessant und unserer Meinung nach gut gelungen ist die Einbeziehung von High-Level-Definitionen ins Glossarium. Durch viele graphische Darstellungen wird die Wirkungsweise von FORTH-Worten in Bezug auf die Stack- und Codesequenzen anschaulich illustriert.

Eine Fundgrube von Lösungsansätzen, Denkanstößen und Methoden bietet das Kapitel "Kompendium der FORTH-Programmierung", wo u.a. zahlreiche komplette Programmlistings zu finden sind. Spezielle Programmierverfahren wie DOER-Konzept, rekursive Programmierung und Makrotechnik werden vorgestellt. Die Möglichkeiten der Interruptanbindung sollten in zukünftigen Auflagen ausführlich beschrieben werden.

Ausführungen zu Programmierumgebungen, zu Aspekten der Softwareentwicklung und -gestaltung, die Beschreibung spezieller FORTH-Prozessoren sowie ein Blick in die Zukunft von FORTH runden das Buch ab.

Den FORTH-System- und Anwendungsprogrammierern wird ein umfangreiches Nachschlagewerk in die Hand gegeben, das über den Wortschatz des FIG-FORTH und FORTH-83 Auskunft gibt. Hier wäre es unserer Meinung nach sinnvoller gewesen, sich auf das FORTH-83 zu beschränken.

Ausführliche Betrachtungen zu den Problemen der Hardware-Einbindung sowie der Multitask- und Multiuserprogrammierung sind wünschenswert.

Inwiefern sich das Buch mit dem gewählten didaktischen Konzept als Anleitung zum Erlernen der Programmiersprache FORTH eignet, muß die Praxis zeigen. Es ist jedoch zu bezweifeln, daß ein Einsteiger, nach 21 Seiten relativ allgemeiner theoretischer Vermittlung der Grundlagen der FORTH-Programmierung, den temporär dreistufigen `CREATE .... DOES>`-Konstrukt begreifen kann.

Die Rezensenten sind der Meinung, daß diese erste große DDR-Monographie zu FORTH auf Grund ihrer hohen Qualität in einer Reihe mit den *FORTH-Klassikern* von Brodie und Zech stehen wird.

Lokale FORTH-Gruppen, die sich regelmäßig treffen:

1000 Berlin	Claus Vogt, ☎ 030/2168938. Treffen am letzten Donnerstag des Monats um 19.30 Uhr in der Technischen Universität Berlin, Mathematikgebäude, 6.Stock im Raum MA 621
4130 Moers 1 Rhein-Ruhr	Friederich Prinz, näheres Tel: 02841/583 98 Jörg Plewe, Tel: 0208/423514, Treffen nach Absprache. Der nächste Termin kann bei Jörg Plewe erreichbar unter obiger Telefonnummer erfragt werden.
6100 Darmstadt	Andreas Soeder, ☎ 06257/2744. Treffen an der VHS an einem Mittwoch in der Mitte des Monats (Termine: 13.9, 11.10, 15.11 13.12 im alten Pädagog, Raum 3-1, 3.Stock).
6800 Mannheim	Lokale Gruppe Rhein-Neckar, Thomas Prinz, ☎ 06271/2830, Ewald Rieger, ☎ 06239/8632. Treffen jeden ersten Mittwoch im Monat im Vereinslokal des Segelflugvereins Mannheim e.V. Flugplatz, Mannheim-Neuostheim.
7000 Stuttgart	Lokale Gruppe Stuttgart, Wolf-Helge Neumann ☎ 0711/882638 und Ulf Katzenmaier, ☎0711/268293.
8000 München	Heinz Schnitter, ☎ 089/3103385 oder Christoph Krinninger 089/7259382. Treffen jeden 4. Mittwoch im Monat 19 Uhr 30 im Vereinsraum 2 im Bürgerhaus Unterschleißheim am Rathausplatz (S-Bahnhaltepunkt S1 Unterschleißheim).
DDR-Leipzig	FORTH-Gruppe Leipzig, Michael Balig, Lützner Plan 17, DDR-7033 Leipzig oder Dr. Jürgen Hesse, Lieselotte-Herrmann-Str. 40, DDR-7050 Leipzig, ☎ 041-69 56 02

FORTH-Fachgruppen:

8000 München	RTX 2000 Gruppe, Koordinator Herr Krämer, 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:

3300 Braunschweig	Martin Holzapfel, Bassestr.17.
8500 Nürnberg 20	Thomas G. Bauer, Fichtestr. 31, ☎ 0911/538321.
5000 Köln 60	Michael Heycke, Boltensternstr.
4830 Gütersloh 1	Ludwig Röver, Holzheide 145A
5110 Alsdorf	Arndt Klingelnberg, ☎ 02404/61648, voraussichtlich zusammen mit der Computergruppe RWTH Aachen

Eine Fachgruppe will gründen:

7000 Stuttgart 80	Grafik/Arithmetik, Jörg Tomes, Anweilerweg 56, ☎ 0711/7802293.
8000 München 70	Btx u. FORTH, Christian Schwarz, Lindenschmitstr.30, 8000 München 70

Hier kann man um Rat fragen:

02103/556 09	Jörg Staben, Dienstag und Freitag, 20.00 - 22.00 Uhr
06187/91503	Frank Stüss
02845/28951	Karl Schroer
05221/23504	Andreas Findewirth, Im Großen Vorwerk 48, 4900 Herford

Ansprechpartner zu bestimmten Interessengebieten:

volksFORTH/ultraFORTH:	Klaus Kohl,	☎ 08233/30524
	Bernd Pennemann,	☎ 0228/640979
	Klaus Schleisiek-Kern,	☎ 040/2202539.
32-Bit Systeme:	Robert Jones,	☎ 02434/4579
Künstliche Intelligenz:	Ulrich Hoffmann,	☎ 0431/678850
NC4000 Novix Chip:	Klaus Schleisiek,	☎ 040/6449412
Realtime & Petri-Netze:	Wigand Gawenda,	☎ 040/446941
Gleitkomma-Arithmetik:	Andreas Döring,	☎ 02631/52786
32FORTH	Rainer Aumiller,	☎ 089/6708355
PostScript/FORTHscript	Christoph Krinninger,	☎ 089/725 93 82
FORTH im Unterricht	Rolf Kretzschmar,	☎02401/4390
Objekt-orientiertes FORTH	Christoph Krinninger,	☎089/725 93 82
	Ulrich Hoffmann,	☎0431/678850
Amiga - MULTI-FORTH	Rafael Deliano,	☎ 089/841 83 17
F-PC Zimmer FORTH, ASYST, Meßtechnik	Arndt Klingelnberg,	☎ 02404/61648, box:geo1:klingelnberg

FORTH-Gesellschaft e.V. - Postfach 1110 - D-8044 Unterschleißheim
☎ 089/3173784, FORTH-Mailbox ☎ 08841/5880
Postgiroamt Hamburg, Kontonr.: 563211-208 BLZ 20010020

Ergänzungen, Änderungen bitte dem Büro der FORTH-Gesellschaft e.V. mitteilen.

FORTH-SYSTEME

ANGELIKA FLESCH

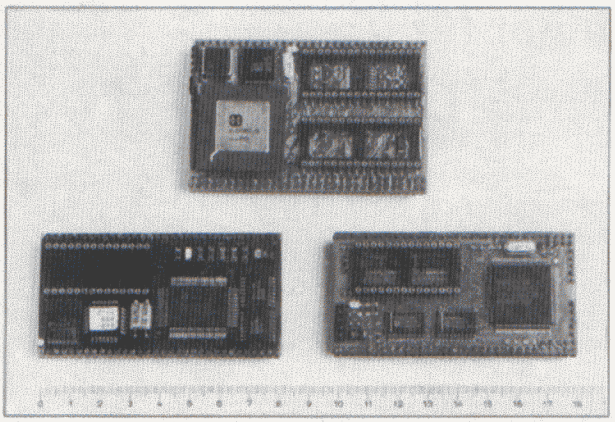
Postfach 1103
D-7814 Breisach

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

UR/FORTH

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

ModuNORM



CPU-Steck-Module im Scheckkartenformat:

- 8 Bit z. B. 6303
- 16 Bit z. B. V25
- Highspeed RTX-2000/1
- Softwareunterstützung durch SwissFORTH™
- Thermodrucker und Kontroller

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

LMI FORTH-83 Metacompiler

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

8086/8088	8096/97
Z80	HD64180
8080/8085	8031/32/535
68000	6303
Z8	6502
1802	V25
6809	68HC11
65816/65802	RTX 2000

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

Forth Programme, die mit einem LMI interaktiven Forth System z. B. PC/FORTH oder Z80 Forth geschrieben und getestet wurden, werden im Normalfall mit nur geringen Änderungen übersetzt.

Serieller ROM/RAM Simulator

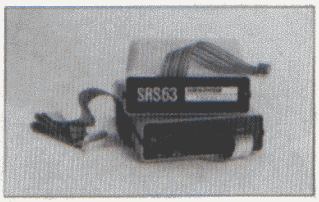
Entwickeln Sie romfähige Programme ?

Müssen Sie neu entwickelte Einplatinencomputer testen ?

Setzen Sie 2764, 27128, 27256, 27512 oder 4364, 43256 oder kompatible ROM/RAM-Bausteine ein ?

Wollen Sie diese Bausteine mit bis zu 38 400 Baud über die serielle Schnittstelle laden ?

Können Sie eine zusätzliche serielle Schnittstelle über den Speichersockel zum interaktiven Programmieren gebrauchen ?



Dann ist unser SRS63 die optimale Ergänzung Ihres Arbeitsplatzes.

Sie werden vom Preis-Leistungsverhältnis überrascht sein.

Unsere ROM-Compiler liefern direkt verwendbare Dateien, wir akzeptieren auch Intel-Hex oder Motorola-S-Formate.

mikrap

Mikrap
AG für Mikroelektronik-Applikation

CH-8840 Einsiedeln
Langrütistrasse 44
Postfach

Telefon 055 53 65 55
Telefax 055 53 61 23