VIERTE DIMENSION

7. Jahrgang Nr. 3 September1991

Echtzeit '91

Platine für den Super8

Optimierender Compiler

ANS Forth versus Forth-83

FORTH MAGAZIN

7,50 DM

FORTH-*Magazin*
Vierte Dimension

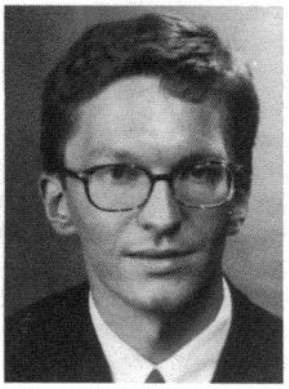
Nr. 3 September 1991

Inhaltsverzeichnis

Direktorium '91

MACH"2

Ich kam 1961 zur Welt und befinde mich seit frühester Jugend im Kampf mit Chips und Lötkolben. Nach harten Jahren des Studiums bin ich seit kurzem glücklicher



Christoph Krinninger

Zahnarzt und ein typischer Vertreter der Anwender und Mitglieder, die nicht vom Verkauf von FORTH leben. Meine unmittelbaren Interessen in Sachen FORTH gelten überwiegend Medizin-bezogenen Themen, wie z.B. Künstliche Intelligenz, Bildverarbeitung, CAD/CAM etc.

Seit cirka sechs Jahren ist FORTH meine ideale Programmierumgebung. Als Rechner bevorzuge ich den Macintosh, als FORTH verwende ich seit Jahren das MACH2, ein 32-bit FORTH für den Mac. Meine nächsten Arbeiten betreffen ein FORTH, das in Hypercard eingebunden ist und die dortige Sprache Hypertalk ersetzt.
Mitgliedschaft im MAC e.V.;

Hobbies
FORTH, Musizieren, Steppen.

comFORTH und FORTecH

Geboren 1948 in einem kleinem Dorf Bornitz bei Oschatz als waschechter Sachse.

Studium Elektroingenieurwesen (mit Algol 60) an der Rostocker Universität. Dann Assistenzzeit mit "Programmzeilen und Lochkarten" in Fortran. Jedoch die starre Reglementierung von Fortran ließ nach besseren Möglichkeiten der Programmierung suchen.

Ab 1979 übernahm ich die Aufgabe, Mikrorechner für automatische Steuerungen aufzubauen und stieß auf FORTH.

Dies war der Startschuß für die Rostocker FORTH-Aktivitäten: für verschiedene Projekte entstand comFORTH, das die Rostocker Uni-

Hartmut Pfüller

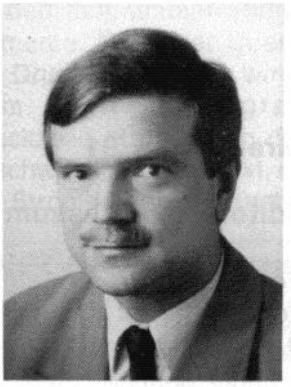
versität inzwischen an viele Betriebe, Hochschulen und Institute verkauft hat. 1990 gründeten wir die FORTecH Software GmbH, die mit comFORTH zweimal den Programmierwettbewerb bei der ECHTZEIT-Messe in Sindelfingen gewann.

Zur Zeit arbeite ich an der Technischen Universität Hamburg-Harburg im Arbeitsbereich Regelungstechnik.

Hobbies:
FORTH, Lesen, Flugmodelle

Dr. FORTH

Studium an der Technischen Universität in Dresden (Informationstechnik) - Assembler und ALGOL60. 1981 am Zentralinstitut für Kybernetik und Informationsprozesse der Akademie der Wissenschaften der DDR - inzwischen aufgelöst - hielt mich ein Freund davon ab, einen PASCAL-p-Code-Interpreter zu schreiben und gab mir statt dessen FORTH. Erster Schritt: Portieren des 8080-FIG-FORTH auf eine PDP11. Da wir im Institut kein Entwicklungssystem hatten und mit



Gert-Ulrich Vack

16 kByte Speicher auskommen mußten, wurden DDC-Algorithmen für Adaptivregler von nun an in FORTH programmiert. Und alle waren begeistert!

1984 mußte ein Berliner Mitarbeiter des Instituts aus einer Kollektion von Algorithmen in Assembler, PASCAL und FORTRAN ein vernünftiges industrielles Bildverarbeitungssystem realisieren. Wir kamen schnell dahinter, daß man das gar nicht anders als mit FORTH machen konnte. Mit objektorientierter Programmierung, overloading und generischen Datentypen wurde der Nachweis geführt, daß man auch in weniger als 20 kByte Schlüsselkonzepte von ADA realisieren kann. 1986 begann die Arbeit an einer FORTH-Monographie. 1989 erwarb ich den akademischen Titel "Dr. FORTH", und kurz vor der Währungsunion erschien das Buch in einer Auflage von 7500 Exemplaren.

Neben FORTH gibt es noch andere Dinge im Leben:
Meine Frau, zwei Töchter, unsere 1990 gegründete Elektronik-Vertriebsfirma, ein alter Ford Granada und ein Fahrrad.

IMPRESSUM

FORTH-*Magazin "Vierte Dimension"*

Herausgeber:
FORTH-Gesellschaft e.V.

Editor, Satz und Layout:
Peter Dinies, Metzstraße 38, W-2300 Kiel 1, Tel. 0431/1 32 39

Beirat:
Ulrich Hoffmann, Jens Storjohann, Michael Kalus

Illustrationen
Rolf Kretzschmar

Kontaktadresse:
FORTH-Büro, Postfach 1110, W-8044 Unterschleißheim, Tel. 0 89/3 17 37 84 oder FORTH-Mailbox, München, Tel. 0 89/871 45 48 8 N1 "Konferenz Vierte Dimension".

Quelltextservice:
Der Quelltext von Beiträgen, die mit dem Diskettensymbol gekennzeichnet sind, ist auf der Leserservice-Diskette zur jeweiligen Ausgabe oder in der FORTH-Mailbox zu finden.

Redaktionsschluß:
Erste Woche im mittleren Quartalsmonat
Erscheinungsweise vierteljährlich

Auflage:
Ca. 1.000

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

Preis:
Einzelheft DM 7,50, Abonnementpreis DM 40,-, bei Auslandsadresse DM 45,- inklusive Versandkosten

EDITORIAL

"Forth ist schneller als Assembler" so lautetet ein altes Gerücht aus der Pionierzeit von Forth. Dieses Gerücht geht auf einen Artikel von James Bell aus dem Jahre 1973 zurück, in dem die Realisierung von "direct threaded code" auf einer PDP-11 beschrieben wird. Die in diesem Artikel gemachte Aussage ist folgende:

Auf einer PDP-11 ist es schneller, Fragmente von Assemblercode durch NEXT, also JMP @(R)+, zu verbinden als durch Paare von JSR und RTS.

Diese Aussage gilt nicht nur für die PDP-11 sondern auch für zahlreiche der heute verwendeten Mikroprozessoren.

Der Artikel spricht aber nur über das Verknüpfen von 'Code'-Definitionen, nicht aber über das Abarbeiten von 'colon'-Definitionen, für die das gesagte eben nicht gilt.

Die Argumentation des Artikels wurde offensichtlich vielfach fehlverstanden und mündete schließlich in das obige Gerücht.

Natürlich ist Forth nicht schneller als Assemblercode. Da es aber i.a. in einer Anwendung nur wenige zeitkritische Stellen gibt, ist es möglich, durch Codierung dieser Bereiche in Assembler, sehr dicht an die Geschwindigkeit von reinen Assemblerprogrammen heranzukommen - das jedoch mit einem wesentlich geringeren Entwicklungsaufwand. Wie man diese zeitkritischen Stellen in Applikationen ermitteln kann, beschreibt Herr König in seinem Artikel über die Suche nach dem "Flaschenhals".

Forth muß sich in seinen Einsatzgebieten mit anderen Programmiersprachen messen lassen. Auf der Echtzeit'91 geschah das erfolgreich gegenüber anderen Echtzeitsprachen wie Ada, Pearl, C oder Modula 2, wie es meinem Bericht über die Echtzeit hoffentlich zu entnehmen ist.

Die Entwicklung bei der Implementierung anderer Programmiersprachen bleibt nicht stehen. Sehr viel Energie wurde in die Optimierung des von gängigen C-Compilern erzeugten Object-Codes investiert. Daß Forth sich entsprechenden Optimierungen nicht verschließt, zeigt der Artikel von Bernd Paysan über den optimierenden Compiler des bigFORTH-Systems.

Schließlich bleibt bei Forth immer noch die Möglichkeit, durch geeignete Hardware-Unterstützung dem Durchsatz nachzuhelfen. Der im Beitrag von Heinz Schnitter vorgestellte Microcontroller Super8 leistet genau dies. Er besitzt die Operationen NEXT, ENTER und EXIT (DTC) als Maschinenbefehle und kann daher Forth deutlich schneller ausführen als eine 8086, und das obwohl der Super8 nur eine 8-bit-CPU ist.

Ulrich Hoffmann

Vom 11. bis zum 13. Juni 1991 fand zum zweiten mal in der Sindelfinger Messehalle die Kongreß-Messe Echtzeit statt.

Ulrich Hoffmann

ECHTZEIT '91

FORTH-Teams an der Spitze

Unter den 83 Ausstellern fanden sich in diesem Jahr auch zahlreiche Firmen, die mit Forth arbeiten. Auf dem Stand der Forth-Gesellschaft war u.a. die Firma "Allinger und Partner" präsent und gab dort einen Überblick über die von ihr in der jüngsten Vergangenheit realisierten Forth-Projekte, von ganz klein bis reichlich groß. Ein Protokollkonverter für Siemens 3964R Datenübertragung war dort ebenso zu begutachten, wie auch ein RTX-2000 basiertes System für die Echtzeitverarbeitung und Auswertung von Meßwerten. Das Computer-Camp RCN aus Berlin machte wie schon auf der Jahrestagung in München forthige Echtzeit-Computermusik mit ihrem IDEAL 56 System (Motorola DSP 56001).

Bei DELTA t, die wie im vergangenen Jahr in Zusammenarbeit mit ETA Nürnberg ihren Stand gestalteten, gab es brandneue Informationen zum fränkischen Forth Risc Prozessor FRP 1600. Zum modularen RTX-2000 Echtzeitsteuerungs-System Muck wurde ein weiterer Peripherieadapter vorgestellt, der den Anschluß des Muck-Systems an den SCSI-Bus ermöglicht. Das Muck-System besteht damit aus den fünf Komponenten: "Prozessorplatine RTX 2000", "Interruptfähige Paralle I/O Karte", "Analogkarte mit 4 Ausgängen", "Analogkarte mit 4 Eingängen" und dem "SCSI-Adapter". Ebenfalls neu vorgestellt wurde eine neue Version des Methusalems, einem RTX-2000 basierten Gerät zur Langzeitdaten-Aufzeichnung via DAT-Recorder.

Die Rostocker Firma FORTecH, die Vorjahressieger des Programmierwettbewerbs (s.u.), stellte ihr Softwareentwicklungs-System comFORTH 2 vor, ein Forth-System, das unter CP/M, MS/DOS und UNIX eine einheitliche Programmieroberfläche liefert. Unterstützung von Fließpunktzahlen finden sich ebenso, wie automatische Glossargenerierung, Dokumentationsunterstützung und ausführliche On-Line-Dokumentation. Als besondere Spezialität wird ein KI-Paket angeboten, das es ermöglicht Methoden Wissens- und Regelbasierter Systeme für Anwendung im Echzeitbereich einzusetzen. Zur Programmierung von "embedded Systems" bietet FORTecH zwei Lösungen an. Das ist erstens der comFORTH 2 DownCompilers, mit dem sich Anwendungen interaktiv auf dem Zielsystem entwickeln lassen und zweitens der comFORTH 2 CrossCompiler, der zur Generierung ganz neuer Zielsysteme eingesetzt wird. Der DownCompiler wurde auf dem vom AUCOTEAM Berlin entwickelten ROM-Simulator ROMEO 2.0 demonstriert.

Auf dem Stand von Forth-Systeme Flesch ging es dann feucht her. Hier gab es das Bilderkennungssystem VISION-2000 zu bestaunen, das mit Hilfe einer Videokamera und wiederum einem RTX-2000 System den jeweils größten Fisch in einem Aquarium ermittelte. Für MS-WINDOWS Freunde gab es eine Vor-Beta-Version des LMI-Forth-for-Windows zu bewundern und auch die ModuNORM-Reihe ist wieder um einige Mitglieder größer geworden (vgl. letzte Umschlagseite).

Die Firma Reilhofer KG stellte gleich zwei ihrer Entwicklungen auf dem Gebiet der Online-Geräteüberwachung vor. Die On-Line-Analyse von Materialermüdung wird mit Hilfe der Geräte STRESS 2 und STRESS 8 vorgenommen. Die vom STRESS-Gerät aufgenommenen Ermüdungsdaten werden dort verarbeitet und zur graphischen 2D/3D-Darstellung der sog. Rainflow-Matrizen zum Host-Rechner (PC/AT) übertragen. Durch den Einsatz von STRESS können die Erneuerungsintervalle von Verschleißteilen wesentlich genauer bestimmt werden, als das mit herkömmlichen Methoden möglich ist. Das ist insbesondere im (Schwer-)Maschinenbau von großer Bedeutung. Das zweite Gerät, der delta-Analyzer, zielt auf ähnliche Anwendungsgebiete. Er stellt charakteristische Schwingverhalten etwa von Fertigungsmaschinen fest und ist in der Lage Abweichungen von diesem Schwingverhalten zu registrieren, aufzubereiten, graphisch darzustellen und sie einem Host-Rechner zu signalisieren.

Von der Dr. Weiß GmbH wurde die MAKmodul Palette vorgestellt. MAKmodule sind Baugruppen mit

einem geringen Formfaktor, die die Konfiguration modularer Mess-Steuer- und Automatisierungsrechner ermöglichen. Der Kern des MAK-Systems ist die Zentraleinheit CPU+, in der auf Basis des Motorola 6809 Mikroprozessors ein Multitasking Forth-System die benötigten Systemfunktionen für die alle restlichen MAKmodule liefert. Eine 68030 Zentraleinheit befindet sich in der Entwicklung. Aus den knapp einhundert MAKmodul-Einzelkomponenten sind schon etwa 170 MAKmodule fertig konfiguriert, neue MAKmodule können aber auch individuell zusammengestellt werden. Unter den Einzelkomponenten finden sich Karten zum Aufbau der Zentraleinheit, außerdem Massenspeicherkomponenten, wie Floppy- und Harddisk-Controller und RAM-Floppies, Karten zur digitalen und analogen Ein- und Ausgabe in zahlreichen Variationen und auch Systemkarten, wie Farbgraphik-Controller und Prom-Programmer. Für Anwendungen in der Meßtechnik findet man ein weites Spektrum an Meßwertaufnehmern (Wegaufnehmer, Dehnungsmessung, Thermoaufnehmer, u.a.), Schrittmotorsteuerungen, PID-Regler, analoge Endstufen u.ä.

Der Kongreß

Auf dem gleichzeitig stattfindenden Echtzeit-'91-Kongreß mußten die rund 400 Kongreßteilnehmer die schwere Auswahl zwischen den über 60 Seminarvorträgen treffen, die meist in zwei Sitzungen nebeneinander vorgetragen wurden. In 4 der 19 Vortragsblöcke wurde intensiv über Forth und über Forth-Anwendungen berichtet.

Im Teil "Programmiersysteme" berichtete Herr Heinz von der Synics AG über einen von ihnen realisierten Interpreter der Sprache GRAFCET (ähnlich Kontaktplan) ("GRAFCET Interpreter in Forth") der sich nahtlos in Forth-Programmierumgebungen einbettet. Herr Woitzel von der Universität Rostock schilderte die dortigen Bemühungen zur Formulierung "Portabler Multitasking-Erweiterungen für Forth".

Während der Sitzung "Anwendungen mit verteilten Systemen" trug Herr Kabitzsch über die "Programmierung verteilter Systeme für die landtechnische Automatisierung" vor und erläuterte dabei ein Konzept bei dem die Programmentwicklung für "embedded" Applikation durch ein Zusammenspiel von Petri-Netzen und Forth vorgenommen wird. Herr Kurfürst von der Marco GmbH erläuterte die Realisierung von "Forth83 Multitasking auf gekoppelten Rechnern im Bergbau", wobei es insbesondere auf die Fehlertolleranz des Systems ankommt.

Im Block "Anwendungen mit Forth-Systemen" berichtete Herr Schleisiek-Kern von der DELTA t GmbH über das System Methusalem ("Vier Mikrosekunden, ein Stackprozessor und Forth",s.o.) und die bei der Programmentwicklung dieses Systems auftretenden prinzipiellen Schwierigkeiten. Herr Flesch (Forth Systeme) stellte die Konzeption des VISION-2000 Systems ("Echtzeit Bildverarbeitung mit dem RTX-2000",s.o.) vor. Herr Clemens von der Uranit GmbH erläuterte, in welcher Weise Forth bei der Steuerung von Industrie-Lasern ("Steuerung gepulster Laser mit einem Echtzeit-FORTH-System") eingesetzt wird.

Unter der Rubrik "Regelungstechnische Fallstudien" berichteten Frau Nikolova und Herr Hristozov von der Bulgarischen Akademie der Wissenschaften über "Eine optimierte Lösung zur Steuerung von Montagerobotern", die in Forth und Assembler realisiert wurde.

Konferenz-Bände, in denen ausführliche Ausarbeitungen der einzelnen Vorträge enthalten sind, sind noch über den Veranstalter Kongreß-Agentur Drebinger in München erhältlich.

Der Wettbewerb

Wie schon im vergangenen Jahr fand auch 1991 auf der Echtzeitmesse der Programmierwettbewerb statt, bei dem unter den angetretenen Teams dasjenige ermittelt wird, das eine gegebene Aufgabe am schnellsten zu lösen vermag. In diesem Jahr galt es den "LABGNIRBS" (Springball rückwärts) zu programmieren. Zu diesem Zweck wurden unter Federführung von Prof. Gehrt (Uni Hannover) zehn Modelle eines Springball-Versuchs in präziser Feinarbeit angefertigt.Auf einer senkrechten Führungsstange konnte ein Hohlzylinder (der Springball) über eine Feder und ein Pleuel durch einen Motor in Schwingungen versetzt werden. Ziel war es nun, die Schwingungen des Hohlzylinders, also des Springballs, programmiert unter Kontrolle zu bekommen und in einem vorgegebenen Muster schwingen zu lassen. Nachdem das geforderte Verhalten durch einen 68008 EMUF (RTOS/UH, PEARL) demonstriert worden war, hieß es für die zehn teilnehmenden Teams: "Auf die Plätze, fertig, los!". Nach ca. zwei Stunden hatte - wie im vergangenen Jahr - die Firma FORTecH aus Rostock mit ihrem comFORTH-System die Aufgabe gemeistert. Nur knapp fünf Minuten später folgte das Team von DELTA t, die ihr Muck-System und das muckFORTH eingesetzt hatten. Keines der anderen angetretenen Teams war in der Lage, die gestellte Aufgabe innerhalb der vier zur Verfügung stehenden Stunden noch zu lösen, sodaß die weiteren Platzierungen davon bestimmt war, in wie weit das Modell beherrscht wurden (Ball springt hoch, Pleuel in bestimmte Position fahren, Motor drehen ...). Das Wettbewerbsergebnis 1991 spiegelt damit das Resultat von 1990 wieder: Forth Teams liegen an der Spitze und lösen die ihnen gestellte Aufgabe in kurzer Zeit - die anderen Teams können sie nicht innerhalb der Bearbeitungsfrist lösen.

Alles in Allem war die Echtzeit'91 (nicht nur) für FORTH ein voller Erfolg. Die Anzahl sowohl der Forth-Aussteller als auch der Forth-Vorträge ist enorm angewachsen.

Die Echtzeit'91 macht klar, das Forth auf dem Gebiet der Echtzeitprogrammierung ein Werkzeug ist, das von vielen Entwicklern genutzt wird, und das seinen Platz neben den anderen Echtzeitprogrammiersprachen gefunden hat.

Leserbriefe

Der schmucke Schwan oder das häßliche Entlein

Das perfekte Layout gibt es nicht. Niemand hat ein Erfolgsrezept in der Tasche. Trotzdem muß ich sagen das sich das Layout zum schlechten gewandelt hat. Es sieht ein bißchen lieblos dahingezogen und unübersichtlich in seiner jetzigen Form aus. Nicht nur der Inhalt einer Zeitung ist entscheidend, sondern auch ihre Aufmachung. Schließlich will doch jeder sein Essen am liebsten auf einem silbernen Tablett serviert bekommen.

Die Form und das Aussehen einer Vereinszeitschrift sollte dem Charakter des Vereins entsprechen und natürlich dem Charakter der Sprache FORTH: Jung, dynamisch, lebendig. Niemals sollte man auf ein Minimum an Gestaltungsaufwand zurückfahren, das Ergebnis ist - wie man sieht - ein sehr steriles Aussehen das auf den ersten Blick schon abschreckend wirkt. 'Seriöses Aussehen' hat nichts mit absolut fadem Einheitsdesign zu tun (mit solchen Denkweisen kommt man nicht weit wie man in den letzten Jahren gesehen hat). Es sollte stets experimentiert werden um Artikel für den Leser noch ansprechender und übersichtlicher zu gestalten. Dazu sollte man kein Mittel der heutigen modernen Technik auslassen, schließlich sind wir doch im Postscript Zeitalter und leben - wie mein Geschichtslehrer ab und an zu sagen pflegt - in der Welt der Designer. Das dies sehr gut klappt sieht man in der fetzigen und poppigen Aufmachung heutiger Computerzeitschriften. Die populären Computerzeitschriften müssen natürlich mit tollem Aussehen hervorstechen, denn sie sollen natürlich beim Zeitschriftenladen ins Auge stechen. Das hat eine Vereinszeitung natürlich nicht nötig. Wenn man aber sich das Ziel gesetzt hat, die Programmiertechniken der Sprache FORTH an die breite Öffentlichkeit zu bringen, kann man sich nicht auf solche Art und Weise von der Außenwelt abnabeln und sein eigenes Süppchen kochen.

Also können wir nur darauf hoffen, daß unser häßliches Entlein doch noch zu einem schmucken Schwan wird, der sich in die Lüfte erhebt um die frohe Kunde vom FORTH weiterzuverbreiten!

Jens Wilke

Nachfolgemodell eines Rechners für Apple 2 und IBM PC XT und C 64

Die Grundmodelle der heute noch erfolgreichen Computer C 64, Apple 2 e, und die MS Dos XT kompatiblen erblickten vor ca. zehn Jahren das Licht dieser Welt. Nun möchte ich diese Modelle langfristig durch neuentwickelte Computer mit CPU's aus der 680X0 Familie ablösen.
Als erstes wäre ein Tastaturcomputer mit der CPU 68008 1 Mbyte Version zu nennen. Er ist für Homecomputeranwendungen und Spiele konzipiert, und soll die Nachfolge des C 64 und des CPC antreten. Gesucht werden für diesen Computer Programmierer, die Systemsoftware und Spiele auf eigene Rechnung programmieren und Leute, die aktiv an der Entwicklung, Herstellung und Verkauf mitarbeiten.
Als zweites wäre ein Slotcomputer mit DIN-Steckern und der CPU 68008 4 Mybte Version für gehobene Homecomputeranwendungen und kleine Büroanwendungen sowie Labortechnik, Meß- und Regeltechnik zu nennen. Als Betriebssystem wäre RTOS Pearl eventuell zu nennen. Ebenfalls für diesen Computer werden System- und Anwendungsprogrammierer gesucht und Leute die aktiv an der Entwicklung, Fertigung und Verkauf mitarbeiten.
Als drittes wäre zu nennen, das größere Computer aus VME Bus Steckkarten zusammengesteckt werden und für den Anwendungsfall maßgeschneidert werden. Hier suchen wir ebenfalls Leute, die ein solches System konfigurieren können.

Als Programmiersprache kann Assembler oder FORTH oder C oder Modula 2 oder eine Mischung herhalten.
Leute, die Interesse haben an einem solchen System mit zu wirken sollen sich bei Braun Ludwig jun. Postfach 1236, 8425 Neustadt/Donau melden.

P.S:
Eventuell kann auch der FRP 1600 Verwendung finden. Kommt auf den Preis an.

---

eFORTH

Liebe FORTH-Freunde,

in der FORTH-Dimension Volume XIII/1 fand ich eine Beschreibung des eFORTH-Modells. Dieses Modell ist leicht zu portieren und speziell für Embedded-Controller gedacht.
Bei mir besteht großes Interesse dieses FORTH an den Motorola 68hc11 anzupassen. Daher meine Frage an die Mitglieder der FORTH-Gesellschaft: Wer hat ein eFORTH für den 8086 oder 8051 als Quelltext und kann mir diesen zur Verfügung stellen?
In dem FORTH-Dimension Artikel wird auf die verschiedenen Implementationen des Fig-Forth hingewiesen. Vielleicht ist eFORTH das fortschrittliche Modell für Implementationen auf kleinen Systemen. Hieran haben sicherlich auch die Mitglieder der FORTH-Gesellschaft e.V. Interesse.

Jan Michaels
Postfach 30 38 51
1000 Berlin 30
Tel. 324 60 67

IBM Cad

IBM hat im PC-CAD-Bereich vor kurzem das System IBM Cad vorgestellt, das in Konkurrenz zum Marktführer AUTOCAD tritt. IBM Cad ist ein 2D/3D Entwurfs-System, das einen zu AUTOCAD vergleichbaren Befehlsumfang besitzt, aber nur etwa ein Drittel kostet. Es ist in zwei Versionen erhältlich: IBM Cad 1.2 und IBM Cad Plus. IBM Cad Plus besitzt dabei die Möglichkeit, über die dem Benutzer zugängliche Programmiersprache CadForth erweitert und modifiziert zu werden. IBM Cad Plus wird dadurch zur "Graphic Engine", die in eigene Applikationen eingebunden werden kann.

Reverse Polish Lisp

Hewlett Packard (HP) bekannt für seine umgekehrt polnischen Taschenrechner benutzt seit geraumer Zeit für die Implementierung der Taschenrechner-Betriebssysteme eine Forthähnliche Programmiersprache, die inoffiziell als RPL (Reverse Polish Lisp) gehandelt wird. HP-Taschenrechner wie HP-17B, HP-19B, HP-27S, HP-28C und HP28S basieren auf dieser Sprache. Auch HP's jüngstes Kind, der HP-48 ist in RPL programmiert und es erreicht dort sogar die Benutzerebene. Um HP-48-Informationen und -Quellcode auszutauschen sind vor Kurzem im USENET die beiden Newsgroups comp.sys.hp48 und comp.sources.hp48 gegründet worden. Wie in comp.lang.forth findet man dort forthige Programme, z.B. statistische und numerische Pakete. Diese neuen Newsgroups sind sicher einen Blick wert.

Eine Übersicht von RPL findet sich in "RPL: A Mathematical Control Language", W.C.Wickles, Proceedings of the 1988 Rochester Forth Conference, Seiten 27-32, Institute of Applied Research, Inc., Rochester 1988

weitere Nachricht auf S. 21

Kleinanzeige

Super8™ -FORTH in Silicon

- FORTH im 8Kbyte ON-Chip ROM
- Transparente Entwicklungsumgebung kompiliert ROM-fähigen Code
- Zugrifff auf alle ON-Chip Resourcen
- ausführliches Handbuch
- Quellcode von S8-Assembler und S8-FORTH

Muster 0887520PSC bei:
FORTHWORKS
Ulrike Schnitter
Tel: 089-3103385

Kleinanzeige:

Die Platine zum Super8 FORTH-Prozessor gibt's von:

Hans-Günther Willers
Gartenstraße 11
8047 Karlsfeld
Tel.: 08131 - 96375

Kleinanzeige:

Wer kann Englisch in Wort und Sprache, um für mich Verhandlungen mit amerikanischen und Fernostfirmen zu führen. Faxgerät vonnöten. Ludwig Braun jun. Postfach 1236, Schwaiger Str. 4, 8425 Neustadt/Donau.

Nachrichten und Klein-Anzeigen

Wo ist der Flaschenhals??

H. König

Bei größeren Programmen ist es oft schwer festzustellen, welche Programmteile das schlechte Laufzeitverhalten bewirken und deshalb in Assembler zu kodieren sind. Als Abhilfe wird eine Methode vorgestellt, mittels derer man feststellen kann, wie häufig ein Forthwort beim Ablauf einer Software aufgerufen wird.

Schlüsselworte:

Aufrufhäufigkeit, Dynamische Softwareanalyse, Utility

Wenn die Kontrollstrukturen eines Programmes einigermaßen komplex sind und von äußeren Einflüssen oder gar von den Wünschen der Benutzer abhängig sind, ist es oft schwer zu sagen, welche Teile des Programmes denn nun der berühmte Flaschenhals sind, den es zu optimieren gilt.

Nachdem ich mehrfach Programmteile in Assembler codiert habe, nur um festzustellen, daß danach das Programm um zwei Prozent schneller ablief, kam mir eine erstaunlich einfache Idee, wie man herausfinden kann, welche Teile des Programmes am häufigsten aufgerufen werden.

Der Grundgedanke ist, daß jedes Wort, welches man testen möchte, bei seinem Aufruf einen Zähler incrementiert. Diese Zähler sollten dann zusammen mit den Wortnamen ausgedruckt werden. Weiterhin ist bei Microcontrollern oder 64K-Systemen der Speicherplatz knapp, also sollte das Ganze in dieser Hinsicht flexibel sein. Dann fiel mir noch ein, daß ich keine Lust hatte, nach der Optimierung alle diese Zähler wieder mühsam aus dem Quelltext zu entfernen und sie bei Programmänderungen noch mühsamer alle wieder einzubauen.

Die Realisierung sieht folgendermaßen aus:

Es wird ein Array angelegt, das für jedes zu testende Wort die CFA und einen 32-Bit Zähler enthält, also für jeden implementierten Zähler sechs Byte. Die ganze Arbeit erledigt ein Wort namens CountIt, dieses muß in der Definition jedes zu testenden Wortes mindestens einmal aufgerufen werden. Das Wort CountIt ist IMMEDIATE, trägt beim Kompilieren in das nächste freie Element des Arrays die CFA des gerade kompilierten Wortes ein und erweitert die Definition des gerade kompilierten Wortes um einige Befehle, welche zur Laufzeit dieses Wortes den Zähler in dem Array incrementieren. Wird CountIt mehrfach in der Definition eines Wortes aufgerufen, so werden mehrere Zähler für dieses Wort angelegt. Auf diese Weise kann man Einblick in einzelne Zweige von Kontrollstrukturen nehmen.

CountIt, beim Kompilieren

1- CFA des Wortes feststellen, welches gerade definiert wird

2- diese CFA in den nächsten freien Platz des Arrays eintragen

3- Adresse des Zählers in diesem Arrayelement bestimmen

4- diese Adresse als LITERAL kompilieren

5- die Sequenz "2@ 1. D+ addr 2!" kompilieren; dabei ist addr wieder die Zähleradresse

6- Zeiger auf das Array incrementieren

CountIt, zur Laufzeit

Da CountIt IMMEDIATE ist, hat es kein Laufzeitverhalten, es läuft aber immer die von CountIt kompilierte Sequenz "addr 2@ 1. D+ addr 2!" ab, welche den zu dem betreffenden Wort gehörenden Zähler incrementiert.

Wird nun das Gesamtprogramm aufgerufen, werden alle Worte, deren Definition das Wort CountIt enthält, fleißig ihre zugehörigen Zähler incrementieren. Das kostet natürlich Zeit und wird den Programmablauf erstmal heftig bremsen. Ist das Programm abgelaufen oder abgebrochen worden, kann man anhand des Arrays feststellen, welches Wort wie häufig aufgerufen wurde.

In dem abgedruckten Listing wird diese Aufgabe sehr primitiv durch das Wort ShowCounts gelöst. Ich leite die Ausgabe dieses Wortes meistens auf den Drucker um und modifiziere das zu analysierende Programm so, daß ShowCounts durch eine spezielle Taste aufgerufen wird.

Mit diesen Ergebnissen ausgerüstet kann man dann die eigenen Assemblerkenntnisse an den erfolgversprechenden Stellen bemühen.

Jetzt stellt sich noch die Frage, wie man die gerufenen Geister möglichst elegant wieder los wird, bevor man sein Werk als Turnkeyapplikation verkauft.

Die Lösung ist eine Neudefinition von CountIt und ein erneutes Kompilieren des Programmes.

```
: CountIt NOOP ; IMMEDIATE
```

Dieses ist die erste Routine von mir, die nichts tut, das aber sofort, in der Applikation kein Byte Speicher und keine Mikrosekunde Laufzeit vergeudet und trotzdem nützlich und allgemein anwendbar ist.

Bleibt noch zu sagen, daß ich mit LMI-UR/FORTH arbeite, das Array außerhalb des Forthsystems verwalte und deshalb keinen Arrayüberlauf abfange, da das Array Platz für 10000 Zähler hat.

Das Wort PrintName muß für alle anderen Forth-Systeme angepaßt werden, UR/FORTH hält Header, Code und Daten in getrennten 64K-Segmenten. PrintName muß aus der CFA eines Wortes das Count-Byte des Namens dieses Wortes ermitteln und den Namen anzeigen.

Das Bestimmen der CFA des gerade kompilierten Wortes ist wohl auch systemspezifisch, bei UR/FORTH macht dies die Sequenz "LAST @ NAME>", ich glaube das dem unter FIG-FORTH die Sequenz "LATEST CFA" entspricht.

Es wäre wohl sinnvoller, direkt die NFA zu benutzen, hier hat die alte Gewohnheit zugeschlagen.

Abgedruckt sind zwei Versionen des Quelltextes, die Version Counter1 legt das Array im Dictionary an, ist deshalb schneller und leichter auf andere Systeme zu übertragen.

Wenn 16-Bit Zähler reichen, kann man von CountIt die Sequenz "1 adr +!" kompilieren lassen, das spart Zeit und Code; wer Echtzeitsysteme programmiert, kann nur die Adresse und den Aufruf einer Assemblerroutine kompilieren lassen, welche dann den Zähler incrementiert.

```
Screen # 0

\ Counter für Performance-Tests              HK  16:37 21.07.91
\ dient dazu festzustellen, wie oft ein Wort aufgerufen wird.
\ Einsatz: muß innerhalb der Definition, deren Aufrufhäufigkeit
\ festgestellt werden soll einmal aufgerufen werden.
\ Funktion:
\ Es wird ein Array angelegt, dessen Elemente 6 Bytes sind,
\ 2 Byte CFA, 4 Byte Häufigkeit des Aufrufes.
\ zur Compilierzeit wird die CFA des gerade compilierten Wortes
\ bestimmt und in das Array eingetragen, die Häufigkeit wird
\ initialisiert,und in die aktuelle Definition wird eine Sequenz
\ kompiliert, die die Häufigkeit incrementiert.
\ Weiterhin wird ein Zeiger auf das Array verwaltet.
\ Für Große Programme wird das Array im DOS-RAM verwaltet
\ !!!! Dieses hat den Nachteil, nicht in TURNKEYS verwendbar
\ zu sein, da der Speicher für das Array sowohl beim compilieren
\ als auch bei der Ausführung bekannt sein muß.




Screen # 1

\ Counter für Performance-Tests              HK  20:40 21.07.91
FORGET TASK : TASK ;
-->
\ Dieser Nachteil entfällt, wenn das Array im Dictionary an
\ gelegt wird, sofern man nur einige Worte auszählen will,
\ kostet das auch nicht zuviel Speicher.
\ Dazu müssen !CFA, GetCounter und in CountIt 2@L und 2!L
\ angepaßt werden.
\ Außerdem müssen für ShowCounts die Header verfügbar sein,
\ damit ' funktioniert.
```

```
Screen # 2

\ Memory-Allocation                                HK  17:26 21.07.91
1000 CONSTANT COUNTERS
6 CONSTANT COUNTERSIZE
2VARIABLE COUNTARRAY
VARIABLE ARRAYINDEX 0 ARRAYINDEX !
: GetArray COUNTERS COUNTERSIZE * MALLOC COUNTARRAY 2! ;

: !CFA   ( CFA  --- )
COUNTARRAY 2@ ARRAYINDEX @ 6 * + !L ;

: GetCounter ( --- ptr )
COUNTARRAY 2@ ARRAYINDEX @ 6 * 2+ + ;

: @CFA   ( Index --- CFA )
>R COUNTARRAY 2@ R> 6 * + @L ;
-->



Screen # 3

\ Memory-Allocation                                HK  18:36 21.07.91

: ClearArray GetArray COUNTERS COUNTERSIZE * 0
 DO 0 COUNTARRAY 2@ I + C!L LOOP ;



: @Counter  ( Index --- DCounter )
>R COUNTARRAY 2@ R> 6 * + 2+ 2@L ;

ClearArray
-->



Screen # 4

\ CountIt                                          HK  18:36 21.07.91
\ LAST holt die NFA der gerade compilierten COLON-DEF, das
\ kann auch LATEST heißen
\ NAME> convertiert eine NFA zur CFA heißt oft CFA oder >CFA
: CountIt
STATE  @  IF
LAST @ NAME> !CFA
\ CFA in COUNTARRAY untergebracht
GetCounter [COMPILE] DLITERAL \ compiliert die aktuelle CntAdr
COMPILE 2@L 1 0 [COMPILE] DLITERAL COMPILE D+
GetCounter [COMPILE] DLITERAL COMPILE 2!L
\ Compiliert wird ptr 2@L 1. D+ ptr 2!L ; ptr wird zur
\ Compilierzeit berechnet
1 ARRAYINDEX +!
ELSE CR ." Compilation only " CR
THEN ; IMMEDIATE  -->
```

```
Screen # 5

\ PrintName                                          HK  18:51 21.07.91
( cfa --- ; display name field; clamping to 20 characters )
: PrintName PAD 20 BLANK
            DUP DUP >NAME NAME> =
            IF  >NAME DUP 1+ HS0 ROT C@L >R
                HS0 SWAP DS0 PAD R@ CMOVEL
                PAD R> 31 AND DROP 20 TYPE
            ELSE DROP ." ???" THEN ;
-->
\ Dieses ist aus Debug.scr geklaut, und bezieht sich auf
\ LMI-UR/FORTH, welches die Header in einem eigenen Segment
\ hält, in Standardsystemen mit 64K Adreßraum für alles geht
\ das Viel einfacher. LAST oder LATEST liefert hier direkt die
\ NFA, diese wird in COUNTARRAY abgespeichert, PrintName muß
\ nur noch den Count herausziehen und den Namen an TYPE über-
\ geben.



Screen # 6

\ Auswertung                                         HK  18:49 21.07.91

: ShowCounts
  COUNTERS 0
  DO
    I @CFA 0= IF LEAVE THEN
    CR I @CFA PrintName I @Counter 12 D.R
  LOOP ;

-->



Screen # 7

\ TESTS                                              HK  18:00 21.07.91
: TEST1
CountIt 5 6 + DROP ;
: TEST2
CountIt 10 0 DO TEST1 LOOP ;
: TEST CountIt TEST1 TEST1 TEST2 TEST2 ;
```

```
Screen # 0

\ Counter Version f. 64K Forth                HK  20:08 21.07.91
\ dient dazu festzustellen, wie oft ein Wort aufgerufen wird.
\ Einsatz: muß innerhalb der Definition, deren Aufrufhäufigkeit
\ festgestellt werden soll einmal aufgerufen werden.
\ Funktion:
\ Es wird ein Array angelegt, dessen Elemente 6 Bytes sind,
\ 2 Byte CFA, 4 Byte Häufigkeit des Aufrufes.
\ zur Compilierzeit wird die CFA des gerade compilierten Wortes
\ bestimmt und in das Array eingetragen, die Häufigkeit wird
\ initialisiert,und in die aktuelle Definition wird eine Sequenz
\ kompiliert, die die Häufigkeit incrementiert.
\ Weiterhin wird ein Zeiger auf das Array verwaltet.




Screen # 1

\ Counter für Performance-Tests               HK  20:39 21.07.91
FORGET TASK : TASK ;
-->
\ Dieses ist eine Marotte von mir, liegt an meiner
\ Entwicklungsumgebung









Screen # 2

\ Memory-Allocation                           HK  18:33 27.07.91
100 CONSTANT COUNTERS     \ Anzahl möglicher Zähler
6 CONSTANT COUNTERSIZE    \ 6 Byte pro Zähler
VARIABLE COUNTARRAY       \ hier gibt's Platz dafür
COUNTERS COUNTERSIZE * ALLOT
VARIABLE ARRAYINDEX 0 ARRAYINDEX !

: !CFA    ( CFA  --- )          \ trägt aktuelle CFA ein
COUNTARRAY ARRAYINDEX @ 6 * + ! ;

: GetCounter ( --- ADR )        \ liefert aktuelle CounterAdr
COUNTARRAY ARRAYINDEX @ 6 * 2+ + ;

: @CFA    ( Index --- CFA )     \ holt CFA aus Array
>R COUNTARRAY R> 6 * + @ ;
-->
```

```
Screen # 3

\ Memory-Allocation                              HK  18:33 27.07.91

: ClearArray
  COUNTARRAY  COUNTERS COUNTERSIZE * ERASE ;

\ Hier fehlt ein Wort, welches nur die Zähler löscht.

: @Counter  ( Index --- DCounter ) \ holt Zähler aus Array
>R COUNTARRAY R> 6 * + 2+ 2@ ;

ClearArray
-->




Screen # 4

\ CountIt                                        HK  20:15 21.07.91
\ LAST holt die NFA der gerade compilierten COLON-DEF, das
\ kann auch LATEST heißen
\ NAME convertiert eine NFA zur CFA heißt oft CFA oder CFA
: CountIt
  STATE  @  IF
  LAST @ NAME> !CFA
\ CFA in COUNTARRAY untergebracht
  GetCounter [COMPILE] LITERAL \ compiliert die aktuelle CntAdr
  COMPILE 2@ 1 0 [COMPILE] DLITERAL COMPILE D+
  GetCounter [COMPILE] LITERAL COMPILE 2!
\ Compiliert wird adr 2@ 1. D+ adr 2! ; adr wird zur
\ Compilierzeit berechnet
  1 ARRAYINDEX +!      \ Platz für den nächsten
  ELSE CR ." Compilation only " CR
  THEN ; IMMEDIATE  -->



Screen # 5

\ Auswertung der Ergebnisse                      HK  20:06 21.07.91
( cfa --- ; display name field; clamping to 20 characters )
: PrintName PAD 20 BLANK
            DUP DUP >NAME NAME> =
            IF  >NAME DUP 1+ HS0 ROT C@L >R
                HS0 SWAP DS0 PAD R@ CMOVEL
                PAD R> 31 AND DROP 20 TYPE
            ELSE DROP ." ???" THEN ;
-->
\ Dieses ist aus Debug.scr geklaut, und bezieht sich auf
\ LMI-UR/FORTH, welches die Header in einem eigenen Segment
\ hält, in Standardsystemen mit 64K Adreßraum für alles geht
\ das Viel einfacher. LAST oder LATEST liefert hier direkt die
\ NFA, diese wird in COUNTARRAY abgespeichert, PrintName muß
\ nur noch den Count herausziehen und den Namen an TYPE über-
\ geben.
```

```
Screen # 6

\ Auswertung der Ergebnisse                    HK  18:36 27.07.91

: ShowCounts
  COUNTERS 0
  DO
    I @CFA 0= IF LEAVE THEN
    CR I @CFA PrintName I @Counter 12 D.R
  LOOP ;

: PrintCounts
  PRINTER ShowCounts CONSOLE ;
-->
\ Das ist der Vorteil von vektorisierter IO


Screen # 7

\ TESTS                                        HK  20:06 21.07.91
: TEST1
CountIt 5 6 + DROP ;
: TEST2
CountIt 10 0 DO TEST1 LOOP ;
: TEST CountIt TEST1 TEST1 TEST2 TEST2 ;
```

Vergleich

'ANS forth Core Word Set' mit 'Forth-83 Required Word Set'

Michael Kalus

Dieser Beitrag ist eine leicht gekürzte Übersetzung des Anhangs F aus dem Dokument BASIS15. Darin werden die Änderungen geschildert, die auf Grund der Unzulänglichkeiten des Forth-83 fällig wurden. Es werden also weitere Unterschiede des ANS zum Forth-83 besprochen. Denn ANS Forth wird eine Reihe neuer Eigenschaften enthalten. Eine Übersicht darüber habe ich im letzten Heft gegeben (VD 2/91). Die wichtigsten Änderungen des 'core word set' wurden ebenfalls dort gezeigt. Die folgenden Abschnitte beschreiben, warum ersetzt oder ergänzt oder verworfen wurde und in wieweit sich das Verhalten von Forth dadurch wandeln wird.

Einleitung

Ein Ziel des ANS FORTH soll es sein, Forth auf möglichst vielen Maschinen einsetzbar zu machen. Die 'virtuelle Maschine' des Forth-83 war präzise für 16-Bit Datenoperationen in hintereinanderliegenden 8-Bit Speichersegmenten geschaffen. Andere Prozessoren hatten Probleme mit Forth-83. Diese 16-Bit Vorgaben sind so tief im '83 Standard eingegraben, daß eine automatische Portierung auf andere Systeme nicht möglich ist. So ist es z.B. nicht ohne weiteres klar, ob 2+ eine Adressenzelle weiterschieben oder einfach Zwei zu einer Zahl hinzuzählte sollte. Ein '83iger Programm ist sozusagen ein ANS Programm mit einer 16-Bit Umgebungsbeschränkung.

Die nun versuchte generalisierte Fassung eines Standards bedeutet aber nicht, das ältere Programme damit wertlos würden. Sie arbeiten ja weiterhin - wenn auch nur in ihrer angestammten Umgebung. Die Chance bei einer Umstellung auf ANS Forth wird darin liegen, zum Beispiel alle 2+ auf cell+ zu ändern und so neue Gebiete für alte Programme zu eröffnen.

Hinzugekommen

Neben den schon besprochenen wichtigen Verbesserungen im 'core word set' gibt es noch einige kleinere Ergänzungen. Die Grundoperationen werden dadurch komplett:

```
2drop 2dup 2over 2swap 2!
2@ 2* c@ c! c,
```

Die Bedeutungen sind wohl allgemein bekannt unter Forthlern. Durch diese Worte wird der Wortschatz abgerundet. Damit hat sich die vorangestellte "Zwei" als Kennzeichen für "doppelte Wortbreite" wohl durchgesetzt.

Ersatz und Streichungen

Ergänzungen bereiten in der Regel im Forth keine Probleme, sie werden dazu geladen. Streichungen aus dem alten Standard hingegen würden bewirken, daß Programme unter ANS Forth nicht mehr ablaufen. Solche Streichungen bedürfen daher aller größter Vorsicht. Viele der neuen "Streichungen" sind bei näherem Hinsehen reine Verschiebungen in andere 'word sets' und daher ohne besondere Bedeutung. So wanderten z.B. die Blockworte in das 'Block Extension Word Set' oder FORTH in das 'Search Oder Word Set'. Aber einige inzwischen obsolete Worte wurden tatsächlich entfernt. So wurde 2+ gegen CELL+ ausgetauscht, kann aber leicht nachgemacht werden, um wieder Forth-83 Kompatibilität zu bekommen.

Worte wurden immer dann verworfen, wenn sie zu wenig leistungsfähig waren oder zu schwierig zu implementieren. Sie wurden durch Bessere mit gleicher oder erweiterter Bedeutung ersetzt - CMOVE ist so ein Wort. Forth-83 legte fest, daß CMOVE Bitmuster im Speicher hinterläßt, wenn Quelle und Ziel der Verschiebung überlappen. Das fordert aber byteweises Arbeiten und ist uneffektiv auf vielen Maschinen, die ganze Blöcke auf einmal verschieben können. Somit war CMOVE unglücklich festgelegt. Konsequenterweise ist es dem move nun erlaubt, so wirksam wie möglich zu sein - als reiner Blockoperator. Muster erzeugt man

Liste 1:

Ergänzungen gegenüber 'Forth-83 Required Words'

>NUMBER
Ersetzt CONVERT

2! 2* 2@
Essentiel

2DROP 2DUP 2OVER 2SWAP
Vervollständigung der Zwei-Zellen-Worte

ACCEPT
Ersetzt EXPECT und SPAN

ALIGN ALIGNED
Portable Adressenbestimmung

BL
Leerzeichen

C,
Vervollständigung der Zeichen-Worte

C" CHAR S" [CHAR]
'String and character literals'

CELL+ CELLS CHAR+ CHARS
Portable Adressierung

ENVIRONMENT?
Abfrage des 'Environment'

EVALUATE
Der Forth Interpreter

FM/MOD M* SM/MOD
Kompromiss zur Division

INVERT
Kompromiss zu NOT

MOVE
Ersetzt CMOVE und CMOVE>

PARSE
Nützlicher Compilierfaktor

POSTPONE
Ersetzt COMPILE und [COMPILE]

RECURSE UNLOOP
Verbesserter Kontrollfluß

SHIFT
Portable Bitmanipulation

schließlich genauso leicht mit C@ C! oder C, und DO LOOP.

EXPECT und SPAN wurden durch das Wort ACCEPT ersetzt. EXPECT hatte eine "verborgene Begleiterscheinung". Es hinterlegte - für Forth untypisch - in der Variablen SPAN die Länge der empfangenen Zeichenkette. ACCEPT hingegen arbeitet nun forthlike mit dem Stack. Kompatibilität zu älteren Programmen ist leicht hergestellt:

```
VARIABLE SPAN
: EXPECT ACCEPT SPAN ! ;
```

>NUMBER ist die verbesserte Version von CONVERT, der Routine, die die Ziffern zur Zahl wandelt. CONVERT brauchte ein unwandelbares Zeichen als Endmarke. Deshalb war man oft gezwungen die Zeichenkette in einen gesonderten Puffer zu übertragen und dort diese Endmarke anzuhängen, weil dies in der Quelle direkt nicht erlaubt bzw nicht möglich war. >NUMBER kann das jetzt an Ort und Stelle erledigen. So können Ziffernfolgen aus dem 'input stream' heraus direkt gewandelt werden. CONVERT kann man aber nachmachen:

```
: CONVERT
  CHAR+ 65535 >NUMBER DROP ;
```

POSTPONE ersetzt gleich zwei Worte: COMPILE und [COMPILE]. Diese machten im Grunde das gleiche - sie verlegten das Compilierverhalten des folgenden Wortes in die Laufzeit der neuen Definition. [COMPILE] war für 'immediate words' und COMPILE für die übrigen gedacht. Man mußte also wissen, welchen Status ein Wort hat. Außerdem legte der Standard diesen Status für jedes Wort ausdrücklich fest. Dadurch wurde die Entwicklung gehemmt. Ein weiteres Problem entstand durch die Art und Weise in der viele diese Worte implementierten:

```
: COMPILE
  R> DUP @ , CELL+ >R ;
```

Diese Form funktioniert nicht in 'nativ code' Forth Systemen, weil dabei eben nicht exakt eine Wortlänge abgelegt wird. Optimierender Code kann dies so nicht verwerten (inline code expansion and peephole optimization). Daher wurde COMPILE verworfen. In fast allen Fällen kann man POSTPONE statt COMPILE verwenden, zum Beispiel:

```
: COMPILE
  POSTPONE POSTPONE ; IMMEDIATE
```

Das Verhalten von [COMPILE] war nur für 'immediate words' definiert. In typischen Anwendungsfällen wurde auch so verfahren. Daher wird diese Eigenschaft jetzt ausdrücklich festgelegt. [COMPILE] darf somit nur noch für Worte benutzt werden, die vom Benutzer selbst stammen, weil der Standard zum Status der Worte ja gerade keine Stellung mehr nimmt. Dieser Mangel an universeller Verwendbarkeit führte zum Entschluß, [COMPILE] aus dem 'core word set' heraus zu schieben.

Will man Forth-83 Programme portieren, die [COMPILE] nur für die verlagerte Compilation des folgenden Wortes verwendet haben, so ist es gleichbedeutend zu POSTPONE. Die Ersatzdefinition lautet dann:

```
: [COMPILE]
  POSTPONE POSTPONE ; IMMEDIATE
```

In anderen Fällen eignet sich eine allgemeinere Version vielleicht besser:

```
: COMPILE, POSTPONE LITERAL
   POSTPONE EXECUTE ;

: [COMPILE]    ' COMPILE, ;
  IMMEDIATE
```

Im Forth-79 war NOT zu 0= synonym, prüfte eine Zahl und hinterließ ein 'flag' auf dem Stapel. Forth-83 änderte dies ab; NOT invertierte nun jedes einzelne Bit seines Argumentes. Zudem wurde die Bedeutung von "wahr" und "falsch" geändert auf "alle Bits gesetzt" und "alle Bits gelöscht". Zwar verhielt sich NOT in Sequenzen wie < NOT im FORTH-79 und -83 gleich. Leider konnte NOT nun aber den Sinn eines 'flag' vor IF nicht richtig wandeln. Denn die Entscheidung beruhte auf der Logik "Null" oder "Nicht-Null" - invertiert man aber "Nicht-Null" bitweise, so entsteht leider nicht "Null", sonder häufig wieder "Nicht-Null". Im Forth-79 konnte NOT solche 'flags' umkehren und wurde ständig in dieser Weise benutzt. Solche Programme waren im Forth-83 plötzlich nicht mehr nicht lauffähig. Das gab Ärger. Es ist nun zu spät die Bedeutung wieder zu-

rückzunehmen, denn dies würde jetzt alle '83 Programme zerstören. Daher wurde jetzt NOT ganz entfernt. Zur Invertierung der Bits wird INVERT der korrekte Operator sein. Im ANS Forth sind daher sowohl Forth-79 wie -83 Programme wieder ablauffähig, wenn eine kleine Erklärung vorangeschickt wird:

```
: NOT   0= ;      ( Forth-79)
: NOT   INVERT ;  ( Forth-83)
```

So wird dieser klassische Streit nun endlich beigelegt.

Richtig und falsch oder wahr und unwahr

Fall A:
true 11111111 wahr
false 00000000 unwahr

Fall B:
true IF 00000001 richtig
false IF 00000000 falsch

Im Fall A galt im Forth-83: TRUE NOT ist FALSE. Dies bleibt so im ANS-Forth, heißt aber anders: TRUE INVERS ist FALSE. Hier ist die 2er-Komplement-Inversion gemeint.
Im Fall B ist TRUE von anderer Bedeutung: "Richtig" signalisiert hier eine Entscheidung, die andere Möglichkeit dabei heißt "falsch". Es sind Bedeutungsträger.
Die sprachliche Gleichsetzung von richtig mit wahr sowie falsch mit unwahr führt zur Verwirrung in der Logik. Diese Unschärfe hat viel Ärger bereitet, obwohl die praktische Fallentscheidung mittels IF funktionierte. ANS trennt nun richtigerweise die beiden Arten der Umkehrung. INVERS für die bitweise Umkehrung und 0= für die Umkehrung der Entscheidungsrichtung.

PICK und ROLL sind problematisch. Sie sind sehr langsam auf einigen Prozessorarchitekturen und haben den Ruf häßlich zu sein. Daher wurden sie in die 'core extensions' verbannt. (Laxen und Perry verspotteten die zwei im F83 durch den Vorschlag, die Swoperatoren auf SHAKE, RATTLE und ROLL zu erweitern). Es wird empfohlen PICK und ROLL zu implementieren, um ältere Programme zu unterstützen. Neuerer Code sollte darauf verzichten.

Änderungen der Semantik

Ein weiteres Ziel des ANS war es, die Wahlmöglichkeiten für Implementatoren zu erweitern. Aus diesem Grunde wurde darauf verzichtet, Worte als 'immediate' oder 'non-immediate' festzulegen. Solange wie die Standard Worte eines Systems sich an die Ausführungs- und Compilationsbedingungen des Standards halten, ist ihr Status eine Sache der Implementation. LEAVE zum Beispiel kann auf zweierlei Weise implementiert werden. Die Version 'immediate' erzeugt zur Compilezeit eine Verzweigung hinter das Schleifenende LOOP und die andere benutzt für den Schleifenabbruch Informationen auf dem Returnstack, welche dort vom DO abgelegt worden sind. Beide Versionen sind im ANS legal. Die Frage 'immediate' oder nicht offen zu lassen erlaubt es auch eine spezielle Compilation mit Forthmaschinen durchzuführen. Dabei sind eine Reihe primitiver Forthworte wie DUP DROP usw. 'immediate' und erzeugen optimierten Code (peephole optimization).

Diese Änderung hat Folgen. Will man eine Compilation in die Laufzeit verlegen, kann man dazu nicht mehr COMPILE oder [COMPILE] benutzen. Denn man weiß ja nicht mehr, welchen Status ein Wort hat. POSTPONE ersetzt die beiden zwar, es läßt aber diesen Unterschied im Verborgenen. Applikationen, die diese Unterscheidung brauchen, sind jedoch selten. Falls nötig kann man aber mit FIND den Status ermitteln.

Der Begriff 'parameter field' wurde in 'data field' umbenannt und bedeutet nur noch "Speicher für ein Wort erzeugt durch CREATE". Im Forth-83 hatte jedes 'defining word' eine 'pfa'. In Nativcode-Forthsystemen hingegen ist alles nur Opcode und nur die mit CREATE erzeugten Worte haben traditionelle Parameterfelder. Diese Beschränkung auf das Wesentliche bewirkt aber keinerlei Einbuße an Fähigkeiten. Auch im Forth-83 war eine portable Manipulation der Parameterfelder nur für solche Worte gegeben, die durch CREATE erzeugt worden waren. Und eben dies wird nun standardisiert. Die entsprechende Ände-

Liste 2:

Ersetzte oder weggelassene 'Forth-83 Required Words'

-TRAILING
Verschoben zum 'Strings word set'

.(0>
Trivial und unpopulär

2+ 2-
Trivial und obsolet

BLK BLOCK BUFFER
Verschoben zum 'Block word set'

CMOVE CMOVE>
Nicht portabel und leistungsschwach

COMPILE [COMPILE]
Incompatibel mit 'native code'

CONVERT
Ersetzt durch >NUMBER

D+ D< DNEGATE
Verschoben zum 'Double Number word set'

DEFINITIONS VOCABULARY
Ersatz als Vokabular-Kompromiss

EXPECT SPAN
Ersetzt durch ACCEPT

FLUSH LOAD SAVE-BUFFERS
Verschoben zum 'Block word set'

UPDATE FORGET
Verschoben zum 'Programmers Toolkit word set'

FORTH
Verschoben zum 'Search Order word set'

FORTH-83
Obsolet

NOT
Entfernt als Forth-79 /83 Kompromiss

PAD
Unsicher

PICK ROLL
Ungelenk und leistungsschwach

Liste 3:

Semantische Änderungen gegenüber FORTH-83

FORTH-83: Legte fest, welches Wort 'immediate' sein mußte.
ANS: Nicht länger festgelegt LEAVE DUP DROP etc. COMPILE [COMPILE] FIND

FORTH-83: 'Parameter Field'
ANS: Heißt jetzt 'Data Field'. Speicherplatz für ein Wort das mit CREATE erzeugt wurde. Entsprechende Änderung in >BODY.

FORTH-83: Nur 'floored division'
ANS: Spezifikation entspannt. 'floored', gemischte oder symmetrische Division erlaubt.
/ /MOD MOD */MOD */

FORTH-83: Selbsttätiger Wechsel des Vokabulars erlaubt.
ANS: Aufgehoben. : (colon)

FORTH-83: Beendigung einer Eingabe durch 'return' oder Zeichenzahl möglich.
ANS: Nur noch durch 'return' EXPECT

FORTH-83: Compilier-Verhalten von Worten der Kontrollstrukturen zum Teile eingeschränkt.
ANS: Erweitert. WHILE jetzt möglich - auch mehrfach - in:
BEGIN ... REPEAT
BEGIN ... UNTIL
DO ... LOOP

FORTH-83: Konzept von 'definig words'
: MAKE ... CREATE ...
DOES> ... default ... ;
ANS:
Verallgemeinert. Neben dem bisherigen Verhalten jetzt alternativ dazu möglich:

```
: INSTEAD   DOES> ...
alternative ... ;
MAKE THIS                ( THIS
does default )
MAKE THAT INSTEAD  ( THAT
does alternative)
```

rung an >BODY wurde vorgenommen.

Die Bestimmungen für die Operatoren der vorzeichenrichtigen Division (/ /MOD MOD */MOD und */) wurden dahingehend entkrampft, daß nun beide Arten zugelassen sind. (floored or symmetric division)

Forth-83 legte fest, daß : (COLON) auch die Suchfolge änderte Es wurde umgeschaltet vom ersten Vokabular der Suchfolge auf das aktuelle Compilationsvokabular. Einige Benutzer haben diese Eigenschaft ausgenutzt, für die meisten war es ohne besonderen Effekt, für einige aber war es sehr hinderlich - die Suchfolge war nicht präzise zu kontrollieren. In ANS Forth wird : (COLON) die Suchfolge nicht mehr beeinflussen. Das alte Verhalten kann jedoch wieder erzeugt werden, wenn man : redefiniert und dabei die Möglichkeiten des 'search order word set' benutzt.

Die alten Standards bestimmten, daß die Zeicheneingabe für EXPECT dann abgeschlossen ist, wenn ein 'return' empfangen wurde oder eine zuvor festgelegte Zahl von Zeichen eingetroffen war. Außerdem mußte für das 'return' ein Leerzeichen angehängt werden. Solcher Art krause Vorschriften sind schwierig, teuer oder manchmal sogar unmöglich zu implementieren. Demzufolge weichen einige Implementationen in diesem Punkt auch von Forth-83 ab. In Anerkennung der Tatsache, daß editieren und einsammeln von Zeichen oft durch das Betriebssystem übernommen wird und außerhalb der Kontrolle des Forth liegt, wird nun darauf verzichtet, dem EXPECT oder ACCEPT solche Vorgaben zu machen.

Einige Worte wurden in ihren Fähigkeiten erweitert. So kann jetzt WHILE auch mehrfach in einer BEGIN REPEAT Schleife verwendet werden und darf auch in BEGIN UNTIL Schleifen eingesetzt werden. Die Hintergründe dazu sollen an anderer Stelle besprochen werden. Der Gebrauch sollte mit gleicher Syntax wie bisher möglich sein und keine Probleme bieten.

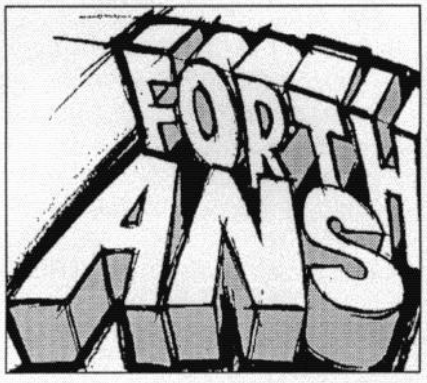
Das Wort DOES> wurde ebenfalls verallgemeinert. Im Forth-83 mußte noch vor jedem DOES> einmal CREATE direkt oder indirekt aufgerufen werde. Dieses war jedoch eigentlich unnötig und wurde daher aufgehoben. So kann man z.B. ein definierendes Wort mit einem Default-Verhalten so bilden:

```
: MAKE    ...  CREATE  ...
  DOES>   ...  default ...  ;
```

Ein Alternatives Verhalten der Kinder kann jetzt so erreicht werden:

```
: INSTEAD  DOES>
  ...  alternative  ...  ;

MAKE THIS MAKE THAT INSTEAD
(  THIS does default,
 THAT does alternative )
```



Damit sind nun grundlegenden Änderungen des kommenden Standards ANS Forth aufgezeigt. So in etwa wird ANS Forth aussehen. Ob im kommenden Jahr noch wesentliches dazukommt bleibt abzuwarten.

Literaturangabe:

Forth Programming System X3J14 Basis Dokument, revision: BASIS15
ANS X3/X3J14 Technical Commitee 1991

Fortsetzung von S. 9

F68K - letzte Neuigkeiten

F68K, das portable Forthsystem für ALLE 68000er, entwickelt sich beständig weiter. Auch die Verbreitung geht zwar nicht rasend, aber ziemlich stetig vonstatten. Es habe sich inzwischen engagierte Entwickler gefunden, die ihre Zeit und ihr Know-how für die Verbesserung und Erweite-

rung von F68K einsetzen. Einige F68K-Installationen sind auch schon in das europäische Ausland gegangen. Es geht also vorwärts.
So wird ab sofort der Lader von Dirk Kutschner für Sinclair QL standardmäßig mit ausgeliefert. Von Marcus Redeker stammen einige Utilities, wie z.B. ein Pendant zum AUTOEXEC.BAT, die nun immer in den mitgelieferten Blockstreams zu finden sind. Aus dieser Richtung ist wohl auch ein schicker Editor bereits im Anmarsch. Natürlich sind schon wieder ca. 100 Millionen Fehler gefunden und entfernt worden. Die zweiten 100 Millionen habe ich mir für die nächste Ausgabe aufgehoben.
Außerdem hat sich am F68K-Kern einiges verändert. So driftet er immer noch langsam aber unaufhaltsam in Richtung ANSI. Auf (schnelle) lokale Variablen braucht man z.B. schon lange nicht mehr zu verzichten und COMPILE oder [Compile] sucht man ebenso wie ASCII im Kernel vergebens. Andere Parameter für FIND oder die Implementation von EVALUATE stehen demnächst an.
Die Dokumentation ist inzwischen auf etwa 130kB gewachsen, so daß man auch etwas zum lesen bekommt. Die Portierung auf andere Rechner ist mit dieser Dokumentation leicht möglich. Immer noch schwer gesucht ist ein Amiga-Besitzer, der bereit ist, eine halbe Stunde seiner Zeit zu opfern, um F68K der gesamten Amiga-Welt zugänglich zu machen. Ruhm und Anerkennung wärem ihm sicher (Reichtum weniger). Ähnlich gilt dies für Mac oder OS/9-Anwender. Oder hat schon jemand einen NeXT? Natürlich ist auch jede andere Form der Beteiligung erwünscht und wird von mir und vielleicht manchem F68K-Anwender mit unbezahlter Dankbarkeit gewürdigt.
Wie man sieht, ist eigentlich für jeden etwas dabei. Auch Sie (ja, Sie) sollten der Welt zeigen, daß es noch wahre Helden gibt und sich uneigennützig in die Arbeit stürzen.

Weitere Informationen wie immer bei:
Dipl. Phys. Jörg Plewe,
Großenbaumer Str. 27,
4330 Mühlheim an der Ruhr.
Tel.: 0208/423514

Kompatibilität zur Forth-83 Programmen herstellen

Das folgende Programm kann auf einem ANS Forth geladen werden um eine zum Forth-83 kompatible Umgebung zu erzeugen. Damit sind die Worte wieder verfügbar, die sonst nurmehr als Extensionen bestehen. Umgebungsbedingungen können naturgemäß damit nicht angepasst werden. Korrektheit ging vor Leistungfähigkeit.

```
( FORTH-83 COMPATIBILITY SUITE )
DECIMAL
: .(    [CHAR] ) PARSE TYPE ; IMMEDIATE
: 0>    0 > ;
: -TRAILING      ( C-ADDR U1 -- C-ADDR U2 )
  DUP IF
    DUP 0 DO
      2DUP 1- CHARS + C@ BL - IF LEAVE THEN 1-
    LOOP
  THEN ;
: 2+    2 + ;
: 2-    2 - ;

( NOTE: IF THE RANGES DO NOT OVERLAP, CMOVE AND
CMOVE> MAY BE DEFINED AS:
: CMOVE   CHARS MOVE ;
: CMOVE>  CHARS MOVE ; )
: CMOVE  ( c-addr1 c-addr2 u -- )
  ?DUP IF
    0 DO  OVER C@ OVER C!  SWAP CHAR+  SWAP CHAR+
    LOOP
  THEN  DROP DROP ;
: CHAR-    1 CHARS - ;
: CMOVE>   ( c-addr1 c-addr2 u -- )
  ?DUP IF
    DUP >R 1- CHARS + SWAP  R@ 1- CHARS +
    SWAP R> 0 DO
      OVER C@  OVER C!  SWAP CHAR-  SWAP CHAR-
    LOOP
  THEN DROP DROP ;

: COMPILE   POSTPONE POSTPONE ;  IMMEDIATE
: COMPILE,   POSTPONE LITERAL  POSTPONE EXECUTE ;
: [COMPILE]    ' COMPILE, ;  IMMEDIATE
( IF [COMPILE] IS ONLY BEING USED TO DELAY
COMPILATION OF FOLLOWING WORD:
: [COMPILE]   POSTPONE POSTPONE ; IMMEDIATE )

: CONVERT   CHAR+ 65535 >NUMBER DROP ;

VARIABLE SPAN
: EXPECT  ( c-addr +n -- )  ACCEPT SPAN ! ;

: NOT   INVERT ;

CREATE PAD  84 CHARS ALLOT

: PICK   ?DUP IF  SWAP >R 1- RECURSE  R> SWAP
  ELSE DUP  THEN ;
: ROLL   ?DUP IF  SWAP >R 1- RECURSE  R> SWAP THEN ;
```

(The following are only needed if the rounding direction matters.)

```
: /MOD   >R  S>D  R> FM/MOD ;
: /      /MOD SWAP DROP ;
: MOD    /MOD DROP ;
: */MOD  >R M* R> FM/MOD ;
: */     */MOD SWAP DROP ;
```

Ein optimierender FORTH-Compiler

Bernd Paysan

Wie holt man das letzte aus einem FORTH-Compiler heraus? Ein portabler Optimierer für Native-Code-FORTH wird am Beispiel bigFORTH für den 68000 vorgestellt. Es wird gezeigt, wie man die Schnittstellen zwischen zwei Makros verkürzt. Ein passender Decompiler wird erläutert.

Stichworte:

- **68K-FORTH**
- **Native Code**
- **Optimierender Compiler**
- **Peephole-Optimierung**

Schon Jörg Plewe hat in [1] gezeigt, daß der MC68000 ein idealer Prozessor für ein Native-Code-FORTH ist. Seinen Weg zur High-Speed will ich hier nur kurz nachzeichnen und dann dort weitermachen, wo er aufgehört hat (bei der Optimierung).

Traditionelles FORTH erzeugt indirekt gefädelten Code. Der Prozessor ist hauptsächlich mit dem Lesen von Zeigern und dem Springen nach DOCOL beschäftigt. Ein klassisches NEXT/DOCOL/UNNEST besteht etwa aus den Befehlen

```
NEXT:    move   (IP)+,W
         move   (W)+,A0
         jmp    (A0)
docol:   move   IP,-(RP)
         move   W,IP
         NEXT
unnest:  move   (RP)+,IP
         NEXT
```

Man sieht, daß für jedes Hochsprachewort zur Ausführung einen Overhead von 9 Befehlen benötigt, von denen ganze 6 auch noch auf den Speicher zugreifen. In einem Call-List-FORTH wird der Overhead auf nur noch zwei Befehle (jsr und rts) reduziert, ohne den FORTH-Compiler nennenswert zu verändern. Da auch jsr und rts einen nicht unerheblichen Overhead beinhalten (diesen aber auf den Prozessor verschieben), erreicht man etwa eine Leistungsteigerung um den Faktor 2 bis 3.

Richtig ab geht die Post, wenn man kleine und kurze Primitives (mit CODE definierte Wörter) als Makros compiliert. Die 36 Taktzyklen, die so ein Hin- und Rücksprung verbrauchen, sind damit getilgt. Angesichts der Tatsache, daß ein Wort wie DROP selbst nur 4 (vier!) Taktzyklen braucht, eine nicht zu unterschätzende Leistungssteigerung!

Anhand eines einfachen Beispiels, das auch fantastisch gut optimiert werden kann, will ich zeigen, wie so ein FORTH-Wort dann als Code aussieht. Hier erst mal der FORTH-Source:

```
: TEST DUP 0= IF DROP THEN ;
```

Auf der Optimierungsstufe der Makrocompilation sieht das dann etwa so aus:

```
         move.l   (SP),-(SP)    ;DUP
         move.l   (SP)+,D0      ;0=
         seq      D0
         ext.w    D0
         ext.l    D0
         move.l   D0,-(SP)
         move.l   (SP)+,D0      ;?BRANCH
         beq      then
         addq.l   #4,SP         ;DROP
then:    rts                    ;UNNEST
```

Wie man sieht, ist das ein rechter Wust an offensichtlich unnötigen, weil redundanten Assembler-Befehlen. So wird mehrmals hintereinander ein Wert auf den Stack gelegt und gleich wieder heruntergenommen, ganz besonders deutlich an der Stelle zwischen 0= und ?BRANCH.

Ein erfahrener 68000-Programmierer würde wahrscheinlich folgenden Code produzieren:

```
         move.l   (SP),D0       ;DUP 0=
         bne      then          ;?BRANCH
         addq.l   #4,SP         ;DROP
then:    rts                    ;UNNEST
```

Von 0= ist fast nichts übrig geblieben. Gerade die 68K-Reihe von Motorola ist durch eine leistungsfähige

und orthogonale Flag-Behandlung ausgezeichet. Ein Test oder Vergleich mit anschließendem Sprung kann direkt in diesen Sprung hineingezogen werden. Auch ein DUP mit anschließendem Verbrauch des TOS kann verkürzt werden.

bigFORTH produziert genau diesen Code. Wie macht es das? Ein maschineller Optimierer muß wissen, welche Code-Sequenzen er durch andere (kürzere und schnellere) ersetzen kann, die dieselbe Wirkung haben. Diese Form der Optimierung nennt man "Peephole-Optimierung", da der Optimierer durch ein kleines "Guckloch" auf den Code blickt, den Ausschnitt, den er sieht mit dem ihm bekannten Mustern vergleicht und bei Erfolg verkürzt.

In traditionellen Systemen ist der Optimierer ein eigener Pass, der über den Assembler-Source, oft auch über den linkfähigen Objektcode läuft (für den Optimierer ist das dann eine reine Peepshow!).

Eine solche Vorgehensweise widerspricht natürlich zutiefst der FORTH-Philosophie. Ein FORTH-Compiler ist gezwungen, aus einem gefundenen Wort sofort lauffähigen Code zu compilieren - normalerweise durch Absetzen des Tokens mit , . Diese Methode muß beibehalten werden, wenn man das Token vielleicht auch mit einem Wort namens CFA, absetzen kann.

Als Guckloch steht also nur noch die Schnittstelle vom letzten und dem gerade compilierten Makro zur Verfügung. Gottseidank kann man gerade hier die meisten und wirkungsvollsten Optimierungen anbringen. Der Optimierer muß dazu nur wissen, was das letzte Wort gemacht hat und was das nächste Wort braucht. Hat z.B. das letzte Wort einen Wert aus D0 auf den Stack gelegt und das nächste hätte den gerne wieder in D0, so läßt man den letzten Befehl vom letzten Wort und den ersten vom nächsten Wort einfach weg.

Wie kommt der Optimierer nun an seine Informationen? Der erste Einfall ist sicher, den Code direkt anzusehen und danach zu entscheiden. Beispiel: $2D00 ist der Code für move.l D0,-(SP), $201E der für move.l (SP)+,D0. Treten beide Codes nebeneinander auf, werden sie entfernt (also: entferne alle $2D00201E). Leider ist diese Idee nicht brauchbar, da ein gut programmiertes ?DUP auch mit $2D00 aufhört; dieses darf aber nicht wegoptimiert werden, da es nur bedingt aufgerufen wird.

Eine bessere Idee ist es, die Informationen über das Verhalten des Makros mit zusätzlichen Werten zu beschreiben. So reicht ein Byte für den Makro-Anfang ("Take-Byte") und eines für das Makro-Ende ("Push-Byte") völlig aus. Betrachtet man die FORTH-Primitives, so kommt man zu dem Schluß, daß sie die vielen Register des 68000 gar nicht benutzen, ebenso wie nur einige wenige Befehle zur Verwendung kommen.

Grundsätzlich wird ein Register für arithmetische Operationen und ein Register für Adressen benötigt, wir nehmen mal D0 und A0. Da die 68K-CPUs bei allen Operationen die Flags setzen, erübrigt es sich oft, den TOS zu testen, wenn eine Flag gebraucht wird. Dank des othogonalen Befehlssatz (zumindest, was Bedingungen angeht), kann man ohne Probleme einen Test mit folgenden ?BRANCH "verwursten". Der umständiche Code zum Erzeugen von Flags zwingt einen geradezu dazu, ihn zu optimieren:

```
scond     D0
ext.w     DO
ext.l     D0
move.l    D0,-(SP)
```

Es kann also neben A0 und D0 auch noch eine FLAG zurückgegeben bzw. geholt werden. Weitere Untersuchungen zeigen, daß auch @ zu den Ausnahmen gehören sollte, sowie die Operationen mit dem Returnstack und Stackbefehle wie OVER und DROP (vor DROP kann der letzte Push-Befehl einfach weggelassen werden).

Ebenso wichtig ist es, die arithmetischen Operationen zu beschleunigen, da sie oft zusammen mit Literals auftreten, wie etwa 50 + oder $FF AND. Auch mit Literals "verknotete" arithmetische Befehle können und sollten nochmals abgeändert werden, wenn darauf eine weitere Operation folgt, die den TOS nach D0 oder A0 haben will, wie etwa bei einem ARRAY + @.

Insgesamt ergibt sich aus der anfangs einfachen Idee doch ein recht kompliziertes Regelwerk. So muß ein Wort nicht nur seine Anforderungen angeben, sondern auch, was es mit dem Wert getan hat bzw. zu tun gedenkt (im Falle der arithmetischen Befehle). Beides muß zudem getrennt werden, da verschiedene Optimierungen auch verschieden reagieren, so ist es einem DUP wurst, ob dahinter ein + kommt, oder ein SWAP; Hauptsache, beide wollen den obersten Wert in D0 haben (wobei DUP SWAP kein sehr erhellendes Beispiel ist, es ist obsolet).

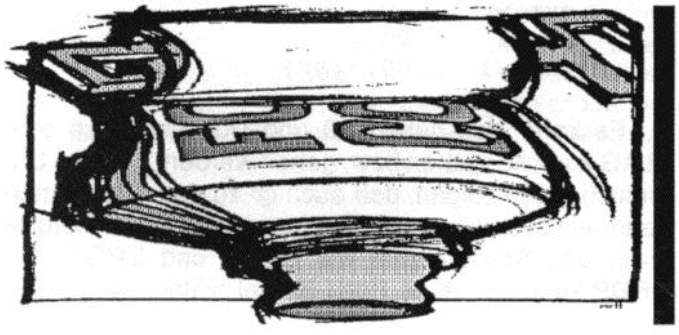
Trotzdem das hier alles recht schwierig und kompliziert klingt, ist der eigentliche Code nur drei Screens lang.

Ich möchte diesen Code "top-down" erklären: CFA, ersetzt das , im Compiler. Alles was darüber liegt, dürfte einem FORTH-Profi ohnehin klar sein. 'CFA, ist nur für Benutzer da, die ihre eigene Optimierung brauchen. (Im ANS-FORTH heißt dieses CFA, übrigens COMPILE, - ich halte den Namen CFA, aber für besser).

Die (User-)Variable LASTDES ist eine 4-Byte-Variable, die die letzten Deskiptoren "aufhebt" und in einem verarbeitbaren Format bereitstellt. Sie enthält der Reihe nach: First Take (das erste Takebyte); Previous Push (was das letzte Makro hinterlies); Actual Take (was dieses Makro haben will) und Actual Push (was dieses Makro hinterlassen wird).

!LASTDES schiebt nun einfach den letzten "Actual Push" in den "Previous Push" und löscht die aktuellen Werte. Es ist hauptsächlich wegen der prägnanten Kürze des Codes in Assembler geschrieben.

Nun gilt es noch zwei Sonderfälle abzufangen: Variablen und andere ohne änderndes DOES> von CREATE definierten Wörter werden als Literals compiliert. Konstanten werden analog als Literal @ compiliert (man kann sie dann mit einem TO-Konzept patchen).



Wie es weitergeht, bestimmt das unterste Bit im Length-Field (das ist ein 16-Bit-Wort direkt vor dem Code Field). Ist es 0, so wird ein normales Wort mit jsr bzw. bsr angesprungen. Ansonsten wird der Makrodeskriptor (der am Ende des Codes liegt) gelesen und in die hintere Hälfte von LASTDES gespeichert. Sollte dieses Wort das erste compilierte sein, wird sein Take-Byte in das Feld "First Take" geschrieben (damit können auch Hochsprache-Makros ohne explizite Angabe von Take und Push richtig optimiert werden).

Nun wird noch geschaut, ob überhaupt etwas zu tun ist (also keiner der beiden Deskriptoren 0 im Low-Nibble hat), wenn nein, wird das Makro einfach kopiert (mit 0 MACRO,), sonst geht's erst richtig ab mit OPT,!

Auch hier erst die Ausnahmebehandlungen: Die meisten Makros geben ja außer dem Wert auch noch eine korrekte Flag zurück, aber leider nicht alle. So gibt C@ kein korrektes Vorzeichen zurück (für den 68000 ist ein Zeichen>127 negativ, für FORTH positiv - das ist auch der Grund, warum so viele C-Programme bei Umlauten spinnen, FORTH-Programme aber nicht). Solche Fälle werden durch ein move.l D0,D0 (Opcode $2000) gelöst. Ein tst.l D0 hätte es auch gemacht, aber das ist ein Code, den jeder Assemblerprogrammierer benutzt; $2000 kann dagegen vom Decompiler einfach ignoriert werden. Die Ausführungsgeschwindigkeit ist gottseidank gleich.

Auch die Optimierung von aufeinanderprallenden Flags wie 0= IF ist eine Außnahme. FLAG> behandelt sie. Es merkt sich die Flag, und setzt sie nach gelungener Compilation mit den notwendigen Modifikationen wieder in den Code ein.

Der Rest ist Tabelle. Gut, es sieht wüst aus, die zwei verschachtelten DO LOOPs, es soll auch nicht ästhetisch sein, sondern schnell. Die äußere Schleife sucht die zum Push-Byte passende Tabelle (das oberste Push-Bit hat eine Extrafunktion für die Ausnahmebehandlung der Flags, wird also ignoriert). Die innere Schleife sucht dann nach dem passenden Code zum Take-Byte. Beide Schleifen werden nur dann durch ihren natürlichen Ausgang verlassen, wenn die Suche erfolglos war. Dann wird ein normales Makro compiliert.

Nun geht es nur noch um Basiswörter. Die Erkennung und Behandlung von CREATEd Words und CONSTANTs ist eigentlich offensichtlich. Schwieriger wird's da bei MACRO, und (OPT,. Die dritte Zeile von MACRO, kann man gleich wieder vergessen, sie ist nur notwendig, weil mein bigFORTH relokatibel sein muß. Der Rest wäre besser so:

```
: macro, ( addr count skip - )
  /string here swap dup allot cmove ;
```

Also wird das Makro einfach mit CMOVE verschoben. Auch (OPT, ist im Grunde recht einfach. Der 8-Bit-Wert, der an dritter Stelle im Tabellen-Eintrag steht, bestimmt, wieviel vom letzten Makro überflüssiger Code ist. Dahinter steht als counted String, was einzusetzen ist. Da manche Werte variabel sind (z.B. der Wert eines Literals), muß eine Konvention gefunden werden, damit sie nicht überschrieben werden: Jeder Opcode, der 0 ist, wird nicht compiliert, es wird damit der alte Wert behalten.

Zu guter Letzt wird noch das Push-Byte des letzten mit dem Pushbyte des vorletzten Makros verknotet. Das hat folgenden Sinn: War z.B. das vorletzte Makro ein Literal, das letzte ein + und das nächste ist ein @, so ist vom Literal nichts mehr übrig geblieben, vom + aber auch nicht. Die Kombination aus beidem, die im Code entstanden ist, muß auch im Pushbyte stehen. Wann das der Fall ist, und wie das geschieht, soll jeder, der Spaß an solch kniffligen Rätseln hat, selbst herauskriegen.

Damit man noch einen Eindruck von der Tabelle gewinnt, hier eine auf drei Screens verkürzte (aber funktionsfähige) Tabelle. Meine ist im Moment 8 Screens lang, aber auch mit dieser könnte man schon ganz gut leben.

Es ist also nicht viel dabei an einem optimierenden Compiler für FORTH. Diese Definitionen sind zwar nicht gerade der Ausbund an schönem Stil, aber sie tun, was sie sollen, sie ergeben kompakten Code und sie sind sehr schnell (gerade an einer so wichtigen Stelle im System ist das entscheident).

Es ist jedenfalls befriedigend für das Selbstbewußtsein, daß dieser Mini-Peephole-Optimierer mit seiner Codequalität alle mir bekannten C-, Pascal- und Modula 2-Compiler schlägt, deren Code in einem Zwischenstadium UPN-Format hat, obwohl diese meist nur 16-Bit-Werte lesen und schreiben (was schneller geht als 32-Bit-Werte).

Nur sehr gute Compiler erzeugen direkt Code für Register-Maschinen wie die 68K (etwa Turbo-C oder

GNU-C). Dafür, daß sie ihren Code als Baum verwalten und viel mit den Registern herumjonglieren müssen, ist ihr Ergebnis nicht berauschend besser (etwa um den Faktor 2 bei Turbo-C).

Nun will ich noch ein wenig protzen und ein paar getürkte Benchmarks bringen ("Benchmarks do not ly, lyers do benchmarks"). Damit man gleich die gesammelten Lügen vergleichen kann, produziere ich denselben Benchmark wie in [1]. Zur Erinnerung: NIP wird als SWAP DROP definiert und dann in einer Schleife einhunderttausend mal ausgeführt. Ohne jede Optimierung (also mit reinen Makros) kommt man auf 1,185 Sekunden.

Mogelt man so, wie Jörg Plewe, und optimiert ein bißchen an der Schnittstelle zwischen der 1 und dem NIP, so kommt man tatsächlich auf 0,880 Sekunden (aber wie er das macht, steht nicht in seinem Artikel). So einen Effekt erreicht man mit dem optimierenden bigFORTH, wenn man NIP als SWAP NOOP DROP definiert (NOOP tut nichts, als einen Dreck in das Peephole zu klecksen).

Mit bigFORTH braucht : NIP SWAP DROP ; MACRO gerade mal 0,620 Sekunden. bigFORTH's eigenes NIP kommt auf 0,465 Sekunden und die Schleife selbst verbraucht davon 0,260 Sekunden. Gerechterweise sollte man die Schleife abziehen, da sie ab Makro-Compilation nicht mehr schneller werden kann und den Faktor der Geschwindigkeitssteigerung herabmildert; sie braucht zum Schluß ja schon mehr als die Hälfte der Zeit!

Zugegeben, in bigFORTH 1.00 habe ich es nicht für nötig gehalten, auch DROP und NIP zu optimieren. Ein Code wie 1 NIP würde man eher als DROP 1 schreiben und jedes SWAP DROP gleich durch NIP ersetzen. Mir sind aber etliche ROT DROP-Sequenzen aufgefallen, die so schneller laufen und auch noch kürzer sind. Aber irgendwie stacheln solche Benchmarks dazu an, es doch noch etwas schneller zu machen.

Gut, nun zur Portabilität: Der Algorithmus weiß (ausgenommen die IF-Abfrage) nichts über den generierten Code. Die eigentlichen Daten stehen in der Tabelle. Hier könnte man jederzeit einen beliebigen anderen Assembler verwenden und damit optimierten Code für einen anderen Prozessor produzieren. Für einen aufgebohrten 8-Bit-Prozessor wie den 80x86 wird man allerdings die Markierung im Längenfeld anders vornehmen (z.B. im höchsten Bit). Eine Tabelle an andere Gegebenheiten anzupassen, ist auf alle Fälle viel einfacher, als einen neuen Code zu entwerfen.

Nun habe ich noch versprochen, zu erklären, wie man diesen Code denn wieder decompiliert. Die jsrs zu decompilieren, ist keine Kunst. Von vielen Makros ist aber kaum etwas übriggeblieben. Grundsätzlich aber gilt: Das erste Makro eines Wortes fängt vorne so an, wie es definiert wurde (von seinem Anfang kann zwar nichts übriggeblieben sein, das ist aber immer noch sehr viel). Alles, was an ihm verändert wurde, steht in der Optimier-Tabelle. Man durchsucht also alle Vokabulare nach Makros, vergleicht sie mit dem Codestück, passen sie ganz, ist es am besten, passen sie nicht ganz, ist es auch noch gut.

Man sucht dann in der Optimier-Tabelle nach den Einträgen zu dem Push-Byte des verdächtigen Makros. Alle Einträge, die vom Makro soviel abschneiden, wie nicht mehr gepaßt hat, kommen in Betracht. Auch hier heißt es vergleichen, paßt die Schnittstelle ganz, dann ist es geschafft: Das Makro ist entlarvt.

Nun muß man noch einige Vorkehrungen für die Decompilation des nächsten Makros treffen: Ein Teil von ihm ist ja verschwunden. Er kann aus dem Take-Byte-Eintrag der Tabelle rekonstruiert werden. Auch der Übergang zwischen beiden Makros ist als "erledigt" einzustufen. Man kommt nicht umhin, den zu analysierenden Code "zusammenzubasteln".

Und (das soll man nicht verschweigen): Es dauert und dauert. Dreiviertel der Zeit geht auf die Identifikation der bis zur Unkenntlichkeit verstümmelten Makros, ein Viertel dann auf die Suche nach dem Namenfeld (das ja in dem volksFORTH-Derivat bigFORTH auch nur über die Vokabular-Liste gefunden werden kann). Wie schnell es gehen könnte, sieht man, wenn man ein rein mit Sprüngen definiertes Wort decompiliert.

Trotzdem: Man kann damit leben. volksFORTH ist langsamer, wenn es seinen indirekt gefädelten Code decompiliert (nicht, weil das schwieriger ist, sondern, weil volksFORTH so viel langsamer ist). Und welche Compilersprache ist sonst noch decompilierbar?

Auf alle Fälle: Der Stand der Technik im Compilerbau ist erreicht. Alle Behauptungen, wie 'Kein Mensch baut heutzutage mehr einen Compiler von Hand', die aus der intellektuellen Ecke der Informatiker kommen, werden in den Wind geschlagen. Der Vier-Zeilen-Compiler von FORTH läßt sich mit dreißig weiteren Zeilen FORTH in einen optimierenden Compiler verwandeln, ohne daß er deshalb langsamer wird. Daß er gegen einen mehrere Megabyte Sourcen langen Compiler (die zu 90% von irgendwelchen Programmen erzeugt werden, die ja auch Fehler haben können) wartbarer ist, dürfte ohnehin klar sein.

Nun muß ich noch etwas Farbe bekennen: Ja, hinter bigFORTH stecken wirtschaftliche Interessen. Ein euphorischer (hust, hust, ich meine: Ein neutraler) Testbericht von bigFORTH 1.00 steht in [2]. Da mit solchen spärlichen Angaben eigentlich niemand selbst ein optimierendes FORTH schreiben kann (es sei denn, er hätte schon einen Targetcompiler oder ein assembliertes Kernal), kann ich den Code trotzdem veröffentlichen.

[1] Jörg Plewe, Schnelles FORTH für den MC68000,
Vierte Dimension, Volume VI, 1/90(29)
[2] M. Schultheis, Testbericht: bigFORTH,
Vierte Dimension, Volume VI, 3/90(12)

Source:

```
\ compiler für Create, Variable und Constant            19may91py

| : create?  ( cfa -- cfa f )
    dup cfa@  [ ' udp cfa@ ] ALiteral =  ;
| : constant?  ( cfa -- cfa f )
    dup cfa@  [ ' c/l cfa@ ] ALiteral =  ;
| : constant,  ( cfa -- )  >body   compile ALiteral compile @ ;
| : macro, ( addr count skip - ) dup >r /string
    2dup here swap cmove
    under + 4+ relinfo r> 2/ here >rel 2/ 4 pick 2/  movebits
    allot ;
| : (opt, ( addr -- )
    dup 2+ c@ negate allot  dup 3+ count bounds
    ?DO  i w@ dup  IF  w,  ELSE  drop 2 allot  THEN   2 +LOOP
    c@  IF  0 lastdes 3+ c@ dup  IF  $F0 and
            lastdes 1+ c@ $F and or  T&P ELSE 2drop THEN THEN ;
```

```
\ opt,                                                  19may91py

Variable opt?   opt? on
| : flag>  -8 allot 2 here dup >r c@ $F and 6 xor >r macro,
    r> r> ctoggle ;
| : opt, ( addr len Push&Take -- )
    dup $E161 =  over $8181 = or  over $E181 = or
    IF  drop -2 allot $2000 w, 2 macro, exit  THEN
    $6161 case? IF  flag> exit  THEN
    dup $100 Q/ $7F and  opttab #opt bounds
    DO  dup I c@ =  IF  drop $7F and  dup $F and  I 1+ count
        bounds  DO  I c@ pick I 1+ c@ =
                    IF  2drop  I (opt,  I c@ 2* 2+ macro,
                        unloop unloop exit  THEN
                    I 3+ c@ 4+  +LOOP  LEAVE  THEN
        I 1+ c@ 2+  +LOOP  2drop 0 macro, ;
```

```
\ cfa,                                                  19may91py

Code !lastdes  ( -- )  user' lastdes UP D) A0 lea
     .b 3 A0 D) 1 A0 D) move  .w  2 A0 D) clr  Next end-code

Defer 'cfa,      ' noop IS 'cfa, \ may be patched later
: cfa, ( cfa -- ) 'cfa,  !lastdes
  create?            IF  execute compile ALiteral  exit THEN
  constant?          IF  constant,  exit  THEN
  dup 2- w@ 1 and 0=  IF  jsr, exit  THEN
  dup 2- w@ -2 and  2dup + w@ lastdes 2+ w!
  here lastcfa @ = IF lastdes 2+ c@ lastdes c! THEN
  2- lastdes 1+ w@
  dup $0F00 and 0= over $000F and 0= or opt? @ not or
  IF  drop 0 macro, exit  THEN  opt, ;
```

```
\ T&P: creates literal macros, defined in TARGET.SCR   30apr91py

$00 T&P: :none  $01 T&P: :d0    $02 T&P: :a0    $03 T&P: :>r
$04 T&P: :drop  $05 T&P: :+loop $06 T&P: :comp  $07 T&P: :nip
$08 T&P: :lit   $09 T&P: :user                  $0B T&P: :r>
$0C T&P: :r@    $0D T&P: :over                  $0F T&P: :dup
$11 T&P: :+     $21 T&P: :-     $31 T&P: :or    $41 T&P: :and
$51 T&P: :xor   $61 T&P: :flag
$62 T&P: :!     $72 T&P: :@     $79 T&P: :user@ $78 T&P: :lit@

$18 T&P: :#+    $28 T&P: :#-    $38 T&P: :#or   $48 T&P: :#and
$58 T&P: :#xor
$81 T&P: :d0\-  $E1 T&P: :d0\f

1 2 +thru
```

```
\ opttab                                                28apr91py
Create opttab   Assembler  BEGIN
\ Umwandlung in immediate mu zuerst kommen!
BEGIN  :lit  c,  0 c, \ Dummy
1 c,  :+     c,  6 c, 6 c,  0 # SP ) addi
1 c,  :-     c,  6 c, 6 c,  0 # SP ) subi
1 c,  :or    c,  6 c, 6 c,  0 # SP ) ori
1 c,  :and   c,  6 c, 6 c,  0 # SP ) andi
1 c,  :xor   c,  6 c, 6 c,  0 # SP ) eori
1 c,  :!     c,  6 c, 6 c,  SP )+ 0 L#) move
1 c,  :@     c,  6 c, 6 c,  0 L#) SP -) move
0 c,  :comp  c,  6 c, 6 c,  0 # SP )+ cmpi
0 c,  :d0    c,  6 c, 6 c,  0 # D0 move
0 c,  :a0    c,  6 c, 6 c,  0 # A0 move
0 c,  :>r    c,  6 c, 6 c,  0 # RP -) move
THEN
```

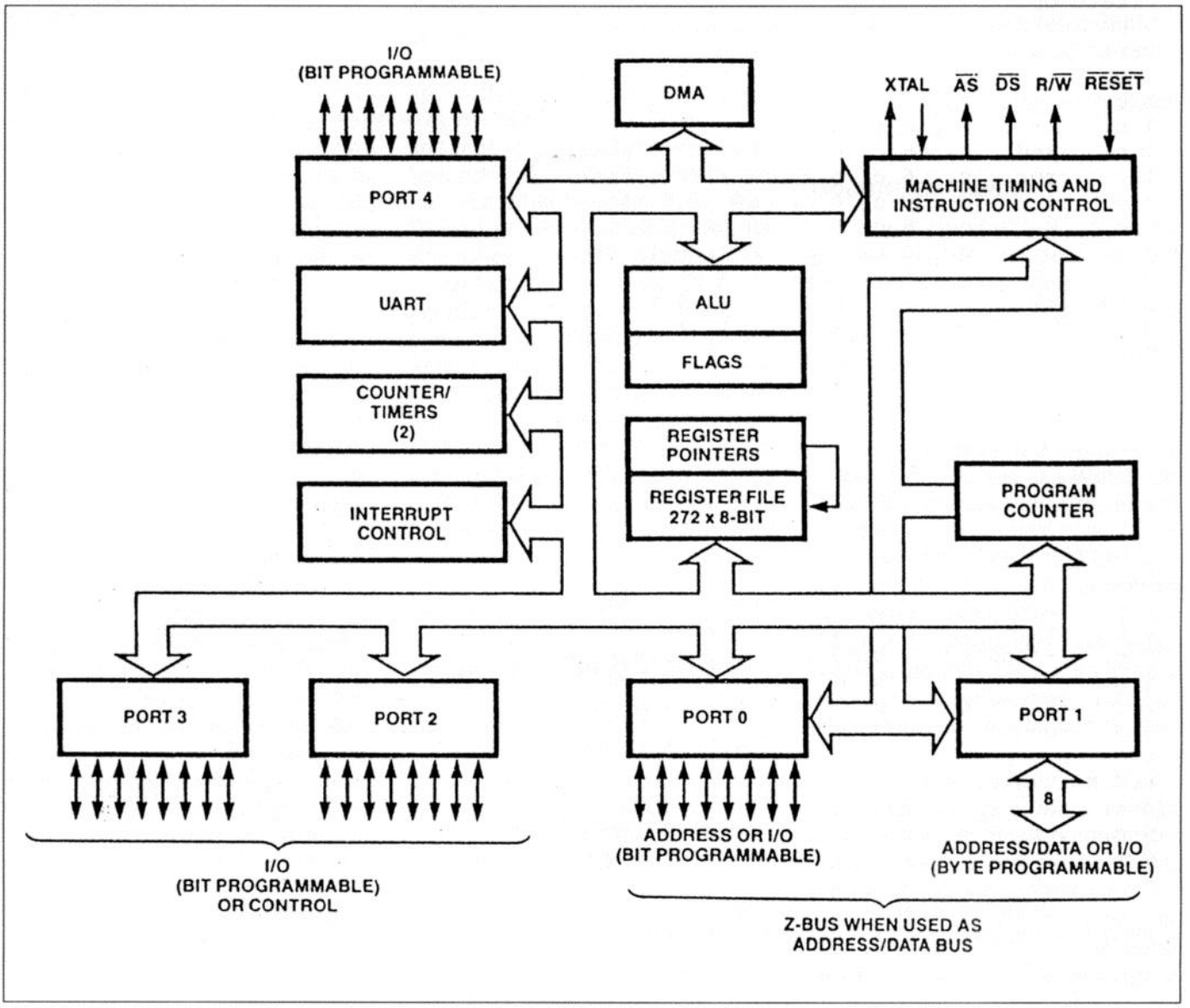
```
\ opttab Fortsetzung                                    28apr91py
BEGIN  :d0   c, 0 c,
0 c,  :comp  c,  2 c, 2 c,  D0 SP )+ sub
0 c,  :a0    c,  2 c, 2 c,  D0 A0 move
0 c,  :+loop c,  2 c, 2 c,  D0 loopreg add
0 c,  :>r    c,  2 c, 2 c,  D0 RP -) move
0 c,  :nip   c,  2 c, 2 c,  D0 SP ) move
0 c,  :d0    c,  2 c, 0 c,
0 c,  :drop  c,  2 c, 0 c,  THEN
BEGIN  :a0   c, 0 c,
0 c,  :d0    c,  2 c, 2 c,  A0 D0 move
0 c,  :>r    c,  2 c, 2 c,  A0 RP -) move
0 c,  :nip   c,  2 c, 2 c,  A0 SP ) move
0 c,  :a0    c,  2 c, 0 c,
0 c,  :drop  c,  2 c, 0 c,  THEN
BEGIN  end-code  swap - Constant #opt
```

Platine für den Zilog Super8-FORTH-Chip

Heinz Schnitter und Hans-Günther Willers

Der Super8 Microcomputer von Zilog ist ein Nachfolger des Z8. Er zeichnet sich durch die bewährte Register-Architektur, sowie durch den integrierten DMA-Kanal und die schnelle, leistungsfähige Interruptstruktur mit insgesamt 27 möglichen Interruptquellen aus [1] [2].

Als Erweiterung des Befehlssatzes sind Divisions- und Multiplikationsbefehle, sowie die Instruktionen ENTER, EXIT und NEXT implementiert. Die drei letztgenannten Befehle sind genau die, die in den virtuellen FORTH-Maschinen die meiste Rechenzeit verbrauchen. Durch die Implementierung als Maschinenbefehle wird ein FORTH auf dem Super8, das von diesen Gebrauch macht, deutliche Geschwindigkeitsvorteile verbuchen können.



BLOCKSCHALTBILD DES SUPER8 <COPYRIGHT ZILOG INC>

Aus diesen Gründen wurde von Dezember 1989 bis Juni 1990 von zwei Mitgliedern der FORTH-Gesellschaft e.V. (Klaus Kohl und Heinz Schnitter) ein FORTH für den Super8 geschrieben und von Zilog in das ROM einer Super8-Maske übernommen (Zilog 0887520PSC).

Damit nicht jeder Anwender des Super8-FORTH-Chips bei seinen ersten Versuchen das "Rad neu erfinden" muß, wurde eine sehr kompakte Platine mit den notwendigsten Bauteilen entwickelt (Maße 64 x 54 mm). Auf ihr befindet sich alles, was benötigt wird, um mit einem industriekompatiblen PC (oder einem Terminal) FORTH-Programme entwickeln und testen zu können.

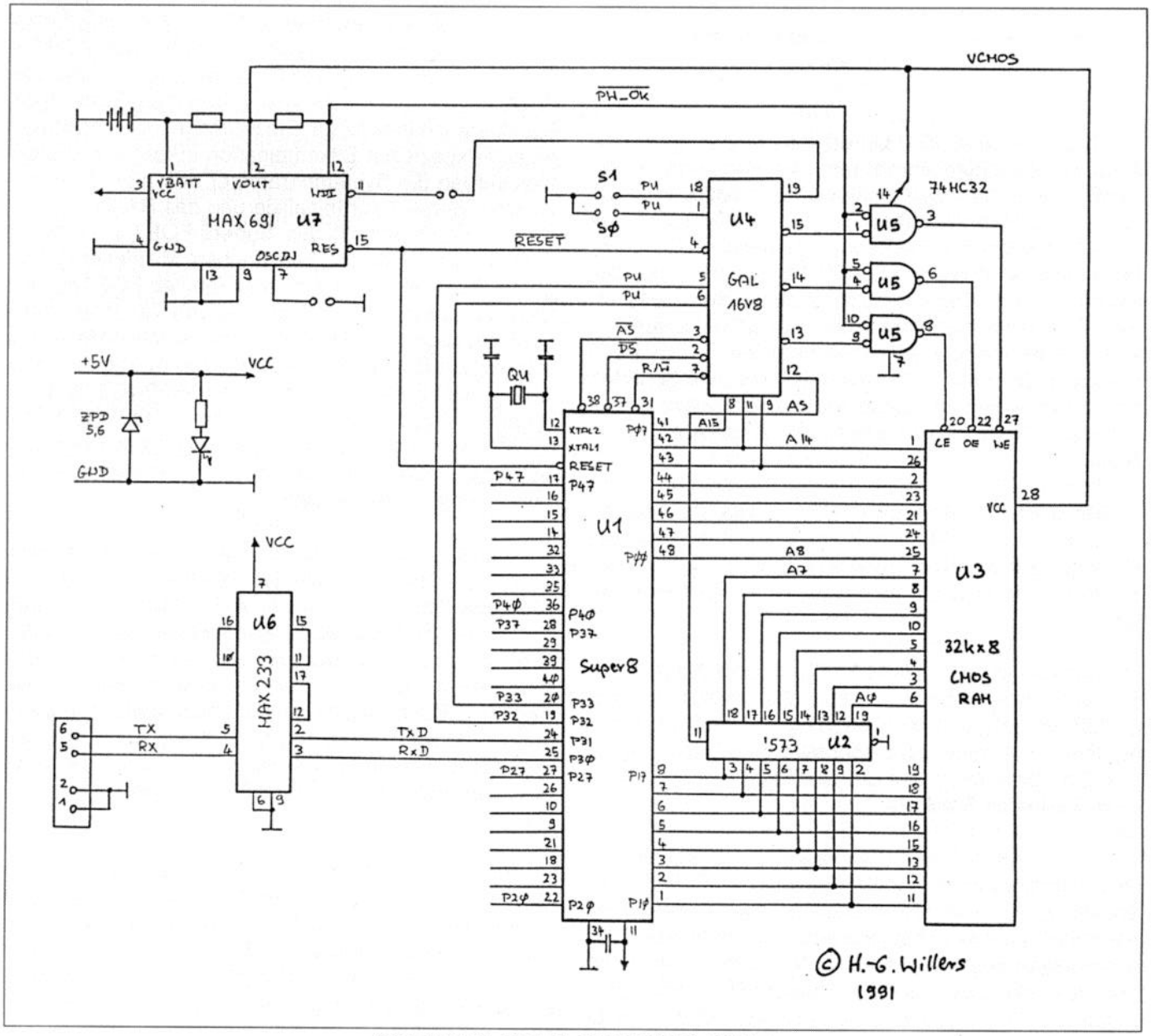
Die Hardware

Herz der Schaltung [4] ist natürlich der Super8 mit dem FORTH im Masken-ROM. Von den fünf 8-Bit Ports des Super8 werden Port0 für die Adressen 8 bis 15, sowie Port1 als gemultiplexter Daten- und Adreßbus verwendet. Während der Adreßphase werden die Adreßbits 0 bis 7 am Bus angelegt und im Latch U2 gehalten.

Alle Port-Bits, sowie die wichtigsten Steuersignale sind auf Pfostenstecker geführt, wo sie zur Adaptierung bereit stehen. Über die Pins der Pfostenstecker wird ebenfalls Masse und die Versorgungsspannung von +5V zugeführt.

Zur Speicherung von Programm und Daten dient das CMOS-RAM U3. Dieses RAM, mit einer Größe von 32 KByte, wird von einem NiCd-Akkumulator auch bei abgeschalteter Versorgungsspannung gepuffert, so daß Programme über längere Zeit (ca. 3 Monate) gespeichert bleiben.

Über U5 werden die Signale 'Chip-Select', 'Output-Enable' und 'Write' des CMOS-RAMs bei abgeschalteter Versorgungsspannung auf inaktivem Pegel gehalten, so daß der Inhalt des RAMs nicht verfälscht wird.



SCHALTBILD DER SUPER8-PLATINE

Im GAL U4 wird die Dekodierung, sowie ein durch Steckbrücken einstellbares Write-Protect gemacht.

Die Gleichungen für die einzelnen Signale sind wie folgt:

```
!ASOUT = ASIN;

  !CS   = A15;

  !OE   = A15 & !DS & WRITE;

  !WR   = A15 & A14 & A13 & !DS & !WRITE
        # A15 & A14 & !S0 & !DS & !WRITE
        # A15 & A14 & !S1 & !DS & !WRITE
        # A15 & A13 & !S1 & !DS & !WRITE
        # A15 & !S0 & !S1 & !DS & !WRITE;

WD.OE = XENABL;

  WD    = P32;

XENABL = RESET & !DS & !WRITE & P32 & !P33
       # RESET & XENABL;
```

Das Super8-FORTH benötigt am oberen Speicherende des 64 KByte Adreßraums ein RAM von mindestens 2 KByte; auf der Platine wird ein CMOS-RAM mit 32 KByte verwendet. Das On-Chip-ROM beginnt bei der Adresse 0 und ist 8 KByte groß. Zwischen On-Chip-ROM und RAM kann ein EPROM oder wie auf der Platine ein schreibgeschützter Bereich des RAMs liegen. Dieser nichtflüchtige Speicher muß auf einer 1 KByte-Grenze beginnen. Innerhalb des CMOS-RAMs kann eine Grenze (RAM-FENCE) in 8 KByte-Schritten mit Steckbrücken eingestellt werden. Unterhalb dieser Adresse RAM-FENCE verhält sich das RAM wie ein ROM.

Mit dem RS-232-Konverter U6 (MAX233) werden die Signale des UARTs des Super8 auf die üblichen Pegel für serielle Schnittstellen gebracht. Die Schnittstellensignale sind auf eine 6-polige Pfostenleiste geführt.

Der Baustein U7 (MAX691 [5]) erzeugt das RESET-Signal für den Super8. Die Umschaltung der Versorgungsspannung von U3 (CMOS-RAM) und U5 auf Batteriespannung geschieht ebenfalls in U7. Das Signal PW_OK (aktiv Low) gibt die Steuersignale des GALs über U5 an das RAM frei.

Die Überwachung der Funktionstüchigkeit von Steuerungsrechnern wird üblicherweise mit Watchdog-Schaltungen sichergestellt. Hierbei muß sich der Rechner periodisch beim Watchdog melden, ansonsten wird das System rückgesetzt. Solch ein Watchdog ist im MAX691 integriert. Er wird durch einen High- oder Low-Pegel am Pin 11 des MAX691 aktiviert. Nach Aktivierung muß am Pin 11 periodisch (je nach Beschaltung von Pin 7 alle 1.6 Sekunden oder 100ms) eine Signalflanke auftreten; bleibt sie aus, wird der Super8 rückgesetzt. Aktiviert wird der Watchdog über eine Kombination von P32 und P33. Die Watchdog-Funktion läßt sich nicht mehr abschalten falls sie einmal aktiviert ist.Dadurch wird dem Programmierer keine Möglichkeit gegeben, den Watchdog zu "überlisten", in dem er ihn kurzfristig ausschaltet. Falls dann vergessen würde, den Watchdog wieder zu aktivieren, wäre die Überwachung auf Funktionstüchtigkeit des Rechners hinfällig.

Die Software

Resetfest programmieren - aber ohne EPROM-Programmiergerät.

Die inkrementelle Programmierumgebung FORTH wird von der Hardware optimal unterstützt. Das Super8-FORTH durchsucht in seiner Startroutine den 64 KByte Adreßraum vom Speicherende in Richtung niedrigere Adressen nach einem RAM und nach einem Bitimage in einem evtl. vorhandenen EPROM. Wird kein Image gefunden, werden die Sytemparameter des FORTH-Kerns aus dem ON-Chip-ROM an den Anfang des RAMs kopiert und die Task- und Variablenbereiche am Speicherende initialisiert. Ein Bitimage im EPROM wird an einer speziellen Bitkombination erkannt. In diesem Fall werden die Systemparameter des hier abgespeicherten Systems zur Initialisierung des RAMs verwendet. Im Handbuch [3] des Super8-FORTH ist dieser Sachverhalt ausführlich beschrieben. Mit dieser Methode hat Klaus Kohl, für ein ROM-fähiges FORTH, eine elegante Lösung zur Verwaltung von variablen Speicherbereichen im RAM gefunden. SAVE-RAM (siehe beigefügten Quelltext) speichert nach Aufruf den aktuellen Zustand ab und gibt die RAM-FENCE und die höchste Adresse des Programms an. Die neue RAM-FENCE wird dann mittels Jumper eingestellt. Eine Warnung erhält man, falls das Programm nicht mehr in den vorhandenen Speicher paßt.

Braucht man z.B. für das zu entwickelnde Programm den S8-Assembler [3], so kann man diesen in das RAM laden, und mit SAVE-RAM und Umstecken der Jumper sichern. Bei einem Programmfehler, während der Entwicklungsphase von neuen Worten, der zum Absturz führt (Normalfall), braucht nach einem Hardware-Reset der Assembler nicht mehr neu geladen werden. Falls die neuen FORTH-Worte wie gewünscht funktionieren, können auch sie mit der oben beschriebenen Prozedur nichtflüchtig gespeichert werden.

Bedingt durch die Hardware (RAM-Größe 32 KByte) kann dieser Vorgang bis zu drei mal wiederholt werden, in dem man in 8 KByte-Schritten die RAM-FENCE nach oben verschiebt. Es stehen dann dem Super8-FORTH bis zu 24 KByte "ROM" im CMOS-RAM und mindestens 8 KByte RAM am Speicherende zur Verfügung. Die eventuell dadurch entstehenden Lücken im Speicher gehen zwar im Moment in der Programmentwicklung

verloren; sind jedoch die so entstandenen Worte ohne Fehler lauffähig, "jumpert" man auf der Platine RAM-FENCE auf die RAM-Startadresse (HEX 08000), lädt alle Teile zusammenhängend in den Super8 und sichert das System wie beschrieben. Beim Neustart des Super8-FORTH wird bei der Initialisierung immer der zuletzt gesicherte Systemzustand gefunden und als aktuelles System in Betrieb genommen.

Es ergibt sich so eine Bottom-Up-Strategie, bei der aufbauend auf Basisfunktionen neue Worte hinzugefügt und nichtflüchtig gespeichert werden, bis das fertige Programm erstellt ist.

Eine Art Hardware-FORGET ist möglich, in dem man mit den Jumpern eine RAM-FENCE einstellt, unterhalb der bereits ein System gesichert worden ist. Nach Reset oder COLD arbeitet man mit dem FORTH-System, das in diesem Bereich abgespeichert wurde.

An einem Beispiel wird die Programmierung des auf der Platine befindlichen Watchdog gezeigt. Dieser Chip erzeugt beim Einschalten der Platine und beim Ablaufen des Wachtdog-Timers einen Reset-Impuls.

Die Aufgabenstellung ist, das FORTH-System in einen Zustand zu bringen, der es ermöglicht, daß bei einer Endlosschleife (das System hat sich "aufgehängt") ein Reset-Impuls generiert wird.

Der Watchdog wird in KEY? und EMIT? nachgetriggert. Um das Auslösen eines Reset-Impulses zu verhindern, dürfen Worte, die KEY? oder EMIT? nicht aufrufen, keine längere Laufzeit als 1,6 Sekunden haben. Dies hat den Effekt, daß die serielle Schnittstelle mindestens alle 1,6 Sekunden bedient werden muß.

Mit WDOG_ON wird der Wachhund scharf gemacht und mit WDOG oder in diesem Beispiel in KEY? und EMIT? immer wieder beruhigt. In einem Multitask-FORTH-System wäre es sicherlich sinnvoll WDOG auch in PAUSE einzubauen. Die Phrase

```
' START_WDOG IS 'COLD
```

in Screen 6 klinkt START_WDOG in COLD ein und SINGLETASK belegt PAUSE mit NOOP. Beim Ausführen von COLD wird dadurch mit START_WDOG das Scharfmachen des Wachhundes veranlaßt, sowie KEY und EMIT auf die neuen I/O-Vektoren umgeleitet.

Mit dem Wort MS kann die Funktion überprüft werden. MS hat eine Laufzeit von ziemlich genau einer Millisekunde. Durch Probieren läßt sich feststellen, daß der Watchdog ab ca. 1,6 Sekunden einen Reset-Impuls generiert.

Diesen Artikel verstehen die Autoren als einen Beitrag zum Rapid Prototyping. Wir wollten zeigen, daß die On-Chip- und On-Board-Hardware schnell und ohne viel Aufwand in FORTH und Assembler programmiert werden kann.

Literaturverzeichnis

[1] Super8 Microcontroller mit 8-Bit-Archi tektur und Hochsprachen-Unterstützung, Werner Meininger, Design&Elektronik, Nr. 5/1986

[2] Super8 Microcomputer Product Specification, 1987
Super8 Microcomputer Technical Manual, 1987
Z8 Family Design Handbook, 1989
Zilog Inc. 210 Hacienda Ave., Campbell, California 95008-6609

[3] Handbuch zum Super8-FORTH V1.0, 1991, Klaus Kohl, Heinz Schnitter

[4] Datenblatt der Super8-Platine, 1991, Hans-Günther Willers

[5] Datenblatt MAX691, MAXIM Databook, 1990

```
Screen # 1

\ RAM-FENCE?                                        hfs 11:33 18.08.91
HEX
| : RAM-FENCE?  ( length -- t|f )
    BASE PUSH HEX                                   \ save BASE
    RAMORG @ +                                      \ höchste Adresse
    DUP  0E000 U<                                   \ zu lang?
    IF   0A000                                      \ erste Möglichkeit
         BEGIN OVER OVER U< NOT
         WHILE 2000 +                               \ 8KB Schritte
         REPEAT CR ." RAM-FENCE: " U.    TRUE
    ELSE 7 EMIT CR ." Programm zu lang " FALSE
    THEN SWAP   CR ." Prog. End: " U.
  ;
```

```
Screen # 2

\ SAVE-RAM                                          hfs 11:33 18.08.91
HEX
: SAVE-RAM  ( -- )
  RAMORG DUP 2- SWAP @  50 CMOVE    \ Zustand speichern
  HERE        RAMORG @  -           \ Programmlänge
  TASKORG   @ VDP    @  - DUP >R +  \ + Variablen- + Heap-Länge
  0D0         TASK#  @  - 8 *    +  \ + 80*Tasks = Gesamtlänge
  RAM-FENCE?                        \ paßt es ins RAM?
  IF   VDP @ HERE R@ CMOVE          \ Variablen und Heap
       HERE R> +                    \ RAM-addr
       LASTTASK  0CF  TASK# @
       DO @ DUP >R OVER 80 CMOVE    \ eine Task um die andere
          80 + ( RAM-addr ) R> 1+
       10 +LOOP DROP DROP
  ELSE RDROP THEN ;
```

```
Screen # 3

\ wdog_on wdog                                      hfs 11:33 18.08.91
HEX
  CODE WDOG_ON
       P3 , # 04 OR,              \ P32 High WDOG einschalten
       NEXT,
  END-CODE

  CODE WDOG
       P3 , # 04 XOR,             \ P32 toggeln WDOG streicheln
       NEXT,
  END-CODE
```

```
Screen # 4

\ wd_emit? (wd_emit                              hfs 11:37 18.08.91
HEX
  CODE WD_EMIT?
       P3  , # 04 XOR,              \ P32 toggeln WDOG streicheln
       AX PUSHW         AL , UTC LD,
       AL , # 02 AND,   Z  , 1$ JP,
       AX , # -1 LDW,   NEXT,
  1$:  AX , # 00 LDW,   NEXT,
  END-CODE

  CODE (WD_EMIT
       UIO , AL LD,
       AX POPW          NEXT,
  END-CODE




Screen # 5

\ wd_key? (wd_key                                hfs 11:37 18.08.91
HEX
  CODE WD_KEY?
       P3  , # 04 XOR,              \ P32 toggeln WDOG streicheln
       AX PUSHW         AL , URC LD,
       AL , # 01 AND,   Z  , 1$ JP,
       AX , # -1 LDW,   NEXT,
  1$:  AX , # 00 LDW,   NEXT,
  END-CODE

  CODE (WD_KEY
       AX PUSHW
       AL , UIO LD,
       AH , # 0 LD,
       NEXT,
  END-CODE



Screen # 6

\ wd_key wd_emit                                 hfs 11:33 18.08.91
HEX
  : WD_KEY  BEGIN WD_KEY?  0= WHILE  PAUSE  REPEAT (WD_KEY  ;
  : WD_EMIT BEGIN WD_EMIT? 0= WHILE  PAUSE  REPEAT (WD_EMIT ;

  INPUT:  WD_INPUT  WD_KEY?   WD_KEY  ;
  OUTPUT: WD_OUTPUT WD_EMIT?  WD_EMIT ;

  : START_WDOG
    WDOG_ON WD_OUTPUT WD_INPUT ;

  : MS 0 DO 1F 0 DO LOOP LOOP 7 EMIT ;

  ' START_WDOG IS 'COLD   SINGLETASK    DECIMAL
```

ModuNORM CPU's, Controller, UR/FORH für Windows

Produktbesprechung zur Systems '91 Forth-Systeme Flesch

ModuNORM CPU SAB 80C166

Das 50x80 mm^2 große ModuNORM™ 80C166 Modul ist hervorragend für Anwendungen im Echtzeitbereich geeignet, da das Interruptverhalten sowie die gute Peripherieunterstützung auf der CPU sehr leistungsfähige Systeme erlauben. Das Modul bietet hochflexible I/O Ports, 2 Serielle Schnittstellen, Zehn 10-Bit A/D Kanäle, Echtzeituhr, 40 MHz Quarzoszillator und 100 ns CPU Zykluszeit. Neben dem MPU (100 Pin Quad-Flat-Pack) ist auf dem Modul 64 kByte SRAM, RTC, RESET- und Batterie-Bakkup-Logik mit Spannungsüberwachung für RAM und RTC, Bus-Pufferung, Adressdekodierung, sowie 2 Stecksockel für maximal 256kByte EPROM untergebracht, von denen 192 kByte ROM adressiert werden können. Erhältlich ist das Modul in zwei Variationen (mit und ohne Multiplex) mit jeweils 16-Bit-Datenbus. Dazu Software-Entwicklungsumgebung Metacompiler, SwissFORTH.

ModuNORM CPU 64180

Das ModuNORM CPU 64180 Modul (50x80 mm) besitzt controllerseitig zwei unabhängige asynchrone (ASCII) sowie eine synchrone serielle Schnittstelle (CSI/O) und zwei programmierbare 16-Bit-Reload-Timer (PRT). DMA wird mit einem internen DMA-Controller (DMAC) unterstüzt. Mit Hilfe des internen Wait-State-Generators lassen sich Wait-States für Memory- und I/O-Zugriffe erzeugen. Die Memory Management Unit (MMU) des Controllers übernimmt die Umsetzung des logischen Adressraumes von 64 kByte in den physikalischen Adressraum von 1 MByte. Die Adressrechnung erfolgt intern parallel zu anderen CPU-Operationen. Ein Z80-PIO-Baustein stellt zwei bidirektionale 8-Bit-Ports mit der Möglichkeit des Handshakes zur Verfügung. Adress- und Datenbus sind vollständig gepuffert, so daß der große Speicherbereich extern gut erweiterbar ist. Auf dem CPU-Modul befinden sich maximal 126 kByte EPROM und 32 kByte statistisches RAM. Die Datensicherung geschieht über ein Batterie-Backup. Die Quarzfrequenz beträgt 12,288 MHz, was einer internen Taktfrequenz von 6,144 MHz entspricht. Eine Echtzeituhr 72423 sichert die Verfügbarkeit von Uhrzeit und DAtum auf dem CPU-Modul und läßt die gewünschten Zeitabhängigkeiten realisieren. Das Rücksetzen des Mikrocontrollers geschieht definitiv mit Hilfe eines Reset-Bausteins DS 1232 (Watchdog) und die Spannungsüberwachung über einen Baustein DS 1210. Die Stromaufnahme liegt aufgrund der CMOS-Realisierung bei einer Betriebsspannung von nominell 5 V (+/-0,25 V) bei nur 50 mA, beim Batterie-Backup reduziert sich diese auf ca. 10 A.

Grafik LCD-Controller

Der Grafik LCD-Controller (180x56 mm) großes Modul in SMD Technologie. Der GLCD-Controller enthält einen V25 Microcontroller sowie einen 6255 Anzeigen-Treiber welcher 64 kByte Anzeigenspeicher unterstützt. Er kann alle gängen Dot-Matrix LCD's mit bis zu 720 x 480 Punkten mittels 1, 2 oder 4 bit Ansteuerung treiben.

ModuNORM Drucker-Controller

Der Drucker-Controller wird in einer 2 Zoll- und einer 4 Zoll-Ausführung zur Ansteuerung der Fujitsu high speed Thermodruckwerke FTP-421 (2 Zoll) bzw. FTP-441 (4 Zoll) hergestellt. Diese Druckwerke sind auch für Zweilagen- bzw. Etikettendruck erhältlich. Sie haben eine Auflösung von 6 Pixel/mm und eine Druckgeschwindigkeit von max. 250 Pixelzeilen/Sekunde. Die Drucker-Controller (108 x 62 bzw. 163 x 62 mm) können direkt am Druckwerk befestigt werden. Die Speisung erfolgt mit 5 Volt ca. 0,5 A für die Logik sowie mit 24 Volt max. 3 A bzw. 5 A für den Schrittmotor und die Heizelemente. Dank der Überwachung der Druckschwärzung wird bei großem Schwärzungsgrad einer Druckzeile automatisch die Druckgeschwindigkeit gesenkt, und damit der Strombedarf begrenzt.

UR/FORTH für Windows

Mit dem neuen UR/FORTH für Windows ist jetzt interaktive Programmentwicklung möglich. Funktionen von Windows zum Erstellen benutzerfreundlicher Applikationen können in vollem Umfang genutzt werden. Dazu gehört z.B. das dynamische Erzeugen von Dialogboxen. In naher Zukunft wird vol allem die Fähigkeit zur dynamischen Datenkommunikation mit anderen Anwendungen (Spreadsheets, Datenbanken) von Bedeutung sein, da es immer mehr Anwendungen gibt, die diese Technik unterstützen. Selbstverständlich wurde die volle Kompatibilität zu den UR/Forth-Versionen unter MS-DOS, OS-2 und der Version für den 80386 gewahrt. Die von diesen Programmen gewohnten Debugging-Hilfsmittel sind erweitert worden, wobei die Vorteile der Bedienoberfläche zum Tragen kommen. Besonders hervorzuheben ist die Tatsache, daß die komplette Dokumentation auch als Windows-Hilfe (also Online) zur Verfügung steht. Beschleunigt werden durch die Coprozessorunterstützung mathematische Operationen. Alternativ dazu steht auch ein Software-Gleitkomma-Paket zur Verfügung. Prägnante Beispielprogramme und diverse Utilities runden das Paket ab.

Zu sehen in München
21-26. Okt. Stand A5, Halle 20

--FORTH-Gruppen--

FORTH-Gruppen:

W-1000 Berlin
Claus Vogt
Tel. 030/2 16 89 38
Treffen jeden Samstag
14 Uhr, Moerser Arbeits-
losenzentrum, Donaustr.1

W-4130 Moers 1
Friederich Prinz
Tel. 02841/5 83 98
Treffen nach Absprache

Gruppe Rhein-Ruhr:
Jörg Plewe
Tel. 0208/42 35 14
W-4000 Düsseldorf
Gebäude des S-Bahnhof
Derendorf,
Münsterstraße 199
Treffen jeden ersten
Sonnabend im Monat

W-6800 Mannheim
Thomas Prinz
Tel. 06271/28 30
Ewald Rieger
Tel. 06239/86 32
Treffen jeden ersten
Mittwoch im Monat im
Vereinslokal desSegel-
vereins Mannheim e.V.
Flugplatz, Mannheim-
Neuostheim

W-7000 Stuttgart
Wolf-Helge Neumann
Tel. 0711/88 26 38
Treffen nach Abspache

O-Leipzig
FORTH-Gruppe Leipzig:
Michael Balig, Lützner
Plan 17
O-7033 Leipzig
Dr. Jürgen Hesse
Tel. 041/69 56 02
Liselotte-Herrman-Str. 40
O-7050 Leipzig

FORTH für Ratsuchende:

Jörg Staben
Tel. 02103/5 56 09
dienstags und freitags
von 20.00-22.00 Uhr

Frank Stüss
Tel. 06187/9 15 03

Karl Schroer
Tel. 02845/2 89 51

Andreas Findewirth
Tel. 05221/2 35 04

Andreas Jennen
W-1000 Berlin, UUCP

Jörg Plewe
Tel. 0208/42 35 14

FORTH Fachgruppen:

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

FORTH Interessengebiete:

volksFORTH/ultraFORTH
Klaus Kohl
Tel. 08233/3 05 24
Klaus Schleisiek-Kern
Tel. 040/2 20 25 39

32-BIT Systeme
Robert Jones
Tel. 02434/45 79

Künstliche Intelligenz
Ulrich Hoffmann
Tel. 0431/67 88 50

NC4000 Novix Chip
Klaus Schleisiek-Kern
Tel. 040/2 20 25 39

Relationale Netze
HS/Forth
Künstliche Intelligenz
Realtime
Wigand Gawenda
Tel. 040/44 69 41

Gleitkomma-Arithmetik
Andreas Döring
Tel. 0721/59 39 35

32FORTH
Rainer Aumiller
Tel. 089/6 70 83 55

PostScript/FORTHscript
Christoph Krinninger
Tel. 089/ 7 25 93 82

FORTH im Unterricht
Rolf Kretschmar
Tel. 0240/43 90

Objekt-orientiertes FORTH
Christoph Krinninger
Tel. 089/7 25 93 82
Ulrich Hoffmann
Tel. 0431/67 88 50

F-PC Zimmer FORTH
ASYST Meßtechnik
Arndt Klingelnberg
Tel. 02404/6 16 48
box:geo1:klingelnberg

FORTH Fachgruppengründung

Grafik, Arithmetik
W-7000 Stuttgart 80
Jörg Tomes
Tel. 07 11/7 80 22 93
nur am Wochenende

FORTH Gruppengründung:

W-3300 Braunschweig
Martin Holzapfel
Tel. 05 31/35 12 62
32-Bit Systeme

W-2000 Hamburg
Wiegand Gawenda
Tel. 040/44 69 41
Treffen jeden ersten
Dienstag im Monat
Themen und Treffen
nach Absprache

FORTH-MAILBOX

SYSOP

Jens Wilke
Tel. 089/8 71 45 48
300-2400 baud
Parameter 8 N 1

FORTH-SYSTEME

ANGELIKA FLESCH

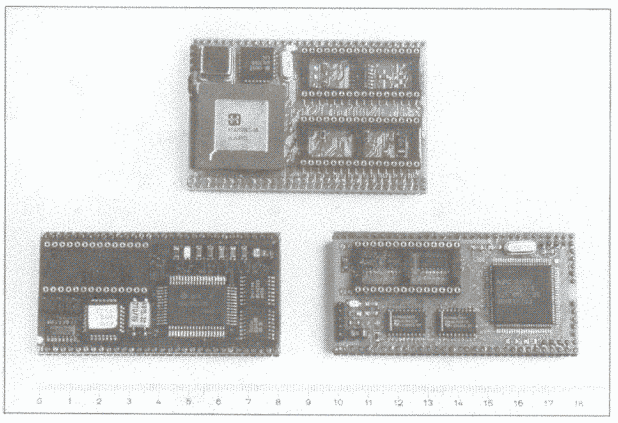
Postfach 1103
D-7814 Breisach

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

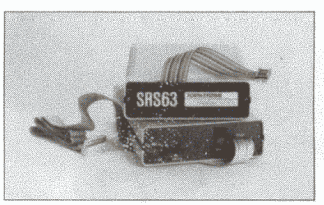
Entwickeln Sie romfähige Programme ?

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