Volume VI/Nr.2 - Juni 1990

VIERTE DIMENSION

Themen

- Realzeit mit KIROS
- CrossCompiler für FORTH
- Lokale Variablen
- Schnelles 68000-FORTH
- Datenübertragung von Commodore auf PC

FORTH MAGAZIN

7,50 DM

Rhf
REILHOFER KG

STRESS2 — Echtzeit-Erfassungs- und Berechnungs-System für Materialermüdung

Das System für

- **die Erprobung von Konstruktionsteilen in der PKW- und Nutzfahrzeugindustrie.**
  Nutzen: Reduzierung der Versuchszeiten durch frühzeitige Antworten aus der Erprobung.
  Weit kürzere Testläufe bei der Simulation des Fahrbetriebs.
- **die Dauerüberwachung von stark beanspruchten Kraftwerkskomponenten oder Walzgerüsten in der Stahlindustrie.**
  Nutzen: das Auswechseln von Komponenten kann auf die technisch notwendigen und richtigen Intervalle in Abhängigkeit der Ermüdung reduziert werden.
- **die Entwicklung.**
  Nutzen: Einsparung von Material.
  Einsatz von billigerem Material.
  Zeit- und Qualitätsvorsprung.

REILHOFER KG, Frühlingsplatz 9, 8047 Karlsfeld, Tel.: 08131/92059, FAX:08131-97447

Aktion: RTX für Einsteiger

kleinerMUCK

Grundausstattung: 64 KB ROM, 64 KB RAM, 8 KB EEROM, Watch-Dog, V24-Schnittstelle, Single-User MUCK Forth 83
DM 2.400,– Incl. Handbuch. 10 % Rabatt für FG-Mitglieder

Optional: Floating-Point-Paket, Multitasker, batteriegepufferte Uhr, 256 KB RAM zusätzlich, 32 KB EEROM

DELTA t
Ulrich Hoffmann Marina Kern Klaus Schleisiek-Kern
Entwicklungsgesellschaft für computergesteuerte Echtzeitsysteme mbH
Telefon 040/229 64 41 · Uhlenhorster Weg 3 · D - 2000 Hamburg 76

EDITORIAL

Als Beilage finden Sie in dieser 'Vierten Dimension' eine Einladung zum FORTH-Workshop in Suhl (DDR). Er findet vom 7. bis 9. November statt. Ein interessantes Datum, ist doch der 9. November der 1. Jahrestag des Mauerfalls in Berlin. Vor einem Jahr wäre dieses Treffen wohl noch nicht möglich gewesen, heute ist es ganz normal - mit Recht. Wir freuen uns alle, daß damit ein weiterer Schritt zur Normalisierung des deutsch-deutschen Verhältnissen gemacht werden kann.

Und noch etwas zu diesem Thema: Die Beiträge aus der DDR werden immer zahlreicher - die aus der BRD immer weniger, vor allem Leserzuschriften erhalten wir kaum noch. Eine Zeitschrift lebt aber vom Dialog mit ihren Lesern. So müßte z.B. die FORTH-Tagung in Frankfurt genügend Anregungen geliefert haben. Schreiben Sie uns doch einfach einmal!

Für Bastler haben wir dieses Mal einen interessanten Beitrag zur Datenübertragung vom Commodore Plus 4 auf den IBM. Auch sonst bietet die vorliegende 'Vierte Dimension' genügend Stoff für ein paar Stunden vergnügliche Lektüre. In diesem Sinne Ihre Redaktion

Rainer Aumiller

Denise Luda

FORTH-Tagung '90 in Frankfurt

IMPRESSUM

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

Herausgeber:
FORTH-Gesellschaft e.V.

Redaktion:
D. LUDA Software, Gustav-Heinemann-Ring 42, 8000 München 83,
Tel. 089/670 83 55, FAX 089/679 22 71

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

Quelltextservice:
Der Quelltext von Beiträgen, die mit diesem Symbol gekennzeichnet sind, ist auf der Leserservice-Diskette zur jeweiligen Ausgabe oder in der FORTH-Mailbox in München Tel. 089/7259625 8N1 zu finden.

Quelltext Service

Autoren dieser Ausgabe:
Jörg Staben, Jörg Plewe, Marcus Werner, Ralf Neuthe, Jyrki Yli-Nokari, John R. Hayes, Markus Redeker, Robert Eppecht, Claus Vogt.

Erscheinungsweise:
Vierteljährlich

Redaktionsschluß:
Die zweite Woche im mittleren Quartalsmonat

Auflage:
ca. 1000 Stück

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

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

Für jedes eingesandte Manuskript sind wir sehr dankbar. Für die mit Namen oder Signatur des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Die in dieser Zeitschrift veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Vervielfältigung, Nachdruck sowie Speicherung auf beliebigen Medien ist allerdings auszugsweise mit genauer Quellenangabe erlaubt. Freie Mitarbeit ist erwünscht. Die Beiträge müssen frei von Ansprüchen Dritter sein. Veröffentlichte Programme gehen, sofern nicht anders vermerkt, in die Public Domain über. Für Fehler im Text, in Schaltbildern, Aufbauskizzen usw., die zum Nichtfunktionieren oder evtl. Schadhaftwerden von Bauelementen führen, kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

Vierte Dimension Inhalt

Leserbriefe und Zuschriften

FORTH public domain-POOL

Wir sind mit der aktuellen Situation von Tools und Quellen für FORTH unzufrieden und möchten versuchen, einen public domain-Pool – den PDP – aufzubauen und vor allem zu pflegen. Er soll unter anderem der Aktualisierung des volksFORTH für den PC dienen. Aber nicht nur PC-Nutzer sollen etwas zum PDP beitragen oder aus ihm abrufen können, sondern es sollen zugleich die Benutzer eines 680x0-Rechners wie zum Beispiel Amiga, Atari, VME angesprochen werden. Dieser Doppel-PDP soll zwischen der 68K und der Intel-Welt abgeglichen werden. Es ist weniger der 40 MegaByte umfassende und ungetestete Pool das erklärte Ziel sondern der konsistente Pool mit den schon immer vermißten Worten und dringend benötigten Tools geplant, damit nicht weitere Generationen von Interessenten FORTH aus der Hand legen, weil man so auf Anhieb keine Zahl einlesen kann. In der 'Vierten Dimension' wird dann immer vom aktuellen Pool-Inhalt berichtet.

Damit die Vereinheitlichung von Motorola und Intel funktioniert und überhaupt Motorola-Beiträge beigesteuert werden können, stellt Jörg Plewe sein F68K – ein Shareware-Produkt – zur Verfügung. Dies ist ein schnelles 68K-FORTH. Es generiert "native code" (Maschinencode), wird auf Basis des FFORTH-Kerns hardwareunabhängig entwickelt und soll auf allen 68K-Rechnern lauffähig sein. Bei der Portierung ist lediglich ein Lader neu zu schreiben, der sich dann um `EMIT`, `KEY` und `BLOCK` kümmert. Dieses FORTH wird in der dieser 'Vierten Dimension' vorgestellt; bis dahin soll auch der Kern fertiggestellt sein. Für Interessenten gibt es bei Jörg Plewe auf Anfrage einen Prototyp.

Es wird natürlich etwas dauern, bis erstmal die schon vorhandenen Beiträge gesichtet und eingefügt sind. Je nach Natur der Beiträge - wir selbst meinen, ein großes Problem im volksFORTH zu sehen und haben eine Lösung dafür erstellt - soll dann auch das volksFORTH selbst geändert werden und nur noch die neue Version verteilt werden.

Grundsätzliches zur Organisation: Wer etwas zuschickt, bekommt den aktuellen Pool-Inhalt zurück. Für das reine Abfordern wird sich auch ein Modus finden lassen. Darüberhinaus soll der Pool die weitere Arbeit mit FORTH organisieren und initialisieren, da dann wirklich übertragbare Lösungen für die beiden meistverbreiteten CPU-Typen existieren. Sollten sich jetzt User von zahlenmäßig nicht so verbreiteten Rechnern benachteiligt fühlen, wäre selbstverständlich jemand, der den PDP für z.B. CP/M oder den Archie aufbereitet, eine zusätzliche Bereicherung.

Nun eine Reihe von Vorschlägen, was in den Pool eingebracht werden könnte, wobei der Sourcecode auf Disk bequemer für uns ist!

- Gesucht wird ein 68000-Assembler in FORTH, optimal 68020 und höher, nicht den L&P-Assembler.
- Da viele Funktionen in FORTH zu zielorientiert sind, fehlen richtige Gigs (überflüssige, aber beeindruckende Leistungsmerkmale). So scannt das TP-Programm eines meiner Bekannten die Platte nach einem TREEINFO.NCD und nutzt es oder baut selbst den Baum auf. Daraufhin kann man auf dem Baum-Bild DIRs markieren und Operationen auf (Sub)Directories machen, z.B. alle FORTH-Subs 'zippen'. Diese Philosophie zugrunde legend könnte man dem FORTH-Nutzer anbieten, seinen Pfad zu setzen, indem er im TREE markiert, für welche Bereiche der Pfad gesetzt werden soll. Zweifelsfrei geht's auch mit PATH und tippen, aber *das andere* ist halt Service!.
- Ganz wichtig: Zugriff auf C-Bibliotheken wie z.B. WINDOW BOSS, denn auch FORTH-Einsteiger wollen die Fenster tosen sehen oder auch mal für eine schnelle Sache auf fertige Lösungen zurückgreifen.
- Als aktuelle Herausforderung: EOS 10 Strichcode. Mein neuer Fotoapparat nimmt die Programmeinstellung der Automatikfunktionen über Strichcode vor, ähnlich wie im Supermarkt an der Kasse die Eingabe der Preise per Bar Code. Nach meinem Dafürhalten ist die Kamera damit frei programmierbar bzw. die internen Programmelemente könnten individuell eingesetzt werden, wenn man die Bar Codes selbst mit dem Drucker machen könnte. Wer kennt sich mit Bar Code aus und kann ihn interpretieren? Denn das wäre etwas Spektakuläres - die Kamera ist neu auf dem Markt und FORTH wäre in aller Munde, wenn man sich mittels eines FORTH-Programmes selbst seine Codes drucken könnte.
- Der Umgang mit den und die Vorteile der Headerfunktionen.
- Bücher könnten aufgearbeitet werden: Townsend/Feucht, Dick Pountain, R.Zech
- Ein "natural Language"-Interface: nach dem Kommando `SYNTAX ON` kann man `RDOP` eingeben und FORTH sagt: "`RDROP erkannt`"; da gab es mal in der MC etwas.
- Anderssprachige Beiträge aus Zeitschriften/Literatur allgemein aufarbeiten; so gab es in der letzten c't einen kleinen Beitrag über OOP mit einem Beispiel, wie man einen generischen Typ *Liste* handhabt. Dies wäre der Anschlußbeitrag zur unserer Lösung der Probleme mit `DECODE` (`DECODE`, denn dafür wollen wir auch Listen einsetzen). Das könnte doch für die OOP-Arbeitsgruppe mal ein Thema sein; der freudige FORTH-User prüft dann direkt im Vergleich, ob er denn nun endlich auf TP5.5 umsteigen soll.
- Der volksFORTH-Debugger nutzt den Disassembler nicht, sonder sagt nur "not traceable".
- Damit es nicht so ernst wird: Ein Spiel wie z.B. ein Wurm frißt Zahlen, wird immer länger. Wenn er den Rand oder sich selbst berührt, ist das Spiel zu Ende.

- Oder ein einfaches Invaderspiel, um den Multi-Tasker zu zeigen, wie es mal in den Toolboxen von TP war.
- Die ganze Handhabung von Blöcken: Editieren, Verschieben, Löschen, Einfügen etc. soll zusammen mit einer neuen Blockgröße von 2K(!) zu einem Tool zusammengefaßt werden - dem BLOCKMASTER. Dies soll dann der FORTH-Übereditor werden.

Jörg Staben

ANS-FORTH - in statu nascendi?

Ich verfolge die Diskussion um die Entwicklung des neuen FORTH-Standards mit großem Interesse und fand den Artikel von John Hayes aus den FORTH Dimensions (VD 4/89 S.17ff) sehr interessant. Hoffentlich folgen bald weitere Artikel zu diesem Thema. (Übrigens: Alle reden vom 83-Standard. Ist er eigentlich irgendwo schriftlich fixiert? Wenn ja, wo kann man eine Ausfertigung bekommen?).

In einem früheren Beitrag (VD 3/87 S.17ff) wurde auf Vorbehalte gegenüber einer zu starken Standardisierung von FORTH hingewiesen. Ich denke aber, daß bis zu einem gewissen Grade die Vorteile von Standardisierung und Flexibilität einer Sprache sich gegenseitig ergänzen, nicht sich ausschließen. Wirft man einen vergleichenden Blick auf die Sprache C, so kann man erkennen, daß ihr Kern recht einfach und überschaubar ist. Leistungsstark wird ein C-Compiler für den Programmierer erst durch eine große mitgelieferte Funktionsbibliothek.

Nun gut, FORTH ist ein ganzes Stück flexibler als C. Dennoch halte ich eine Art Modul- oder Paketlösung im Rahmen eines FORTH-Standards für erstrebenswert. Was sollte in einem solchen Paket zusammengefaßt werden? Nun, sicherlich sollte es Pakete nur für abgrenzbare Problembereiche geben. Für diesen Bereich sollte es einen Mindestbestand an Worten festlegen und deren Funktion definieren. Die Implementierung dieser Worte und die Hinzufügung weiterer Worte bliebe dem Entwickler des FORTH-Systems überlassen. Wenn für das Gebiet noch keine ausgereiften, allgemein anerkannten Lösungen existieren, würde ein solches Paket vielleicht nur eine Handvoll Worte umfassen. Der Wortschatz kann ja im Rahmen der weiteren Entwicklung problemlos ausgeweitet werden (Aufwärtskompatibilität). Die Worte eines solchen Paketes gehören dabei nicht selbstverständlich zum Lieferumfang eines FORTH-Systems. Wenn sie dazugehören, dann alle und mit der exakten Funktionsspezifikation, damit sich die Anwender des Paketes darauf verlassen können, daß einmal entwikkelte Lösungen auf allen FORTH-Systemen laufen, die über das gleiche Paketsortiment verfügen.

Konkret: Nicht jedes FORTH-System hat oder braucht Worte zum Rechnen mit Fließkommazahlen. Wenn aber ein solches angeboten wird, dann möchte ich mich darauf verlassen können, daß ganz bestimmte Funktionen (z.B. Logarithmus, Sinus, usw.) auch zur Verfügung stehen und ich sie unter festgelegtem Namen mit exakt definierter Stackübergabe aufrufen kann. (Siehe VD 1/90 S.13ff: Die unter den Punkten 1 bis 10 beschriebenen Worte sollten z.B. zu einem FP-Paket gehören.). Nun kann man den Problembereich Fließkommarechnung besonders gut überschauen, so daß hier sicherlich ein umfangreicher Wortschatz zum Paket (mit evtl. Unterpaketen) gebündelt werden könnte.

Vergleichbare Standardpakete (wenn auch vielleicht von geringerem Umfang) ließen sich wohl auch für andere Datentypen spezifizieren: Aufzählungen, Mengen (sets, bags), komplexe Zahlen, Felder (arrays, rows), Verbunde/Tupel (records, structs), Listen, Bäume usw. (vgl. auch VD 3/89 S.26ff).

Gibt es verschiedenartige Lösungen einer Problemstellung vergleichbarer Qualität, wie zum Beispiel bei der CASE-Kontrollstruktur (siehe auch z.B. VD 2/89 S.15ff; VD 3/89 S.14ff), so gehören sie gemeinsam in ein Paket. Durch eine standardisierte Namensgebung ist dann sicherzustellen, welches CASE ein Anwender aufruft.

Ein weiteres vielversprechendes Feld für Paketdefinitionen wäre sicherlich der Ein-/Ausgabebereich. Gerade Basic-Routinen für die zeilenorientierte Bildschirmkommunikation (seit jeher aus Basic oder Pascal bekannt) werden doch manchmal vermißt. Von Worten für die Arbeit mit Menüs oder einfachen Bildschirmmasken ganz zu schweigen. Auch elementare, aber leistungsfähige Grafikroutinen gehören hierher (siehe "Hasengrafik", VD 4/89 S.15ff).

Schließlich sei noch die Verwaltung von Dateien genannt (sequentiell, ISAM, BTree usw.) mit Worten zum Suchen, Einfügen, Löschen, Sortieren von Datensätzen. Sehr weitgehend wäre schon die Entwicklung von Paketen für numerische Standardverfahren wie FFT oder Polynominter- und -extrapolation, statistische Auswertungen usw.

Ein solcher paketorientierter Standard bräuchte sicher eine lange Zeit zur Entwicklung. Aber es kann ja auch stufenweise Paket für Paket entwickelt und weiterentwickelt werden. Ich meine jedenfalls, daß ein solches Konzept der Verbreitung von FORTH sehr förderlich sein würde.

Andreas Findewirth

Termine

aus Forth News Berlin 2 3/90

Do, 28.06.90:
FORTH als Produktivkraft
(G. Blanke, Mikrotaurus/Berlin)

Warum benutzen professionelle Software-Entwickler FORTH? Über die Stärken - und auch die Schwächen - der Sprache und ihre verschiedenen Programmierumgebungen bei der Erstellung von Anwendungen nicht nur der Meß- und Steuertechnik wird G. Blanke anhand langjähriger Erfahrungen seiner Firma vortragen.

Do, 26.07.90:
Sommerpause?
(N.N., Berlin)

Weitere Termine und Treffen können bei Claus Vogt, Bülowstr. 67, 1000 Berlin 30 erfragt werden.

Notizen aus der Provinz

Berichte aus den Gruppen

Für den 06. März 1990 hatte ich nach Herford eingeladen zur Gründung einer FORTH-Regionalgruppe in Ostwestfalen-Lippe. Immerhin zwei Interessierte erschienen auch und wir saßen dann abends gut zweieinhalb Stunden in angeregter Unterhaltung zusammen. Am Schluß einigten wir uns jedoch darauf, von der Bildung einer Gruppe und der Durchführung regelmäßiger Treffen abzusehen. Zum einen waren die Interessenschwerpunkte recht unterschiedlich, zum anderen liegen zwischen der mir in der Region bekannten Forthianern Entfernungen bis zu 50 km. Es wird also bis auf weiteres bei sporadischen Treffen, Einzelbesuchen und Telefonaten bleiben.

Das mir ursprünglich vorschwebende Konzept einer FORTH-Lokalgruppe aus 5-7 Aktiven, die aus ihrer Mitte heraus vielleicht sogar FORTH-Kurse an den hiesigen Volkshochschulen anbieten könnten, läßt sich leider in der FORTH-Diaspora nicht realisieren.

Ich stehe aber weiterhin gerne in meiner Region als *Ansprechpartner* für alle zur Verfügung, die sich mit FORTH beschäftigen wollen. In diesem Zusammenhang will ich dann gleich darauf hinweisen, daß in der Stadtbücherei Herford immerhin drei gute FORTH-Bücher vorhanden sind: "Starting FORTH" und "Thinking FORTH" von Leo Brodie in ihren deutschen Übersetzungen sowie "FORTH - Ein Programmiersystem ohne Grenzen" von A. Goppold und R. Bouteiller.

Andreas Findewirth

Bericht von der Jahrestagung 1990 der FORTH-Gesellschaft e.V.

von Jörg Staben und Jörg Plewe

Einleitung

Die Jahrestagung der FORTH-Gesellschaft e.V. fand diesmal am 6. und 7. April in Frankfurt statt. Der Physikalische Verein der Universität Frankfurt hatte die Tagungsräume zur Verfügung gestellt, die perfekte Organisation wurde von der Frankfurter Gruppe um Frank Stüss geleistet.

Dank der Unterstützung des Deutschen Akademischen Austauschdienstes (DAAD) war die Tagung von einer Vielzahl an Fachbeiträgen aus der DDR geprägt. Diese Vielzahl sorgte für ein volles Tagungsprogramm, dessen Hauptthemen traditionell die Anwendungen von FORTH und Aspekte der Spracherweiterung waren.

Herr Peter Grabienski von der Universität Dortmund eröffnete die Tagung mit seinem Vortrag: FLIP - FLOP: Ein Stack-orientiertes Multiprozessorsystem. Es handelt sich hier um einen Stack-orientierten Prozessorkern, mit vier Hochgeschwindigkeits-Links, alles zusammen integriert auf einem Chip. Der 32-Bit breite Prozessorkern erreicht bei einer Taktrate von 10 MHz mehr als 10 Millionen FORTH-Instruktionen pro Sekunde. Die Kommunikationsperipherie (Byte-parallel) arbeitet unabhängig vom Prozessorkern mit einer Datenrate pro Link von 10 MByte/sec. Alle vier Links sind über Busse miteinander und mit einem Interface verbunden. Über dieses Interface hat der Prozessor Zugang zur Außenwelt. Im Vortrag wurde gezeigt, daß die Stackarchitektur gut mit dem auf Nachrichten basierenden Kommunikationsprinzip harmoniert und daß hohe Leistungswerte bei geringem Chipflächenverbrauch erreichbar sind.

Prolog-Erweiterungen

Diese Spracherweiterungen zielten - vielleicht zur Überraschung einiger 'modebewußter' Programmierer - nicht in die Richtung der objektorientierten Programmierung, sondern auf das "Programmieren in Logik", PROLOG. Allein von der Technischen Hochschule Ilmenau lagen mehrere Beiträge vor, in denen gezeigt wurde, wie unter Berücksichtigung der logischen Programmierung die Leistungsfähigkeit der Sprache FORTH noch weiter ausgebaut werden kann.

Besonders deutlich wurde dies an einem Beitrag, in dem die Steuerung eines Transportroboters vorgestellt wurde, der eigenständig das klassische Problem der Hindernisumgehung löste. Diesem Lösungsansatz liegt das Konzept einer virtuellen Doppel-CPU zugrunde, in dem parallel zur gemeinhin bekannten FORTH-Maschine eine Inferenz-Maschine arbeitet. Während FORTH für die problemlose Steuerung des Roboters verantwortlich ist, tritt die Inferenzmaschine dann in Aktion, sobald nicht vorhersehbare Zustände

oder Ereignisse auftreten. Dieser Ansatz verspricht interessante Erkenntnisse über die Kombination einer imperativen mit einer logischen Sprache, wobei der Einsatz von FORTH-Prozessoren wie dem Harris RTX-2000™ auf diesem Gebiet noch entscheidende Laufzeit-Verbesserungen bringen wird.

Programmierung verteilter Systeme

Diese regelbasierte Programmierung findet sich auch in der Programmierung verteilter Systeme wieder.

Solche verteilten Systeme ergeben sich aus der Anforderung von automatischen Steuerungen, wo in den Anlagen vor Ort kleine Rechner eingesetzt werden. Diese haben dort spezifische Aufgaben zu erfüllen, arbeiten eigenständig oder sind mit einem übergeordneten Rechner gekoppelt. Für die Entwicklung der Programme des Vor-Ort-Rechners müssen im ungünstigsten Fall "probeweise" EPROMs programmiert werden, deshalb bevorzugt man die Möglichkeit, zumindest zeitweise mit einem für Entwicklungszwecke geeigneten Rechner Informationen an den Vor-Ort-Rechner zu übermitteln. So läßt sich mit dem Einsatz eines verteilten FORTH-Systems der Entwicklungskomfort drastisch erhöhen.

In diesem Zusammenhang war der Beitrag von Dr. K.Kabitzsch von der Technischen Hochschule Leipzig von besonderem Interesse, in dem das System PROCESS-FORTH vorgestellt wurde. PROCESS-FORTH läuft auf Klein- und Kleinstrechnern, so daß mehrere dieser Rechner gemeinsam zur Steuerung von Landmaschinen (z.B. Mähdreschern) eingesetzt werden können. Dieses System ist gleichzeitig ein gutes Beispiel für die flexible Nutzung von FORTH-Quelltextformaten; es ist in der Lage, Quasigrafiken in Form von Kontaktplänen, wie man sie z.B. in der Starkstromtechnik findet, zu übersetzen. Damit wird FORTH in gewisser Weise auch Nicht-Programmierern zugänglich.

EuroFORML'90

Large Systems
(Forth in Control in the 1990's)

October 12-14th 1990

Call For Papers

Suggested Subject Headings Are:

Connectivity, Multi-processor Systems, Distributed Systems, Project Management, Team Programming, Techniques and Tools.

Please let us know as soon as possible if you would like to speak. Abstracts should be submitted by August 12th and papers, camera ready, by September 12th.

The venue:

The Potters Heron
Ampfield
Hampshire

Situated on the edge of the picturesque New Forest, this extensive thatched hotel offers all modern facilities. The famous Broadlands and Beaulieu stately homes are only a short distance away as are the award winning Exbury and Hillier Gardens. Sample the ancient splendour of the historic Winchester and Salisbury Cathedrals or try a traditional New Forest Cream Tea.

A Demonstration and Exhibition area is available- please contact the Conference Organiser for and information sheet.

All communications to:

The Conference Organiser
EuroFORML'90
133 Hill Lane
SOUTHAMPTON SO1 5AF

Tel: (+44) (703) 631441

Quelltextverwaltung

So wie die Möglichkeit grafischer Quelltexte auf neue Konzepte in der Programmierumgebung hindeutet, nimmt die Unterstützung des Programmierers durch das Programmiersystem zunehmend breiteren Raum ein.

Wegbereitend ist hier die Firma Delta-T mit dem Konzept der Screens, in denen die Begrenzung der Quelltextblöcke auf 1K aufgehoben wird und der Quelltext als Streamfile vorliegt, in dem die Blöcke in beliebiger Länge vorliegen. Ein spezieller Editor ermöglicht dann das Blättern zwischen diesen Quelltextscreens.

Noch wesentlich weiter geht das Konzept des DFF-FORTH der Frankfurter Gruppe um Frank Stüss. Hier gibt es keine Quelltexte im klassischen Sinne mehr, die zur Programmentwicklung notwendigen Worte werden in einer indexsequentiellen Datenbankumgebung gehalten. Die Records dieser Datenbank sind die Quelltexte der einzelnen Worte, deren Anwender- und Implementationsdokumentation. So wird

ein Wort bearbeitet, indem man es aus einem Menü auswählt, statt es im Quelltext zu suchen.

Da nicht nur das Editieren, sondern auch das Dokumentieren von Programmen eine zeitaufwendige Arbeit ist, wurden auch Lösungen zur automatischen Dokumentation vorgestellt. Diese Lösungen werten nicht nur den Stackkommentar aus, sondern analysieren auch den Vokabularkontext und erstellen eine Art Crossreferenz-Liste.

Abschluß

Natürlich kann durch diesen kurzen Abriß kein vollständiges Bild der Tagung widergegeben werden. Neben den oben genannten Beiträgen gab es viele weitere, nicht minder interessante. Martin Tracy, prominenter Gast der Tagung und Mitglied im ANSI-Komitee, zeigte auf gekonnte Art den Stand des von vielen erwarteten ANSI-FORTH-Standards auf. Die Themen der anderen Vorträge reichten vom Menüsystem in FORTH 83 über den Einsatz von FORTH in der Pädagogik bis zur Schilderung der Aktivitäten der FORTH-Gruppen in der DDR. Wie schon fast Tradition auf den FORTH-Tagungen, wurden auch Techniken der Meta- und Targetcompilation angesprochen, durch die ein FORTH-System in der Lage ist, aus einem Quelltext Code für verschiedenen Prozessoren zu erzeugen.

Autor	Titel
P. Grabienski	FLIP FLOP: Ein stackorientiertes Multiprozessorsystem.
D. Heid, F. Raschke, F. Stüss	DFF: Eine neue Art zu programmieren.
R. Kretzschmar	FORTH als Einsteigersprache.
J. H. Schwalm	Hierarchisches Menüsystem in FORTH-83.
B. Ganahl	Aufbau eines Knotenrechners zur Steuerung supraleitender Spulen im Tritron.
U. Hoffmann	Von Blöcken zu Screens.
H. Finsterbusch, R. Großmann	FORTH an der TH Ilmenau - Projekte und Perspektiven.
H. J. Fuchs	Ein interaktives Bildverarbeitungssystem mit FORTH-Nutzeroberfläche.
A. Goppold	Leibniz: Die ersten sieben Jahre.
J. Tolkemit	Logische Programmierung in logFORTH.
C. M. Westendorf	Die Software-Architektur des Bildverarbeitungssystems IBT.
H. Rudolph	Anwendung von FORTH bei der Nutzerinterface-Entwicklung für einen interaktiven Reglerentwurfsplatz.
J. Pohl	Interaktiver Simulator für einen 32-Bit-FORTH-Prozessor.
C. Beckmann	FORTH-Decompiler als Einsteigerübung.
E. Woitzel	Glossare aus dem Quelltext.
K. Noack	FORTH-Spracherweiterung für KI.
R. Neuthe	Kommunizierende FORTH-Systeme.
M. Balig	Die Fachgruppe FORTH in der DDR.
K. Kabitzsch	Programmierung eines Kleinautomatisierungssystems in FORTH.
L. Karadshow	Programmtechnische Umgebung zur Realisierung flexibler und KI-gesteuerter Target-Compilation.
J. Hesse	Programmierung eines Fermentersteuersystems in FORTH.
T. Beierlein	Transformation Bool'scher Gleichungen in Kontaktpläne.
J. Bernhardt	FORTH für dezentralisierte EMR-Steuerungen.

Im Vorraum stellten die Firma FORTH-Systeme Flesch einige ihrer Produkte und Andreas Goppold seine Weiterentwicklung von FORTH, "LEIBNIZ", vor. Alle angesprochenen Themen waren durchweg auch dem interessierten Laien zugänglich. Hobbyisten konnten sich über die Vorträge hinaus in den Pausen und den Gesprächen und Diskussionen am Abend einen weiteren Einblick in die Kenntnisse der Freaks und Profis verschaffen.

Diese Tagung hat aufs Neue gezeigt, daß es nicht die Forderung nach Superrechnern und gigantischen Speichern ist wie in der UNIX-Welt sondern die intelligenten und eleganten Lösungen, was die Mitglieder der FORTH-Gemeinschaft verbindet. Besonders die Teilnehmer aus der DDR, wo schnelle Hardware noch Mangelware ist, konnten mit ihren Beiträgen in diesem Sinne begeistern.

Der Tagungsband, in dem ein Großteil der Beiträge detailliert wiedergegeben wird, kann über das FORTH-Büro bestellt werden.

Inserentenverzeichnis:

Firma _____ Seite der Anzeige

DELTA t Entwicklungsgesellschaft für computergesteuerte Systeme mbH, Hamburg _____ 2

Reilhofer KG, Karlsfeld _____ 2

MICROPROCESS GmbH, Schriesheim _____ 23

Klaus-Peter Schleisiek, 5100 Aachen _____ 25

EDV-Beratung - Software-Design - Goppold, Poing _____ 35

FWD-Team, Ludwig Richter, MZ-Bretzenheim _____ 35

Angelika Flesch, FORTH-Systeme, Breisach _____ 36

SKYLINE COMPUTER, 8042 Oberschleißheim _____ Beilage

Realzeit mit KIROS

von Dipl.-Inform. Marcus Werner

Einleitung

Das Programmieren von Steuerungen oder Regelungen, die Realzeitbedingungen genügen sollen, wird meist in der niedrigsten Art der Programmierung, in Assembler, realisiert. Dies erfordert ein intensives Einarbeiten in die Spezialitäten und Probleme eines jeden Prozessortyps und das neben der eigentlichen Problemanalyse. Es ergibt sich jedoch, bei entsprechendem Aufwand für Implementierung und Test ein schnell ablaufendes Programm.

Das Ziel der zugrundeliegenden Diplomarbeit war es, eine effiziente, schnelle und leichte Realisierungsmöglichkeit von Steuerungs- und Regelungs-Funktionen zur Verfügung zu stellen, die aber trotzdem Realzeitanforderungen genügen. Im Ergebnis wird hier eine quasi Hochsprache zur Verfügung gestellt, in der Probleme der Steuerungs- und Regelungsanwendungen auf eine direkte und einfache Weise angegangen werden können. So soll die Sprachstruktur z.B. die Kontrollstruktur `IF...THEN` und verschiedene Schleifenformen zur Verfügung stellen, Variablen und Konstanten-Definitionen ermöglichen. Der erstellte Code soll außerdem portabel sein, so daß mit diesem System realisierte Probleme auch auf anderen Rechnertypen sofort lauffähig sind.

Die implementierte Sprachstruktur ist FORTH 79 angelehnt. Dem prinzipiellen Realisierungskonzept der Sprache wurde gefolgt, nur ist der Umfang verkleinert worden, da hier speziell Steuerungs- und Regelungsanwendungen realisiert werden sollen. Der realisierte Sprachumfang des Systems kann beliebig in Bezug auf die gewünschten Bedürfnisse erweitert werden.

Das hier realisierte System stellt demnach nicht nur eine Programmiersprache zur Verfügung, die in ihrem vollen Umfang genutzt werden kann, sondern bildet eine vollständig eigene Entwicklungsumgebung für Regel- und Steuerungsfunktionen.

Das realisierte System besteht aus:

- dem Microcontroller Intel MCS 8096 bzw 80C196 mit externem Speicher,
- dem Monitor WERMON, der in erster Linie die elementaren 8096 (80C196) spezifischen Ein-/Ausgabefunktionen bereitstellt und
- der Rechnerstruktur KIROS, die einen virtuellen Rechner darstellt mit integrierten Funktionsentwicklungssystem.

Der vorliegende Beitrag stellt eine Kurzfassung der zugrundeliegenden Diplomarbeit dar und gliedert sich in zwei Hauptteile. Im ersten Hauptteil wird die Implementierung des Monitors WERMON beschrieben und im zweiten Hauptteil die Implementierung der Rechnerstruktur KIROS.

Stichworte

» FORTH,
» Realzeit,
» Microcontroller

Der Monitor "WERMON" für das Microcontrollersystem 8096 und MCS 80C196

Beschreibung des MicroController Systems 8096

Der MCS (MicroController-System) 8096 wurde 1983 von der INTEL Corporation, Santa Clara, CA, entwickelt. Er ist ein sehr schneller Microcontroller, der mehrere Funktionsbausteine auf einem Chip vereint, was seine Anwendung besonders vielseitig und einfach handhabbar macht. Der MCS 8096 verfügt über eine 16-Bit CPU, hat 232 Byte RAM und eine hardwaremäßig realisierte Multiplikation und Division. Er verfügt über verschiedene Timer und hat die Möglichkeit, parallele I/O-Operationen (über bis zu 5 verschiedene Ports) vorzunehmen. Weiterhin hat er eine voll-duplexe, serielle Schnittstelle und eine zusätzliche Hochgeschwindigkeits-I/O-Einheit. Weiterhin ist eine 10-Bit A/D-Konvertierung sowie die Erzeugung eines analogen Ausgangssignals (PWM) möglich. Der Befehlssatz berücksichtigt 6 verschiedene Adressierungsarten und zusätzlich ist die Interrupt-Struktur des MCS 8096 hinsichtlich der Priorität programmierbar.

Diese umfangreichen Möglichkeiten machen den Microcontroller zu einem universell einsetzbaren Baustein, der vorwiegend in der Motor-Steuerung, Robotik, Prozeßsteuerung, Instrumentensteuerung (z.B. Oszilloskope) und Druckersteuerung (Plotter, Impact und Non-Impact-Drukker) eingesetzt werden kann.

Anforderungen an den Monitor

Die Entwicklung eines Assemblerprogramms für einen Prozessor erfordert meist aufwendige Werkzeuge wie z.B. In-Circuit-Emulatoren. Diese haben den Nachteil, daß sie sehr teuer sind und meist nicht die vollen Möglichkeiten des Prozessors emulieren können. Besonders die Emulation eines Timers, der auf einem Prozessor realisiert worden ist, ist sehr aufwen-

dig. Daher ist der Einsatz von Emulatoren im Realzeitbereich nicht unkritisch.

Durch die Verwendung eines Monitors, der auf dem Zielprozessor läuft, wird ein Minimalwerkzeug zur Verfügung gestellt, mit dem die Entwicklung von Programmen für den Prozessor ohne Emulator möglich ist. Hierbei hat der Benutzer des Monitors den Vorteil, daß er direkt auf der realen Maschine arbeitet, so daß ihm die gesamten Fähigkeiten des Prozessors zur Verfügung stehen. Gleichzeitig aber ist der Monitor so komfortabel, daß die Entwicklung von Programmen für den Prozessor leicht möglich ist. Außerdem stehen dem Anwender des Monitors die schon implementierten Routinen des Monitors zur weiteren Anwendung zur Verfügung.

Die Spezifikation des Befehlssatzes

Im folgenden wird der Befehlsumfang des Monitors WERMON angegeben, über den der Monitor verfügt. Der erste Buchstabe kennzeichnet gleichzeitig die Bezeichnung, über die der Befehl ausgeführt werden kann.

- L(oad): Laden von INTEL-Hex-Records ab einer angegebenen Adresse ins RAM
- G(o): Lauf eines Programms ab einer angegebenen Adresse
- R(egister): Anzeigen der definierten Prozessor-Register,PC, PSW und SP
- M(emory): Anzeigen und ändern von Speicherstellen beginnend ab einer Adresse
- D(ump): Anzeigen eines Speicherbereichs ab einer angegebenen Adresse
- B(reakpoints): Setzen, Zurücksetzen und Anzeigen von Breakpoints
- T(race): Tracen einer angegebener Schrittzahl und Anzeigen der Registerinhalte
- H(elp): Hilfe für alle Befehle

Zusätzliche Fähigkeiten des Monitors

Der Monitor liest alle Zeichen, die über die serielle Schnittstelle am Prozessor ankommen, Interrupt-gesteuert ein. Die Zeichen werden in einen Ringpuffer eingelesen und aus diesem für die weitere Verarbeitung ausgelesen. Der Ringpuffer verfügt über eine High-Water-Marke und eine Low-Water-Marke, bei deren Erreichen der Monitor ein XOFF, bzw. ein XON sendet. Die Kommunikation des Monitors über die serielle Schnittstelle arbeitet im XON/XOFF-Protokoll, d.h. ist XON/XOFF-synchronisiert. Die Ausgabe von Zeichen über die serielle Schnittstelle erfolgt im Polling-Mode, das bedeutet, daß nur in einem bestimmten Zustand des Monitors Zeichen in das Datenregister (SBUF(TX)) übertragen werden können. Auf das Erreichen dieses Zustands muß gewartet werden. Der Monitor bietet weiterhin die Möglichkeit, daß der Benutzer eigene Interrupt-Service-Routinen verwenden kann, d.h. die Interrupt-Vektoren können umdirigiert werden. Weiterhin stehen dem Benutzer bestimmte Basis-Routinen des Monitors zur Verfügung, die er ohne Einschränkungen nutzen kann.

Die Entwicklungswerkzeuge

Für die Erstellung des Monitors WERMON für den MCS 8096 und MCS 80C196 wurden folgende aufeinander abgestimmte Werkzeuge benutzt:

- PC XT mit Expansion Unit
- Editor (E3.11)
- ASM-96 ASSEMBLER
- LR-96 LINKER/RELOCATOR
- VLSiCE-96 Emulator

Die Implementierung des Monitors (WERMON)

Die Implementierung des Monitors ist in drei Schritten vorgenommen worden. Der erste Teil umfaßt die Implementierung der elementaren I/O-Funktionen. Der zweite Teil die Implementierung weiterer wiederkehrender Routinen, die für den dritten Teil, die Implementierung der Befehls-Routinen, benötigt werden. Auf Einzelheiten der Implementierung soll hier nicht eingegangen werden.

Die realzeitorientierte Rechnerstruktur "KIROS" für den MCS 80C196

Die Zielsetzung der Rechnerstruktur

Mit dem Monitor WERMON ist die Basis geschaffen worden, den MCS8096 sinnvoll nutzen zu können. Auf dieser Basis soll ein weiteres Programm arbeiten, welches dem Benutzer eine vorgegebene Art von Befehlen und Möglichkeiten zur Verfügung stellt. Dieser Befehlsumfang ist Hochsprachen-ähnlich, unterstützt hierbei aber alle Fähigkeiten des Prozessors. Für den Anwender steht dann nur noch dieser Befehlsumfang zur Verfügung, so daß er von der Notwendigkeit enthoben wird, sich mit den Feinheiten und Besonderheiten des MCS 8096 genauer zu beschäftigen. Auf diese Art wird dem Prozessor eine neue Struktur gegeben, die dem Anwender gegenüber als die eigentliche Rechnerstruktur erscheint.

Diese Rechnerstruktur genügt in erster Linie Realzeitbedingungen. Es ist möglich, das Eintreten und Auslösen eines bestimmten Ereignisses, von einem bestimmten Zeitpunkt aus gesehen, genau vorherzusagen oder vorherzubestimmen. Dies wird durch die Verwendung von verkettetem Code erreicht. Zusätzlich ist die Rechnerstruktur strukturiert. Es können kleinere Bausteine zu größeren zusammengesetzt werden. Außerdem ist der Befehlsumfang so beschaffen, daß er erweiterbar ist. Dies bedeutet, daß aus dem entstehenden Befehls-(Wort-)satz eine neue - gewünschte - Funktion erstellt werden kann, die dann den Sprachumfang erweitert und für weitere Anwendungen sofort zur Verfügung steht. Abschließend ermöglicht es die Rechnerstruktur, mit anderen Systemen erstellte Programme (oder Wörter) auf diesem System auszuführen. Das heißt bei gleicher Syntax ist der Programmcode wiederverwendbar oder portabel.

Aus diesen Gründen wird der Aufbau und der Befehlsvorrat des Systems KIROS (**K**ompiler und **I**nterpreter für eine **R**ealzeit-**O**rientierte **S**prache), in dem die Rechnerstruktur realisiert ist, FORTH 79-ähnlich

sein. Der Sprachumfang wird aber in Bezug auf den MCS 8096 (MCS 80C196) erweitert oder angeglichen sein. (So wurden die Massenspeicherverwaltung und der Editor, die zum Standard gehören, nicht realisiert.)

Die Grundzüge des Systems KIROS

Der Implementierung von KIROS liegen verschiedene Realisierungsprinzipien zugrunde. So sind z.B. alle Routinen miteinander verkettet. Diese Verkettung, wie die für die Verwaltung und Ausführung notwendigen Mechanismen bilden die Grundsäule des Systems. Hierauf basieren alle weiteren notwendigen Systemkomponenten. Diese Verkettung findet nach einem bestimmten Schema statt. Für die Verwaltung der verketteten Routinen werden bestimmte Mechanismen und Werkzeuge benötigt, die auf die Anwendung im Realzeitbereich abgestimmt sind.

KIROS besteht aus einer Menge von Routinen, die Wörter genannt werden und miteinander verkettet sind, welches das Wörterbuch bilden. Die einzelnen Wörter sind unterteilt in einen Wortkopf, welcher das Auffinden der einzelnen Wörter ermöglicht, der Wortverkettung, mit welcher die einzelnen Wörter verbunden werden und dem Wortrumpf. Im Wortrumpf der Wörter ist bei den elementaren Wörtern ausführbarer Maschinencode zu finden und im Wortrumpf der Hochsprachen-Wörter sind die Adressen von elementaren Wörtern zu finden, aus denen das jeweilige Hochsprachen-Wort aufgebaut worden ist.

Das Ausführen von Wörter im realen Prozessor wird von einer virtuellen Maschine (das sind zwei Assembler-Routinen) gesteuert, die das Kernstück von KIROS bilden.

Die Fähigkeiten von KIROS

Das System KIROS arbeitet in zwei verschiedenen Modi. Diese Modi kennzeichnen auch die Fähigkeiten des Systems. Im Interpretier-Modus werden alles Eingaben interpretativ bearbeitet und ausgeführt. Hier wird auf den vorhandenen Wortschatz von KIROS zugegriffen und mit den vorhandenen Möglichkeiten gearbeitet. So hat das System umfangreiche mathematische Möglichkeiten, Wörter für die Stackmanipulation, für adreßbezogene Möglichkeiten, für String-Operationen und Wörter für formatiertes Ausgeben. Seine besondere Stärke liegt aber in der Möglichkeit, neue Funktionen, bzw. Wörter, zu erstellen. Dies wird im Kompilierungs-Modus vorgenommen. Hierbei kann auf diverse Kontrollstrukturen zugegriffen werden.

Die Erweiterbarkeit von KIROS durch Erzeugen neuer Wörter

Die besondere Stärke von KIROS liegt darin, daß auf einfache Weise weitere Wörter gebildet werden können. Diese Wörter können sofort ausprobiert und im Bedarfsfall verbessert werden.

Bei einer gegebenen Problemstellung empfiehlt sich bei der Problemlösung in der Entwurfsphase TOP-DOWN vorzugehen und die Programmierung BOTTOM-UP durchzuführen. Im sogenannten Kompilierungs-Modus können Variablen, Konstanten und Arrays definiert werden. Weiterhin kann die Verzweigungs-Struktur `IF ... ELSE` und eine Reihe von Schleifen (z.B. `DO ... WHILE, BEGIN ... AGAIN`) verwendet werden.

Die TRACE-Möglichkeiten mit KIROS

Das System KIROS bietet die Möglichkeit, Wörter zu tracen. Dies ist eine Hilfe beim Austesten von neuen Wörtern, denn die Wirkung eines neuen Wortes auf den Stack ist nur als Ganzes nach dem vollständigen Abarbeiten des Wortes zu ersehen. Vielfach ist es aber hilfreich, die Wirkung auf den Stack der einzelnen Wörter, aus denen das neue Wort aufgebaut ist, zu beobachten. Diese Möglichkeit bezieht sich nur auf eine Ebene. Werden also Wörter in dem neuen Wort aufgerufen, die wiederum aus verschieden anderen bestehen, so werden diese nicht einzeln, sondern nur das Wort ausgeführt. Es besteht die Möglichkeit, das Tracen generell zuzulassen (`TRON`) oder generell zu untersagen (`TROFF`). Die Stelle, an der in einem Wort der Trace-Vorgang beginnen soll, wird durch das Wort `TR>` gekennzeichnet. Die Stelle, an der der Tracevorgang abgebrochen werden soll, wird durch das Wort `TR<` gekennzeichnet.

So ist ein Beispiel für die Anwendung des Trace folgendes Wort:

```
: ZEIGE TR> 1 2 3 4 SWAP
ROT DUP 2DROP 2DROP DROP
TR< ;
```

Bei den einzelnen Traceschritten, die jeweils durch das Drücken einer beliebigen Taste ausgelöst werden, wird mit Hilfe des (unsichtbaren, also nicht über einen Namen aufrufbaren) Wortes `TRW` der Stackinhalt über das Wort `.ST` (unformatiert) und der Name des nächsten auszuführenden Wortes über das Wort `.ID` angezeigt.

Dies sieht für das Wort `ZEIGE` folgendermaßen aus:

```
STACK =  WORT= 1
STACK = 1  WORT= 2
STACK = 2 1  WORT= 3
STACK = 3 2 1  WORT= LIT
STACK = 4 3 2 1  WORT= SWAP
STACK = 3 4 2 1  WORT= ROT
STACK = 2 3 4 1  WORT= DUP
STACK = 2 2 3 4 1  WORT= 2DROP
STACK = 3 4 1  WORT= 2DROP
STACK = 1  WORT= DROP
STACK =  WORT= TR<
ok
```

Die automatische Stacküberprüfung

Während der Entwicklung neuer Funktionen ist es wünschenswert, die Stackgrenzen zu überwachen, da ein Überschreiten der Stackober- oder Stackuntergrenze zum Systemzusammenbruch führt. Daher ist im System eine automatische Überprüfung vorhanden, welche testet, ob der Stack über seine Grenzen hinausgewachsen ist. Das Anschalten der automatischen Stacküberwachung wird bei der Initialisierung des Systems vorge-

Der Autor

Dipl.-Inform. Marcus Werner, geb. 65, studierte 1984 - 1989 in Hagen (FernUniversität). Sein 1985 begonnenes Studium der Rechtswissenschaften hat er nach 2 jähriger Unterbrechung in Tübigen wieder aufgenommen und arbeitet außerdem als Wissenschaftlicher Mitarbeiter am Lehrstuhl Prof. Dr. F. Haft im Bereich der Juristischen Expertensysteme.

```
COMMA:  DOCOL                   -- n                    ; Beginn des Wortes HERE
        HERE                    -- n Hadr               ; Adresse der erste fr. Stelle
        DUP                     -- n Hadr Hadr          ; verdop., damit es nicht verl. geht
        TMOD                    -- n Hadr f             ; ist die Adresse gerade ?
        ZBRAN COMMA1            -- n Hadr               ; wenn gerade, dann springe
        ONEP                    -- n (Hadr+1)           ; wenn ungerade Adresse + 1
        STORE                   -- *                    ; kompiliere
        THREE                   -- 0003                 ; erhöhe
        ALLOT                   -- *                    ; den Wörterbuchzeiger um drei
        SEMIS                   -- *                    ; Ende des Wortes
        ;
COMMA1: STORE                   -- *                    ; kompiliere
        TWO                     -- 0002                 ; erhöhe
        ALLOT                   -- *                    ; den Wörterbuchzeiger um zwei
        SEMIS                   -- *                    ; Ende des Wortes
```

Bild 1: Realisierung des Wortes , (Komma)

nommen, ist aber auch explizit über das Wort STCON möglich. Da die Stacküberprüfung die Funktionsausführungszeiten verlängert, kann sie mit dem Wort STCOFF abgeschaltet werden. Dies ist auch der Normalzustand beim Ausführen von ausgetesteten Funktionen. Soweit die automatische Stacküberprüfung gewünscht wird, wird geprüft, ob der Stack über seinen festgelegten Bereiche gelaufen ist und es wird eine Reinitialisierung des Systems durchgeführt.

Die Implementierung

Der realisierten Implementierung liegt die Programmiersprache FORTH zugrunde. Ein besonderes Problem in der gesamten Implementierungsphase war die strenge Wortorientierung des INTEL-Prozessors. So ist es im MCS8096 (und auch beim MCS 80C196) nicht möglich, ein Speicherwort über eine ungerade Adresse anzusprechen. Dieses Problem trat bei KIROS in erster Linie beim Zugreifen auf Adressen über Zeiger und beim Kompilieren von Adressen hinter Strings ungerader Länge auf. Das erste Problem wurde durch sorgfältiges Definieren von Variablen und Zeigern gelöst. Die Lösung des zweiten Problems wurde durch das Modifizieren von Wörtern erreicht, die am Kompilierungsvorgang beteiligt sind. Es sind dies die beiden Wörter: ." und , (Komma). Die Implementierung dieser beiden Wörter wird im folgenden detailliert vorgestellt.

Mit dem Wort , wird die einfachgenaue Zahl, die vom Stack entfernt wurde, in die nächste freie Wörterbuch-Stelle abgespeichert und der Wörterbuch-Zeiger wird um 2 bzw. 3 erhöht. Das Wort , wird in erster Linie verwendet, um Adressen von anderen Wörtern in das Wörterbuch zum Aufbau von neuen Wörtern hineinzukompilieren. Die Besonderheit an diesem Wort liegt in der Prüfung, ob die Adresse, in welche die nächste Wortadresse (in Form des obersten Stackelementes vorhanden) abgespeichert werden soll, ungerade oder gerade ist. Diese Prüfung ist besonders dann wichtig, wenn die Linkadresse angelegt werden soll, das letzte Byte des Namens des neuen Wortes aber auf einer geraden Adresse liegt. Es muß hier ein künstliches Loch im Code erzeugt werden. Außerdem ist diese Prüfung von Bedeutung, wenn ein String in einem Wort angelegt wird, und das letzte Byte des Strings liegt wiederum auf einer geraden Adresse. Damit die nächste Adresse auch hier wieder auf einer geraden Adresse liegt, muß auch hier ein Byte frei bleiben. Die Realisierung des Wortes , (Komma) sieht man in Bild 1.

Das Wort (.") ist die mit dem Wort ." in das Wörterbuch hineinkompilierte Laufzeitausführende. Ein dem Wort ." im Wörterbuch folgender Text wird demnach mit Hilfe des Wortes (.") ausgegeben. Die Realisierung des Wortes (.") findet man in Bild 2.

Das Wort beginnt - wie alle Hochsprachen-Wörter - mit DOCOL. Dann wird der IP über das Wort R vom Rückkehr-Stack auf den Stack gebracht. Dieser Wert wird um zwei erhöht, wodurch die Anfangsadresse des Strings erreicht wird. Mit dem Wort COUNT wird festgestellt, wie lang der String ist (n1-Bytes) und wo genau er beginnt (adr2). Nun wird

```
PDOTQ:  DOCOL                   --  *                                   ;### n
        Rx                      --  n                                   ;### n
        TWOP                    --  (n+2)                               ;### n
        COUNT                   --  adr2 n1                             ;### n
        DUP                     --  adr2 n1 n1                          ;### n
        ONEP                    --  adr2 n1 (n1+1)                      ;### n
        FROMR                   --  adr2 n1 (n1+1) n                    ;### *
        PLUS                    --  adr2 n1 ((n1+1)+n)                  ;### *
        DUP                     --  adr2 n1 ((n1+1)+n) ((n1+1)+n)       ;### *
        TMOD                    --  adr2 n1 ((n1+1)+n) f                ;### *
        ZBRAN PDOTQ1            --  adr2 n1 ((n1+1)+n)                  ;### *
        ONEP                    --  adr2 n1 ((n1+1)+n)                  ;### *
PDOTQ1: TOR                     --  adr2 n1                             ;### ((n1+1)+n)
        TYPE                    --  *                                   ;### ((n1+1)+n)
        SEMIS                   --  *                                   ;### ((n1+1)+n)
```

Bild 2: Realisierung des Wortes (.")

Worte für ...	Aufzählung der Namen
die Stackmainpulation	2DROP 2DUP >R ?DUP DROP DUP OVER PICK R R> ROLL RP! SP! SWAP
arithmetische Operationen	+ - * 1+ 1- 2+ 2- 2* 2/ 2MOD D+ D- U* U/
logische Verknüpfungen	AND DNEGATE NEGATE NOT OR XOR
Vergleichs-Operationen	< > 0 0< 0=
speicherbezogene Operationen	@ SP6)
den Aufbau von Kontrollstrukturen	0BRANCH (+LOOP) (DO) (LOOP) BRANCH I J LEAVE
Eingabe- und Ausgabe-Operationen	CR ?KEY KEY EMIT ENCLOSE
die numerische Repräsentation	DIGIT
Definitionsworte	DOES> ARRAY CONSTANT CONVAR USER VARIABLE
die Verwaltung des Wörterbuchs	CFA NFA PFA
das Vokabularsystem	FORTH LFA
System-Wörter und sonstige Wörter	;S : (FIND) COLD EXECUTE LIT WERMON
die Trace-Ausführung und die Stacküberprüfung	TR> TR< TROFF TRON STCOFF STCON
Konstanten, Variablen und USER-Variablen	0 1 2 3 AD_RESULT AD_RESULT_LO AD_RESULT_HI BASE BL BLK COLUM CONTEXT CSP CURRENT DP DPL EXFLAG EXSTAT FENCE HLD HSI_MODE HSI_TIME HSO_COMMAND HSI_STATUS IN IOC0 IOC1 IOS0 IOS1 INT_MASK IP_LIMIT MSGCNT MSGPOOL NEXT OUT PWM_CONTROL R0 S0 SBUF SP SP_CON SP_STAT STARTPTR STATE TIB TXBLK VAR1 VAR2 VOC-LINK VP WARNING WATCHDOG_TIMER WIDTH WP

Tabelle 1: Liste der elementaren Wörter für KIROS

über die Abfolge DUP1+ R< + DUP DUP TMOD festgestellt, ob die End-Adresse des Strings ungerade oder gerade ist. Ist die Adresse gerade, so wird Eins hinzuaddiert (mit Hilfe des Wortes 1+), da über das Wort , hier ein freies Byte erzeugt wurde. Diese errechnete Adresse ist dann der neue IP, an der die Adresse, des nächsten auszuführenden Wortes steht. Ist die Adresse ungerade, so erfolgt keine weitere Aktion und der String wird in beiden Fällen über das Wort TYPE ausgegeben. Mit SEMIS wird das Wort beendet.

Der KIROS-Wortschatz

Im System KIROS wurden die in Tabelle 1 aufgeführten elementaren Wörter implementiert und gehören zum Wortschatz. Die Hochsprachen-Wörter findet man in Tabelle 2.

HINWEIS

Eine detaillierte Beschreibung der Elementaren- und der Hochsprachen-Wörter werden vom Autor auf Wunsch - gegen Kostenerstattung - gerne zur Verfügung gestellt.

Worte für ...	Aufzählung der Namen
die Stackmainpulation	.S .ST ?ST DEPTH ROT
arithmetische Operationen	*/ / /MOD */MOD M/MOD MOD ABS DABS S->D MAX MIN
speicherbezogeneO perationen	+ORIGIN ? BLANKS ERASE FILL FILLALL IOPORT0@ IOPORT1! IOPORT1@ IOPORT3! IOPORT3@ IOPORT4! IOPORT4@ MFREE TOGGLE
Eingabe- und Ausgabe-Operationen	(NUMBER) . D.R D. EXPECT FINIS PAD QUERY SOURCE TYPE WORD
die Formatierung und die Eingabe-/Augabe-Formatierung	<# # #> #S ." .R -TRAILING COUNT HOLD SPACE SPACES
die Verwaltung des Wörterbuchs	' , -FIND ALLOT BACK C
das Vokabularsystem	DEFINITIONS ID. LATEST SMUDGE VOCABULARY
die Sicherheit des Compilerbetriebs	!CSP ?COMP ?CSP ?ERROR ?EXEC ?PAIRS ?STACK ABORT ERROR MESSAGE
die numerische Repräsentation	NUMBER SIGN
die Verwaltung eines Massenspeichers	FORGET
System-Wörter und sonstige Wörter	((ABORT) CLRSCR INTERPRET 0 QUIT STARTUP STATUS VERSION
die Zahlensysteme	DECIMAL HEX
Wörter, die nur in einer COLON Definition benutzt werden dürfen	; <BUILDS [ ] ;CODE (;CODE) (.") [COMPILE] AGAIN BEGIN COMPILE DLITERAL DO ELSE END ENDIF IF LITERAL LOOP REPEAT THEN UNTIL WHILE
Unsichtbare Wörter	STC TRW

Tabelle 2: Liste der Hochsprachen-Wörter für KIROS

CrossCompiler für FORTH

Dipl.-Ing. Ralf Neuthe

An der WPU in Rostock wird seit einem halben Jahr erfolgreich ein eigenständig entwickelter FORTH-CrossCompiler für die Implementierung von interaktiven Grundsystemen und von zugeschnittenen Anwendersystemen eingesetzt. Damit erhöhte sich in weitem Maße deren Wartbarkeit und Portabilität.

In diesem Beitrag sollen die wesentlichen Eigenschaften des comFORTH-CrossCompilers näher beschrieben und erläutert werden.

Als erstes Kriterium für den Anwender des CrossCompilers dürfte eine möglichst strenge Einhaltung der FORTH-Syntax gelten. Das ermöglicht nämlich erst eine einfache Übertragung der Programme vom komfortablen Testsystem auf den Applikationsrechner. Es kommt auf eine möglichst originaltreue Simulation einer interaktiven FORTH-Umgebung an. Da ein Kompilat aber noch nicht ausführbar ist, treten bei diesen Bestrebungen immer dann erhebliche Schwierigkeiten auf, wenn neue Wortklassen oder Compilererweiterungen gewünscht sind. Es müssen an diesen Stellen gut durchdachte Konzepte Abhilfe schaffen, die Eigenschaften wie

- weitgehende Entkopplung zwischen Compiler und Zielsystem
- einfache Erweiterbarkeit des Compilers
- Sicherung der FORTH-Syntax

unterstützen.

Bei bisher bekannten FORTH-Compilern (wie z.B. in /1/ beschrieben) war eine Compilererweiterung durch den Anwender nur möglich, wenn dieser auch spezielles Wissen über die Programmierung des Compilers besaß. Bei der Erzeugung von Zielprogrammen waren einige Tricks und Kniffe unumgänglich, deren Anwendung nur sicher beherrschbar waren, wenn man gut über den Mechanismus des Compilers informiert war. Compiler und Kompilat waren unzureichend voneinander entkoppelt.

Im comFORTH-CrossCompiler wurden diese Mängel durch neue Konzepte weitestgehend beseitigt. Dazu gehört die Verwendung von unterschiedlichen Betriebsarten zur Differenzierung der Zustände, die bei der Interpretation von FORTH-Quellcode auftreten. Dabei existiert auch eine spezielle Betriebsart – der Macro-Mode – der die Definition von Compiler-Worten und neuen Wortklassen auf eine transparente Art und Weise ermöglicht. Die Umschaltung geschieht automatisch durch entsprechende Schlüsselwörter. Bild 1 vermittelt einen Überblick über die wesentlichen Betriebsarten.

Der Control-Mode dient nur organisatorischen Zwecken. In ihm werden unter anderem die einzelnen Speichersegmente mit ihren Adressen festgelegt.

Der zentrale Punkt ist der Cross-Mode, von dem aus die Definition eines Secondaries (→ Übergang in den Secondary-Mode) bzw. die Definition von Primaries (→ Übergang in den Primary-Mode) oder die Erzeugung von Variablen und Konstanten (Verbleib im Cross-Mode) gestartet werden kann.

Die Analogie zwischen dem Secondary-Mode des CrossCompilers und dem Compile-Mode gewöhnlicher FORTH-Systeme ist leicht erkennbar.

Der Primary-Mode dagegen besitzt kein direktes Äquivalent. Es ist zur Vokabularumschaltung auf `ASSEMBLER` analog.

Mit den Betriebsarten wird im wesentlichen der verfügbare Befehlssatz umgeschaltet. Auf diese Art läßt sich die Unmöglichkeit der Ausführung von Definitionen des Kompilates sehr gut verdecken. An der Benutzung der Worte `HEX` und `DECIMAL` ist der Sachverhalt leicht ersichtlich. Im Secondary-Mode müssen die entsprechenden Codefeldadressen in das Wörterbuch des Kompilates eingetragen werden. Im Cross-Mode dagegen sollen sie die Zahlenbasis des Compi-

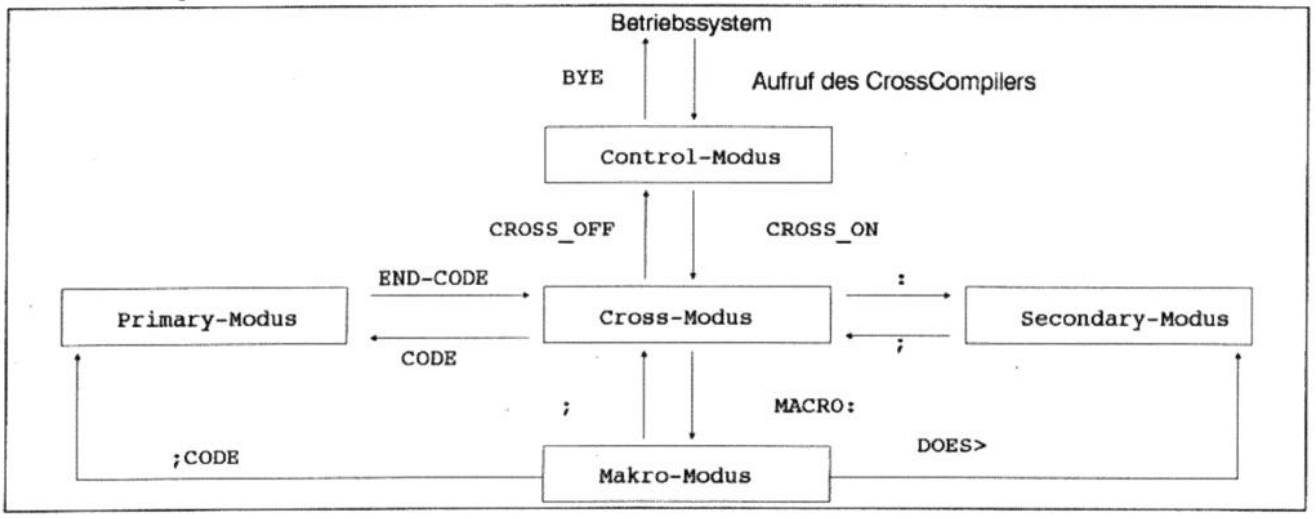


Bild 1

F83

Lokale Variablen und Argumente

Übersetzung von Gert-Ulrich Vack
aus Forth Dimensions Vol.XI,Nummer 1

von Jyrki Yli-Nokari – Tampere, Finnland

Ich beschreibe ein Verfahren, mit dem FORTH um lokale Variablen und Argumente (innerhalb von Prozeduren) ergänzt wird. Der Code ist in F83 von Laxen & Perry geschrieben. Er müßte aber ganz gut auch auf beliebige übliche FORTH-Systeme übertragbar sein. Ich möchte nicht darüber streiten, ob von nun an jedermann lokale Variablen und Argumente verwenden sollte. Falls man bisher gezwungen war, `PICK` zu verwenden, so wird dieser Beitrag doch wenigstens von akademischen Interesse sein.

Entwurfsziele

Mein Hauptentwurfsziel bestand darin, über lokale Variablen und Argumente zu verfügen, die "sauber" waren und zu FORTH passen. D.h., sie sollten portierbar sein, ROMbar, rekursiv und möglichst nicht zustandsabhängig. Sie sollten auch einfach und anschaulich handhabbar sein. Eine einzige Vereinbarung am Beginn der Definition sollte ausreichen. Ein wichtiges Kriterium bestand darin, daß der Quelltext insgesamt nur ein oder zwei Screens umfassen sollte. Anderenfalls würde sich niemand damit herumplagen, ihn auszuprobieren. Lokale Variablen sollten ähnlich wie lokale Argumente funktionieren. Der Hauptunterschied sollte darin bestehen, daß sich Argumente wie Konstanten verhalten und lokale Variablen wie Variablen. Diese Analogie bedingt entsprechende Definitionswörter.

Spezifikationen

Eine lokale Variable wird zur Ausführungszeit gebildet, indem Platz auf dem Returnstack gebunden wird. Die Lebensdauer erstreckt sich vom Beginn bis zum Ende der Definition, in der sie vereinbart wird. Am Ende der Definition wird der gebundene Speicherplatz wieder freigegeben. Weil der Speicherplatz auf dem Stack gebunden wird, ist Rekursion möglich und die ROMbarkeit des Codes wird nicht beeinflußt. Auf lokale Variablen wird wie auf andere Variablen über ihre Adresse zugegriffen.

Lokale Variablen werden mit Returnstackframes implementiert.

Ein Argument ist eines der Stackeingabedaten, die ein Wort erhält. Beim Deklarieren von Argumenten werden diese dem Stack entnommen und auf dem Returnstack abgelegt (lokale Variablen verhalten sich ähnlich, sind aber nicht initialisiert).

Auf Argumente wird über ihren Wert zugegriffen, wie auf Konstanten. In der vorliegenden Implementierung sind diese genauso wie Variablen auch über die Adressen erreichbar, wenngleich das unnötig zu sein scheint, weil Argumente nur Stackeingabeparameter sind.

Programmiermodell

```
ARGS ( arg1 ... argN N – )
```

deklariert N Stackeinträge als Argumente. Die Stackeinträge sind arg1 ... argN. Sie werden auf dem Returnstack abgelegt. Am Ende der Ausführung des Wortes, das `ARGS` aufgerufen hat, wird der Speicherplatz automatisch freigegeben.

Das Hauptwort zum Übertragen des Wertes von Argument N auf dem Stack ist:

```
% ( N – argN )
```

Die Numerierung beginnt beim tiefsten Stackeintrag (in einem Stackdiagramm ist das der am weitesten links stehende Eintrag). Der Name `%` wurde ausgewählt, weil er in FORTH bisher keine Bedeutung hat und zugleich mit der Bezeichnung von Parametern in MSDOS-Batchdateien übereinstimmt.

Für schnelle Zugriffe habe ich die Wörter `%1`, `%2`, ... `%8` definiert. Dazu wurde ein Definitionswort verwendet:

```
ARGUMENT  <name> ( N – )
          <name> ( – argN )
```

`ARGUMENT` erzeugt ein neues Wort `<name>`, welches den Inahlt des Argumentes N auf den Stack überträgt.

So weit die Theorie. Sehen wir uns das Ganze in der Praxis an.

```
: SWAP
  2 ARGS
  ( n1 n2 – n2 n1 )
  %2 %1 ;
```

Stichworte

- lokale Variablen
- lokale Argumente
- Returnstackframes

```
: CONVENIENT
  4 ARGS ( x a b c — a*x*x + b*x + c )
  %2 %1 * %1 * %3 %1 * + %4 + ;

: THEORETIC
  4 ARGS ( x a b c — a*x*x + b*x + c )
  2 % 1 % * 1 % * 3 % 1 % * + % 4 + ;

1 ARGUMENT x
2 ARGUMENT a
3 ARGUMENT b
4 ARGUMENT c
: ACADEMIC
  4 ARGS ( x a b c — a*x*x + b*x + c )
  a x * x * b x * + c + ;
```

Bild 1: Drei Varianten einer Definition

```
: ROT
  3 ARGS
  ( n1 n2 n3 — n2 n3 n1 )
  %2 %3 %1 ;

: NIP
  2 ARGS
  ( n1 n2 — n2 )
  %2 ;

: TUCK
  2 ARGS
  ( n1 n2 — n2 n1 n2 )
  %2 %1 %2 ;
```

Bild 1 zeigt zum Vergleich eine Definition, die in drei verschiedenen Varianten geschrieben wurde. ARGS braucht man nicht am Anfang einer Definition zu schreiben. Es kann auch mitten drin gerufen werden. Die von ARGS bereitgestellten Argumente sind bis zum Definitionsende verwendbar. Es ist allerdings nicht ratsam, ARGS in DO-Schleifen zu verwenden; dies ist in der Tat nicht möglich.

Gewöhnlich wird auf Argumente wie auf Konstanten zugegriffen. Es ist aber auch möglich, sie wie Variablen zu verwenden. Das geschieht mit dem Wort:

```
ARGV ( N — 'argN )
```

Dieses Wort gibt die Adresse des Arguments N zurück. (Der Name wurde C entlehnt).

Lokale Variablen werden mit folgendem Wort vereinbart:

```
LOCALS  ( N — )
```

Dieses Wort bindet Speicherplatz für N Variablen. Das ist identisch mit ARGS, allerdings sind Variablen nicht initialisiert (und enthalten garantiert von Null verschiedenen Unsinn).

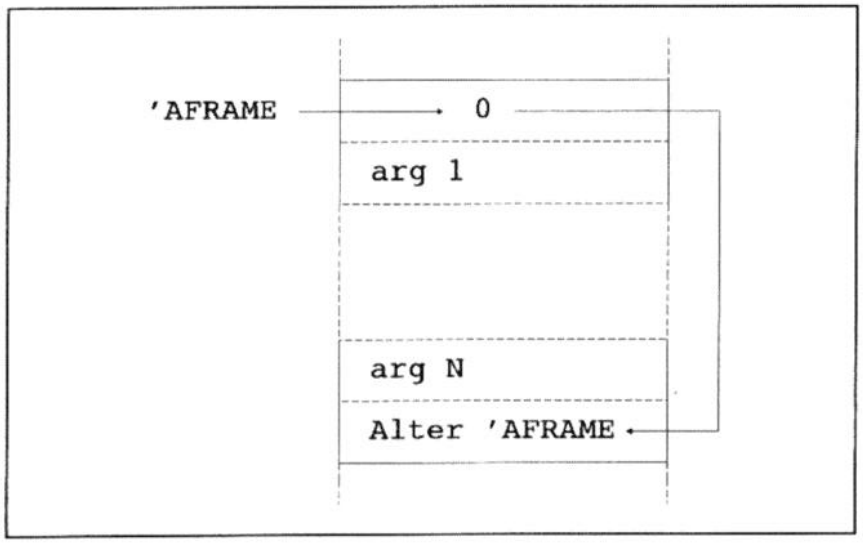


Bild 2: Stackframe für zwei Argumente

Das wichtigste Wort zum Zugriff auf die lokale Variable N ist:

```
LV  ( N — A )
```

Dieses Wort legt die Adresse der lokalen Variablen mit der Nummer N auf dem Stack ab. (Wenn man N LOCALS definiert, hat man von 1 bis N durchnumerierte lokale Variablen, wie mit ARGS).

Für schnellen Zugriff habe ich die Wörter L1 ... L8 vordefiniert. Es gibt auch die Wörter @1 ... @8 für das Lesen des Inhalts und !1 ... !8 für das Abspeichern von Werten in lokalen Variablen. Die Syntax ist an [Bow82] angelehnt. Diese Wörter werden mit speziellen Definitionswörtern gebildet, die nur dazu dienen, lokale Variablen mit Namen zu versehen (Bild 2). Dieses Vorgehen wird jedoch nicht empfohlen - so lange man keine transienten, voneinander unabhängig vergeßbaren Vokabulare hat - weil die auf diese Weise definierten Variablennamen genausogut für andere Definitionen erreichbar bleiben.

In den angegebenen Screens gibt es drei verschiedene Implementierungen des Puzzles "Türme von Hanoi". Die erste Implementierung geht allein von der Nutzung von LOCALS und ARGS aus. Die zweite Version ist eine Mischung aus ARGS und üblichem Code. Die dritte Version ist eine ganz gewöhnliche FORTH-Implementierung. RECURSE ruft die Definition rekursiv auf. Dieses Wort gibt es in FORTH-83 im kontrollierten Referenzwortschatz. Bei einigen älteren FORTH-Systemen heißt dieses Wort MYSELF. Ich bevorzuge das F83-Wort RECURSIVE und verwende den Wortnamen selbst anstelle von RECURSIVE. (Ich hoffe, der neue Standard wird dieses Wort im "controlled reference"-Wortschatz berücksichtigen).

Implementierung

Lokale Variablen und Argumente werden unter Nutzung von Frames auf dem Returnstack implementiert. Das bedeutet, daß der Returnstack im Bereich des wahlfrei adressierbaren Speichers liegen muß (das gilt für die meisten Mikrorechner-Implementierungen). Zusätzlich wird ein Verfahren benötigt, um den Return-

stackpointer lesen und wieder einspeichern zu können. In F83 ist das mit folgenden Definitionen gelöst:

```
RP@ ( – A )

RP! ( A – )
```

Diese Wörter lesen den Inhalt des Returnstackpointers und legen ihn auf dem Parameterstack ab bzw. entnehmen dem Parameterstack das oberste Element und tragen es entsprechend in den Returnstackpointer ein. (Man beachte, daß sich fig-FORTH's `RP!` anders verhält; dort entnimmt `RP!` den Kaltstart-Initialisierungsparameter aus dem Konstantenbereich der Boot-up-Parameter; *A.d.Ü.*)

Diese Implementierung setzt voraus, daß der Returnstack nach unten wächst (in Richtung fallender Adressen), daß der Stackpointer auf den obersten Eintrag zeigt und daß jeder Stackeintrag 16 Bit belegt. Bei der Mehrzahl der Implementierungen trifft das zu. Bei 32-Bit-Systemen muß jedes `2*` durch `4*` oder durch `CELL*` ersetzt werden oder was auch immer das System "versteht".

Sowohl `LOCALS` als auch `ARGS` haben eigene Stackframes (Bilder 2 und 3). Die Variablen `'LFRAME` und `'AFRAME` zeigen auf den Beginn des aktuellen Frames. Datenzugriffe werden über diese Zeigervariablen vermittelt. Beim Aufbau eines Stackframes wird zuerst der vorherige Wert des Framepointers auf dem Stack eingetragen, dann kommt der Datenbereich und schließlich der vorherige Returnstackpointer.

Sowohl `LOCALS` als auch `ARGS` nutzen eine Coroutinentechnik, um den Returnstack zu bereinigen. Zu Beginn entnehmen sie ihre Rücksprungadresse dem Returnstack. Nachdem der Stackframe aufgebaut ist, wird diese Adresse aufgerufen. Damit wird der Rest der Definition ausgeführt (d.h., die Stelle, wo `ARGS` oder `LOCALS` aufgerufen wurde). Danach wird die Steuerung an `ARGS` oder `LOCALS` an die Stelle nach dem `CALL` (Bild 4) zurückgegeben. Anschließend wird der Stackframe entfernt und die Steuerung wird zum Wort zwei Ebenen darüber zurückgegeben. Bild 4 zeigt die Ausführung und die Aufrufreihenfolge der Wörter `FOO` und `BAR`.

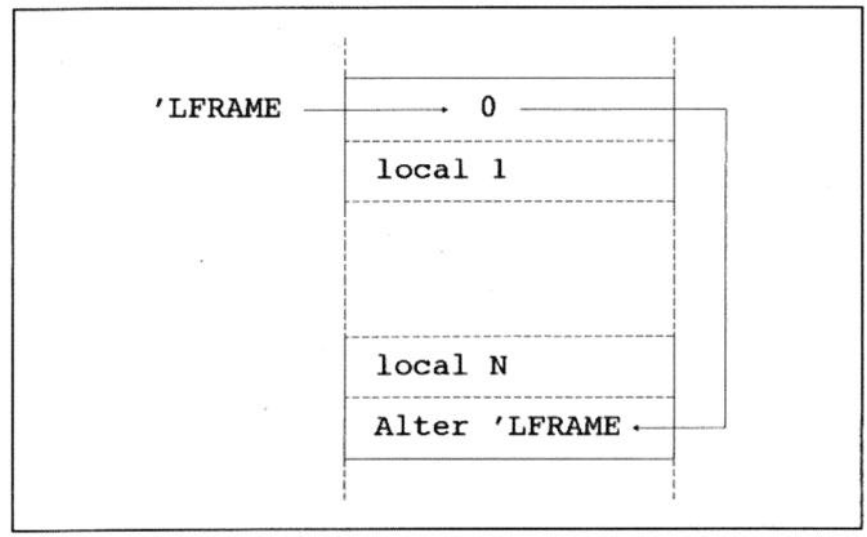


Bild 3: Stackframe für lokale Variablen

Bemerkungen

Den Entwurfszielen wurde recht gut nahe gekommen. Allerdings paßt die Implementierung nicht in zwei Screens. Auch gibt es keinen ganz einwandfreien Weg für `LOCALS` oder `ARGS` mit doppelter Genauigkeit, diese müssen in zwei Teilen behandelt werden.

Um einen praktischen Nutzen zu haben, sollte die Implementierung von `ARGS` und `LOCALS` etwa so aussehen:

```
: ARGS
   (APUSH)  CALL  (APOP)  ;

: LOCALS
   (LPUSH)  CALL  (LPOP) ;
```

wobei `(APUSH)` , `(APOP)` , `(LPUSH)` und `(LPOP)` Codedefinitionen sein sollten, ebenso wie die Wörter `LVARIABLE` , `@LVARIABLE` und `!LVARIABLE`.

`ARGUMENT` sollte so definiert werden:

```
: ARGUMENT
   CREATE 2*  C,  ;CODE
... ;
```

Interessant ist die Tatsache, daß der `;CODE` - Teil von `LVARIABLE` nahezu identisch sein sollte zum Code von Nutzervariablen, wobei jedoch `UP` durch `'LFRAME` ersetzt wird.

Beim Vergleich der Ausführungsgeschwindigkeiten der zweiten und der dritten Variante der "Türme von Hanoi", war die zweite etwa 20% schneller. Wenn `ARGS` wie oben angegeben geschrieben wird, müßte die zweite Variante schneller sein als die dritte. Leider konnte ich das nicht austesten - wer kann sich dazu äußern?

Das Numerierungsschema beginnt bei Eins anstatt bei Null. Diese Konvention wurde getroffen (konträre Auffassung hierzu ist beabsichtigt!),

```
: FOO BAR . ; ( n1 n2 – )

: BAR 2 ARGS %1 %2 + ; ( n1 n2 – n3 )

FOO ──→ BAR ──→ 2
                ARGS ──→ R>
                         (build frame)
                %1 ←──── CALL
                %2
                +
                EXIT ──→ (remove frame)
        . ←──────────────── EXIT
←──────── EXIT
```

Bild 4: Die Ausführungsreihenfolge von ARGS

Lokale Variablen und Argumente

```
    1                                                    8
0 \ Local variables and arguments load block      jty 220688 Local variables and arguments load block          jty 220688
1 2 3 THRU                                                   The source code is in blocks 2 and 3
2
3 VOCABULARY NOVICE  VOCABULARY MIXED                        There are three versions of the Tower of Hanoi puzzle,
4 NOVICE DEFINITIONS  4 LOAD                                 NOVICE that uses only ARGS and LOCALS,
5 MIXED DEFINITIONS  5 LOAD                                  MIXED that uses a mixture of stack manipulation and ARGS and
6 FORTH DEFINITIONS  6 LOAD                                  normal forth version that does not use anything special.
7
8
9
10
11
12
13
14
15

    2                                                    9
0 \ Local variables and procedural arguments, defining  JTY 190688 Local variables and arguments are return stack frames JTY 190688
1 VARIABLE 'LFRAME                                           'LFRAME contains pointer to the current local variable frame.
2 VARIABLE 'AFRAME                                           'AFRAME contains pointer to the current argument frame.
3 : CALL  ( A -- )  >R ;                                     CALL is much like EXECUTE, but takes IP-value instead of CFA.
4                                                            LOCALS allocates space for N local vars from the return stack
5 : LOCALS  ( N -- )  R>   'LFRAME @ >R   RP@   ROT 2* -   RP@    and sets 'LFRAME to point to the beginning of this frame.
6   SWAP RP!   >R   RP@ 'LFRAME !   CALL  R> RP!   R> 'LFRAME ! ;  Previous 'LFRAME is saved to enable recursion. After this it
7                                                            issues a coroutine call to the caller. When that returns,
8 : ARGS  ( k1..kN N -- )  R>  'AFRAME @ >R                  the return stack is reset to the original value and 'LFRAME is
9   RP@ 'AFRAME !   SWAP                                BEGIN  restored to previous value.
10    ROT >R                                  1- ?DUP 0=  UNTIL ARGS behaves like LOCALS, except that it moves N items to return
11   'AFRAME @ >R   RP@ 'AFRAME !   CALL  R> RP!   R> 'AFRAME ! ;  stack and set pointer to frame to 'AFRAME .
12                                                           LV returns the address of local variable number N.
13 : LV   ( N -- 'Ln )  2*  'LFRAME @ + ;                    ARGV returns the address of argument number N. The topmost
14 : ARGV  ( N -- 'An )  2*  'AFRAME @ + ;                    argument is number N (like in a stack diagram).
15 : %  ( N -- An )  ARGV @ ;                                % returns the value of argument number N. See ARGV .

    3                                                    10
0 \ Local variables and procedural arguments, and user  JTY 190688 Local variables & arguments defining & end user words JTY 190688
1 : LVARIABLE  ( N -- )  CREATE  C,  DOES>  ( -- 'n )  C@ LV ;   LVARIABLE creates a word that brings the address of Nth
2 : @LVARIABLE  ( N -- )  CREATE  C,  DOES>  ( -- Ln )  C@ LV @ ;  local variable to the stack at runtime.
3 : !LVARIABLE  ( N -- )  CREATE  C,  DOES>  ( n -- )  C@ LV ! ;  @LVARIABLE creates a word that brings the value of Nth
4 : ARGUMENT   ( N -- )  CREATE  C,  DOES>  ( -- An )  C@ % ;      variable to the stack at runtime.
5                                                            !LVARIABLE creates a word that stores the item in stack
6 1  LVARIABLE L1 2  LVARIABLE L2 3  LVARIABLE L3 4  LVARIABLE L4  to the Nth local variable at runtime.
7 5  LVARIABLE L5 6  LVARIABLE L6 7  LVARIABLE L7 8  LVARIABLE L8 ARGUMENT creates a word that brings the value of Nth argument
8 1 @LVARIABLE @1 2 @LVARIABLE @2 3 @LVARIABLE @3 4 @LVARIABLE @4  to the stack at runtime.
9 5 @LVARIABLE @5 6 @LVARIABLE @6 7 @LVARIABLE @7 8 @LVARIABLE @8
10 1 !LVARIABLE !1 2 !LVARIABLE !2 3 !LVARIABLE !3 4 !LVARIABLE !4 L1 .. L8 are local variables 1 .. 8
11 5 !LVARIABLE !5 6 !LVARIABLE !6 7 !LVARIABLE !7 8 !LVARIABLE !8 @1 .. @8 fetches the values of corresponding variables
12 1   ARGUMENT %1 2   ARGUMENT %2 3   ARGUMENT %3 4   ARGUMENT %4 !1 .. !8 stores the value on stack to the corresponding variable
13 5   ARGUMENT %5 6   ARGUMENT %6 7   ARGUMENT %7 8   ARGUMENT %8 %1 .. %8 bring the corresponding argument to the stack
14
15

    4                                                    11
0 \ Testing args and local variables: Tower of Hanoi #1 jty 180688 The Towers of Hanoi problem algorithm in ADAish       jty 220688
1 : THIRD  ( t1 t2 -- t3 )  2 ARGS   6 %2 -   %1 - ;        TYPE tower is (1, 2, 3);
2                                                            FUNCTION third(a, b: IN tower) RETURN tower IS BEGIN
3 : MOVEDISK  ( from to -- )  2 ARGS                            return 6-a-b;
4   CR ." Move disk from " %1 . ." to " %2 .   ( KEY DROP ) ; END third;
5                                                            PROCEDURE movedisk(a, b: tower); -- Move disk from a to b
6 : MOVETOWER  ( from to n -- )  3 ARGS 1 LOCALS   %3 1 =  IF PROCEDURE movetower(a, b: tower; n: Integer) IS t: tower; BEGIN
7     %1   %2   MOVEDISK                                ELSE     IF (n = 1) THEN -- Is there only one disk to move?
8     %1   %2   THIRD   !1                                        movedisk(a, b);
9     %1   @1   %3 1-   RECURSE                                 ELSE -- More than one
10    %1   %2   MOVEDISK                                        t := third(a, b);
11    @1   %2   %3 1-   RECURSE                         THEN ;      movetower(a, t, n-1);
12                                                                 movedisk(a, b);
13 : HANOI  ( N -- )  1 ARGS   1   3   %1   MOVETOWER ;          movetower(t, b, n-1);
14                                                               ENDIF
15                                                           END movetower;

    5                                                    6
0 \ Testing args and local variables: Tower of Hanoi #2 jty 180688 0 \ Testing args and local variables: Tower of Hanoi #3 jty220688
1 : THIRD  ( t1 t2 -- t3 )  6 SWAP -   SWAP - ;              1 : THIRD  ( t1 t2 -- t3 )  6 SWAP -   SWAP - ;
2                                                            2
3 : MOVEDISK  ( from to -- )   SWAP                          3 : MOVEDISK  ( from to -- )   SWAP
4   CR ." Move disk from " . ." to " .   ( KEY DROP ) ;      4   CR ." Move disk from " . ." to " .   ( KEY DROP ) ;
5                                                            5
6 : MOVETOWER  ( from to n -- )   1-                ?DUP IF  6 : MOVETOWER  ( from to n -- )   1-                ?DUP IF
7     3 ARGS   %1 %2 THIRD                                   7     >R   2DUP THIRD
8     %1 OVER %3 RECURSE   %1 %2 MOVEDISK   %2 %3 RECURSE  ELSE  8   2 PICK OVER R@ RECURSE
9     MOVEDISK                                         THEN ;  9   2 PICK 2 PICK MOVEDISK
10                                                           10    SWAP R> RECURSE DROP                          ELSE
11 : HANOI  ( N -- )  1 3 ROT MOVETOWER ;                    11    MOVEDISK                                      THEN ;
12                                                           12
13                                                           13 : HANOI  ( N -- )  1 3 ROT MOVETOWER ;
14                                                           14
15                                                           15
```

weil die Numerierung in Stackdiagrammen und im menschlichen Bewußtsein ebenfalls bei Eins beginnt.

Da eines der Entwurfsziele darin bestand, keine komplizierten, zustandsabhängigen Wörter zu haben, wurden keine Kompilationssicherungsfunktionen eingebunden. Das bedeutet, daß `LOCALS` und `ARGS` verwendet werden können, ohne daß tatsächlich lokale Variable oder Argumente deklariert werden. In diesem Fall wird der Frame der nächsthöheren Ebene adressiert, d.h. die `LOCALS` und `ARGS` des aufrufenden Wortes. Das wird als Eigenschaft gewertet, nicht als Fehler.

Schließlich scheint es nicht besonders nützlich zu sein, lokale Variablen bei jeder Gelegenheit zu verwenden, weil es bereits den Parameterstack für temporäre Werte gibt. Besonders die Wörter `L1 ... L8` und `@1 ... @8` scheinen nicht sehr nützlich zu sein. Wo es jedoch Bedarf an großen temporären Datenmengen gibt, werden lokale Variablen handlich.

Literatur

[Bow82] Bowhill, S.A.: Fast Local Variables for FORTH. FORML Proceedings 1982

[Bar82] Bartholdi, P.: Another Aid for Stack Manipulation and Parameter Passing in FORTH. 1982 Rochester FORTH Conference on Data Bases and Process Control Proceedings.

[Mor84] Morgenstern, L.: Anonymous Variables. FORTH Dimensions (VI/1)

[Ros87] Ross, P.: Local Variables. FORTH Dimensions (IX/4)

Der Autor

Jyrki Yli-Nokari schrieb 1985 einen UNIX-Guide und ist Konsultant in einem bedeutenden finnischen Softwarehaus, aber er bezeichnet FORTH als seine erste Liebe. 1981 schrieb er als Schulprojekt ein FORTH-79-System und Utilities für die PDP-11; seine Doktorarbeit war ein Multiuser, Multitasking FORTH-83 für den 6809.

F83

Lokale Variablen - ein anderes Verfahren

Übersetzung von Gert-Ulrich Vack
aus Forth Dimensions Vol.XI,Nummer 1

John R. Hayes – Laurel, Maryland

Stichworte

- lokale Variablen
- Gültigkeitsbereiche
- temporäres Wörterbuch

Hier wird eine weitere Methode für die Einführung namentlich ansprechbarer lokaler Variablen vorgestellt. Die Methode hat eine ästhetisch ansprechende Syntax und gestattet die Deklaration lokaler Variablen an beliebiger Stelle innerhalb einer Doppelpunktdefinition. Die Variable wird mit dem obersten Element des Parameterstacks zur Laufzeit initialisiert. Eine leistungsfähige Implementierung vervollständigt diesen Artikel.

Einführung

Die Effektivität und Transparenz von FORTH-Code leidet oft unter übermäßigem Gebrauch von Wörtern, die den Stack manipulieren. Diese Wörter - wie z.B. `DUP` , `SWAP` oder `ROT` - werden primär für das Positionieren von Operanden verwendet und sind als solche reiner Overhead. Noch wichtiger ist, daß Sequenzen wie `SWAP DROP ROT` das Wesen dessen verdunkeln, was der Code wirklich macht. Lokale Variablen würden in großem Maße helfen.

Verschiedene Verfahren zum Bilden lokaler Variablen sind in den vergangenen Jahren vorgeschlagen worden (siehe Literatur). Von der zweiten angegebenen Literaturstelle begeistert konnte ich eine weitere Implementierung von lokalen Variablen erfolgreich erstellen. Meine Methode gestattet die Deklaration lokaler Variablen an beliebiger Stelle innerhalb einer Doppelpunktdefinition. Diese Deklarationen können in gewöhnlichen FORTH-Code eingestreut werden. Der Speicherplatz für lokale Variablen wird dynamisch allokiert. Code, der diese Variablen nutzt, ist damit wiedereintrittsfähig (reentrant). In den folgenden Abschnitten werde ich beschreiben, wie lokale Variablen deklariert werden und wie sie implementiert sind. Entsprechender Quelltext wird ebenfalls angegeben.

Auch eine lokale Variable ist ein Kompilationswort.

Syntax und Semantik

Meine Implementationsmethode der lokalen Variablen basiert auf sogenannten Gültigkeitsbereichen (scopes). Ein Gültigkeitsbereich ist ein Abschnitt innerhalb eines FORTH-Wortes, über den eine gegebene lokale Variable definiert ist. In Anlehnung an C wird ein Gültigkeitsbereich mit Klammern begrenzt, d.h. `{ ... }`. Auf jede innerhalb der Klammern definierte lokale Variable kann zugegriffen werden, bis die rechte (schließende) Klammer den Gültigkeitsbereich beendet. Im folgenden Quelltext

```
: fool   ( a b – a+b )
  { local b
    local a
    a b +
  }
  ;
```

dienen die lokalen Variablen a und b als Parameter für das Wort `fool`. Bei Verwendung der Worte a oder b wird der Wert der lokalen Variablen auf dem Stack übergeben. Zur Laufzeit wird eine lokale Variable mit dem obersten Wert des Parameterstacks initialisiert. Nach der Initialisierung fehlt dieser Wert auf dem Parameterstack. Das bedeutet, daß lokale Variablen auch als temporäre Variablen genutzt werden können. Beispiel:

```
: foo2   ( a – ? )
  { dup *
    local asquared
    0 local temp
    ...
  }
  ;
```

Diese Definition enthält zwei lokale Variablen. Eine ist mit dem Quadrat des Eingabearguments initialisiert, die andere mit Null. Beliebiger FORTH-Code kann verwendet werden, um eine lokale Variable zu initialisieren.

Die Begrenzer des Gültigkeitsbereichs sind Kompilationswörter wie `if` und `then` und müssen in gleicher Weise paarig sein. Gültigkeitsbereiche können ebenso wie `if ... then` -Paare in beliebiger Tiefe verschachtelt werden. Nachfolgend ein Beispiel dafür wie davon Gebrauch gemacht werden kann:

```
: foo3   ( x y – ? )
  { local y
    local x
    x y + x *
    1000 > if
    { 0 local temp1
        x y *
        local temp2
        ...
     }
     then
   }
   ;
```

```
: fool   ( a b -- a+b )
  {  local b
     local a

        a b +
  }
  ;
```

fool		docol	>r	>r	(rpick)	0	(rpick)	4	+	(rpop)	8	(;)

Bild 2: der kompilierter Code für das Beispiel fool

Die Wörter `temp1` und `temp2` werden nur dann allokiert, falls der Test das Ergebnis "wahr" liefert. Man beachte, daß innerhalb des inneren Gültigkeitsbereichs auf die lokalen Variablen der umgebenden Gültigkeitsbereiche zugegriffen werden kann. Wenn eine lokale Variable in einem inneren Gültigkeitsbereich mit dem selben Namen deklariert wird, wie eine lokale Variable in einem umgebenden Geltungsbereich, dann versteckt die innere Definition die äußere so lange, bis der innere Gültigkeitsbereich geschlossen wird. Das ist die selbe Regel wie bei ALGOL, PASCAL und C.

Lokale Variablen verhalten sich wie Konstanten, denn sie übergeben ihren Wert. Das ist das gewünschte Verhalten, denn lokale Variablen werden gewöhnlich dafür verwendet, Funktionsargumente aufzunehmen, die nicht geändert werden.

Allerdings kann auch der Inhalt einer lokalen Variablen dadurch geändert werden, daß dem Namen ein `to` vorangestellt wird. Das bewirkt, daß der oberste Eintrag des Stacks in der lokalen Variablen abgespeichert wird.

```
variable stdregion    ( standard allocation region )
variable regionptr    ( points to a region pointer )
: dp  reionptr @ ;    ( new definition of dp       )
```

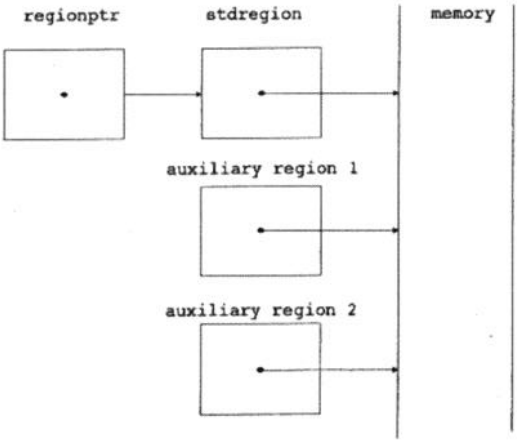


Bild 1: Separate Speicherbereiche

Implementierung

Die oben beschriebene Syntax und Semantik lokaler Variablen erfordert, daß während der Kompilation einer Doppelpunktdefinition die Vereinbarung einer lokalen Variablen in das Wörterbuch eingetragen werden muß, ohne die Kompilation der Doppelpunktdefinition zu stören. Dafür muß außerhalb vom Wörterbuch ein temporärer Speicherbereich eingerichtet werden, der die Definitionen lokaler Variablen aufnimmt.

Diese nur während der Kompilationszeit benötigten transienten Definitionen werden vergessen und ihr Speicherplatz wird freigegeben, sobald der Gültigkeitsbereich geschlossen wird. Wünschenswert ist auch, daß das Allokieren, Initialisieren, Zugreifen und Freigeben lokaler Variablen zur Laufzeit so schnell wie möglich sein sollte.

Lokale Variablen werden auf dem Returnstack gehalten. Eine lokale Variable wird von `>r` schnell allokiert und mit dem Wert auf dem Parameterstack initialisiert. Spezielle Codewörter greifen auf Variablen zu und geben diese auch wieder frei. Um das Problem des temporären Speicherbereichs zu lösen, mußte mein FORTH-System geringfügig modifiziert werden. Die meisten FORTH-Systeme haben eine Variable mit dem Namen `DP` (dictionary pointer - *A.d.Ü.*), die auf den freien Speicherbereich am Ende des Wörterbuchs zeigt. Ich habe eine weitere Stufe der Indirektadressierung hinzugefügt (siehe Bild 1), indem ich zwei Variablen `stdregion` und `regionptr` einführte und `dp`

als Doppelpunktdefinition definierte. Das daraus resultierende FORTH-System verhält sich exakt wie ein Originalsystem. Da jedoch alle die Wörter, die den Platz im Wörterbuch verwalten - Wörter wie `here` , (Komma) und `allot` - auf dp Bezug nehmen, wird es dem Programmierer nun möglich, mehrere Zuweisungsbereiche durch Manipulation von `regionptr` zu verwalten.

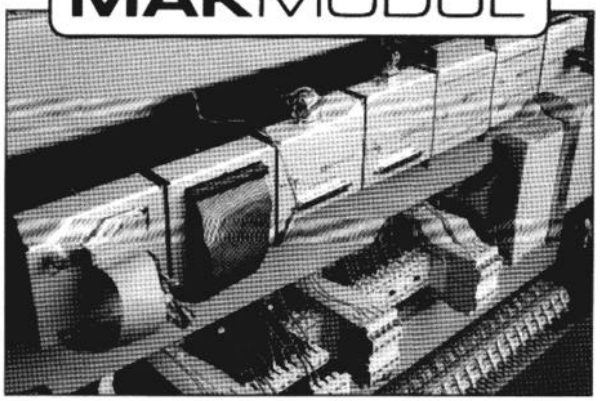
Ein Zuweisungsbereich, der `locregion` heißt, wird definiert, um temporäre Wörterbucheinträge für lokale Variablen aufzunehmen. `local` ist sowohl Definitionswort als auch Kompilationswort. Wenn eine lokale Variable deklariert wird, schaltet `local` den aktuellen Zuweisungsbereich auf `locregion`, verwendet `create` zum Generieren eines Wörterbucheintrags für die lokale Variable und zeichnet einige Informationen auf, die es später gestatten, den Wert der lokalen Variablen zur Laufzeit aufzufinden. Danach schaltet `local` auf den aktuellen Zuweisungsbereich `stdregion` zurück und kompiliert ein `>r`. Die lokale Variable ist ebenfalls ein Kompilationswort, das Code kompiliert, der den Wert vom entsprechenden Speicherplatz im Returnstack auf den Parameterstack überträgt. Die Klammern sind Kompilationswörter, die in erster Linie Verwaltungsaufgaben erfüllen. Die öffnende Klammer { verfolgt, wieviele lokale Variablen deklariert werden. Die schließende Klammer } gibt den durch die Wörterbucheinträge der lokalen Variablen gebundenen Speicherplatz wieder frei, sichert den aktuellen Wörterbuchzeiger und kompiliert Code, der die lokalen Variablen zur Laufzeit entfernt.

Bild 2 zeigt, was in einem 32-Bit-FORTH-System für das erste Beispiel dieses Beitrags kompiliert wird. Das erste `>r` generiert die lokale Variable b und das zweite `>r` die Variable a. `(rpick)` und `(rpop)` sind Codewörter, die ihre Parameter dem Instruktionsstrom (Wörterbuch) entnehmen. `(rpick) 0` kopiert das erste 32-Bit-Wort, das auf dem Returnstack steht, auf den Parameterstack, und `(rpick) 4` kopiert das zweite 32-Bit-Wort des Returnstacks auf den Parameterstack. `(rpick) 8` entnimmt beide lokalen Variablen dem Returnstack (pop-Operation von zwei 32-Bit-Wörtern).

MICROPROCESS

Innovativ, leistungsstark, und einfach zu programmieren:

MAKMODUL® *



Das Prozeßrechner-Konzept für die Meß- und Regeltechnik.

- **Innovativ:** Durch extreme Präzision in der Meß- und Regeltechnik eröffnen MAK-module Ihren Produktionsanlagen und Maschinen ungeahnte Möglichkeiten.
- **Leistungsstark:** Bei laufender Produktion sichern MAKmodule z. B. die Qualität durch Messungen mit einer Auflösung von unter 0,001 mm.
- **Einfach zu programmieren:** Klartext-Programmierung und die große Bibliothek der MAKmodule verkürzen die Programmierzeit entscheidend.

Namhafte Firmen in Europa nutzen den Vorsprung von MAKmodul.
Fordern Sie Informationen an!

MICROPROCESS GmbH
Vertriebspartner der Dr. Weiss GmbH
Division MAKmodul
Talstraße 136 Telefon (0 62 03) 67 81
D-6905 Schriesheim Telefax (0 62 03) 6 81 64

* eingetragenes Warenzeichen der Dr. Weiss GmbH

Lokale Variablen - ein anderes Verfahren

```
\ The following words used in the code are not standard Forth-83 and
\ may not be familiar to everyone.  They provide a way to write word
\ size independent forth code.
\ al+   [ n --- n+sizeof[word] ]  equivalent to 2+ on 16 bit forths,
\                                 equivalent to 4+ on 32 bit forths
\ al-   [ n --- n-sizeof[word] ]  equivalent to 2- on 16 bit forths,
\                                 equivalent to 4- on 32 bit forths
\ a*    [ n --- n*sizeof[word] ]  equivalent to 2* on 16 bit forths,
\                                 equivalent to 4* on 32 bit forths
\

\ The following code words for accessing locals on the return stack are
\ also used.  All three code words use an inline parameter which follows
\ it in the instruction stream.
\ (rpick)  [ --- x ]    return a return stack item n bytes from the top
\ (rstore) [ x --- ]    store x on return stack n bytes from the top
\ (rpop)   [ --- ]      pop n bytes off the return stack
\ In all the above n is found in the instruction stream.

\ 1. The code in this section must be incorporated into the forth kernel

hex

variable stdregion                      \ standard allocation region
variable regionptr                      \ current dictionary allocation region

variable outerdepth                     \ records total number of temporaries
                                        \ created in surrounding scopes.
variable currentdepth                   \ records number of temporaries made
                                        \ in current scope so far.

\ define dp to use an extra level of indirection

: dp   regionptr @ ;    \ ( --- addr )

\ modifiy do and loop to work with scopes

: do            \ ( --- clue here )
 2 currentdepth +!                      \ allow for local variable scopes
 [compile] do                           \ traditional do
; immediate

: loop          \ ( clue here --- )
 -2 currentdepth +!                     \ allow for local variable scopes
 [compile] loop                         \ traditional loop
; immediate

: +loop         \ ( clue here --- )
 -2 currentdepth +!                     \ allow for local variable scopes
 [compile] +loop                        \ traditional +loop
; immediate

\ 2. The code in this section can be incorporated in the forth kernel or
\ loaded into the forth system when needed.

\ Block structure extensions

: region        \ ( size --- ) define allocation region of given size.
 create here al+ , allot
 does> ;        \ ( --- regionptr )



: allocatefrom  \ ( regionptr --- ) start allocating space from given region.
 regionptr ! ;

200 region locregion                    \ create region for making temporary
                                        \ dictionary entries for locals.

: (             \ ( --- oldcurrentdepth region ) begin a scope.
 currentdepth @ dup outerdepth +!
 0 currentdepth !
 locregion @ ; immediate                \ this allows space to be recovered

: )             \ ( oldcurrentdepth region --- ) recover space used by
                \ local headers, unwind vocabulary, compile code to clean
                \ up return stack, and restore depth variables.
 dup locregion @ -
 if drop                                \ if no locals used, do nothing
 else                                   \ otherwise,
    dup name> >link @ current @ !       \    back vocabulary pointer
    locregion !                         \    deallocate space
 then
 currentdepth @ ?dup if
    compile (rpop) a* ,                 \ compile code to clean up rstack
 then
 dup currentdepth !                     \ restore depth counters
 negate outerdepth +! ; immediate

variable to?                            \ 0=@, 1=!
: to    1 to? ! ; immediate

: local         \ ( --- ) create a local variable dynamically allocate
                \ its header from the temporary region.  at run time,
                \ dynamically allocate space for the local on the return
                \ stack.
 locregion allocatefrom                 \ create dict. entries in temp region
 create
    outerdepth @ currentdepth @ + ,     \ record offset from bottom
    1 currentdepth +!
    immediate
 stdregion allocatefrom                 \ revert to standard allocation region)
    compile >r                          \ initialize local
 does>          \ ( addr[offset] --- ) offset is in words from the bottom.
    to? @ if                            \ if to local
       compile (rstore)  0 to? !        \ copy local from pstack to rstack
    else
       compile (rpick)                  \ copy local from rstack to pstack
    then
    @ outerdepth @ currentdepth @ + swap - 1- a* , ; immediate
```

Der Autor

John Hayes ist der Autor mehrerer FORTH-Artikel und eine Schlüsselfigur des VLSI-FORTH-Microprozessorprojekts im Institut für angewandte Physik der John Hopkins Universität.

Das Ablegen von lokalen Variablen auf dem Returnstack macht bei do-Schleifen Probleme. Wenn eine do-Schleife innerhalb eines Gültigkeitsbereichs verwendet wird, wäre es schön, auf die lokalen Variablen dieses Gültigkeitsbereiches innerhalb der Schleife zugreifen zu können:

```
: foo4   ( x — ? )
  {
    local x
    8 0 do
          i x +
          ...
        loop
  }
  ;
```

Das ist einfach realisierbar, wenn zu den Definitionen von do und loop eine einzelne Codezeile hinzugefügt wird, die den Compiler darüber informiert, daß zwei weitere lokale Variablen deklariert wurden[1]. Schön wäre es außerdem, wenn man innerhalb einer do ... loop-Schleife einen Gültigkeitsbereich deklarieren könnte und auf den Schleifenindex i innerhalb des Gültigkeitsbereiches zugreifen könnte:

```
: foo5
  8 0 do
        { 10 local x
          x i +
          ...
        }
        loop
        ;
```

Dieser Fall ist komplizierter, weil ein zustandsabhängiges i gebraucht wird. In meiner Implementierung ist dies nicht gestattet. Eine Implementierung, die einen dritten Stack für lokale Variablen nutzt, würde bei diesem Problem keinen Schaden erleiden.

1 Das setzt voraus, daß die Implementierung von Schleifen zwei Einträge auf dem Returnstack hält.

Zusammenfassung

Viele FORTH-Definitionen sind so einfach, daß durch die Verwendung lokaler Variablen nichts gewonnen wird. Manchmal muß ein Wort allerdings viele Dinge auf dem Stack jonglieren. Dann bringt die Nutzung lokaler Variablen Transparenz in den Code und macht ihn effektiver, indem "schleierhafte" Wortsequenzen vermieden werden, die sonst einen Stack-Crash herbeiführen können.

Literatur

Bowhill, S.: FAST Local Variables. 1982 FORML Conference Proceedings, S. 142-146

Glass, H.: The Implementation of Extensions to Provide a More Writable Forth Syntax. 1983 FORML Conference Proceedings, S. 57- 68

Hart, J.R.: Local Variables. Journal of Forth Application and Research 3,2 1985, S. 159-162

Jekel, R.N.: Local Variables for Forth. 1980 FORML Conference Proceedings, S. 59-63

Korteweg, S.; Nieuwenhuyzen, H.: Stack Usage and Parameter Passing. Journal of Forth Application and Research 2,3 1984, S. 27-50

La Quey, R.E.: Local Variables. 1984 FORML Conference Proceedings, S. 307-316

Volk, W.: Named Local Variables in Forth. 1984 FORML Conference Proceedings, S. 347-357

Die nächste 'Vierte Dimension' erscheint im September '90

Die Sahne

der Programmier-Erfahrung - zu kaufen!

Mitch Bradley in Mountain View, der bei einem führenden Hersteller von UNIX-Maschinen als Senior-Programmierer in C zu Hause ist und dessen Meriten nicht leugnet, bevorzugt dennoch FORTH. Das FORTH aus seiner Küche, bei kalifornischer SUN gereift und nach dem Gusto eines Praktikers abgeschmeckt, dieses FORTH bringt's! Einige der **FORTHMACS**-Features in Stichworten:

Typ
relozierbares Programm, 83-Standard, direkt-gefädelter Code DTC 32-Bit Stacks, unbeschränkter Adreßraum, 680x0-Assembler, Dictionary-Hashing, sauber konstruierter Text-Interpreter, benutzt Text-Files, Mehr-Fenster EMACS Editor eingebunden. Atari: schnelle Grafik(mit Demo), optional System unter GEM. Apple Mac: sowieso im Fenster.

praktisch
Kommandozeilen-Speicher - editierbar - , automatische Wort--Vervollständigung, Liste aller Worte mit gegebenen Anfang, I/O-Umleitung, On-Line-Hilfe, sehr absturzsicher, Show-Crash (Prozessor-Register) und Return-Stack-Trace, Source-Level--Debugger mit Single-Step und Trace, Patch Utility, Auto-Load File, TOS-Kommandos (DIR CD COPY ...).

vielseitig
Multitasking, sämtliche Betriebssystem-Aufrufe incl. Subprozesse, alle I/O-Kanäle, relozierbare Turnkey-Anwendungen, Block-File--Umwandlung, Kommunikation & Filetransfer, Floating-Point optional.

transparent
Disassembler(auch z.B. für TOS) und strukturierter Decompiler, beides quelltextfähig.
230-seitiges DIN-A4 Handbuch und Sources erhältlich.

portabel
Kompatibilitäts-Worte 16- und 32-Bit-Systeme, File-System--Interface, Versionen für Atari-TOS, Apple Macintosh, SUN, OS-9, und 680x0-SBCs. Erhältlich auch als C-Quellcode (abgespeckt) für z.B. UNIX, DEC-VMS, MS-DOS.

Alte Hasen, auch wenn sie sich an mancherlei Beschränkungen längst gewöhnt haben, werden von diesem voll ausgerüsteten 32-Bit FORTH befriedigt. Anfänger werden die ausführliche Dokumentation (engl.) ebenso wie die außerordentlich hohe Absturz-Sicherheit sehr zu schätzen lernen. **Und den Preis !**

Working Disc mit Manual für Atari ST oder Apple Macintosh oder Source Disc für C-Forth mit gedrucktem Glossary kosten **nur DM 121.-** ! Inclusive Forthmacs-Newsletters (38 Seiten) Weitere Produkte auf Anfrage.

Für Vorsichtige:
Atari Forthmacs Working Disc (Shareware) DM 10,- + Versand.
Testbericht in *Vierte Dimension, V.4 Seite 9.*

Bei dieser Qualität ist der Preis nichts als eine Schutzgebühr; Mitch Bradley vertreibt sein Werk eigentlich nur als gemeinnütziges Hobby nebenher. Darum macht er auch keine Werbung und akzeptiert keine Kreditkarten. Als deutscher authorisierter Distributor muß ich es wohl ebenso halten.

Bestellungen bitte schriftlich an:
(gewünschtes Disc-Format angeben)

Klaus-Peter Schleisiek
An den Finkenweiden 38
5100 Aachen

Verzögerte Ausführung von Worten

von Markus Redeker, Hamburg

In diesem Artikel stelle ich ein Konzept vor, Wörter zu erzeugen, die nach ihrem eigentlichen Aufruf noch einmal aktiv werden - eine Verallgemeinerung von PUSH, dessen Tätigkeit nach seinem Aufruf ja auch noch nicht zu Ende ist. Wird PUSH von einem Wort aufgerufen, so bringt es seinen Parameter - eine Variable - sowie deren Wert auf den Returnstack und kehrt dann zum aufrufenden Wort zurück. Aber nach Beendigung dieses Wortes tritt der zweite Teil von PUSH in Aktion, holt Variable und Wert vom Returnstack und setzt die Variable zurück. In der Zwischenzeit kann diese beliebige Werte annehmen, ohne daß das Auswirkungen auf die Zeit danach hat, denn ihr voriger Wert ist ja gesichert.

Der große Nutzen, den PUSH bringt, ist natürlich dadurch begrenzt, daß es nur mit Variablen - genauer gesagt mit 16-Bit-Werten - arbeiten kann. In all den anderen Fällen, wo z.B. eine Variable nicht existiert, oder die falsche Größe hat, aber ein Zustand nur vorübergehend gesetzt werden soll, ist PUSH nicht zu gebrauchen.

In solchen Fällen lassen sich aber mit LATER Wörter schreiben, die dasselbe verzögerte Verhalten wie PUSH haben und auf die jeweilige Lage zugeschnitten sind.

Gebrauch

Quelltext Service

Bevor ich genauer auf LATER eingehe, möchte ich aber zuerst ein verwandtes Konzept vorstellen, das ähnlich funktioniert und einfacher ist.

Es handelt sich dabei um die interaktive Version der verzögerten Ausführung. Sie ist zum Beispiel nützlich, wenn man Wörter testet, die AT verwenden. Will man sich nach deren Ausführung an irgendeiner Stelle des Bildschirmes wiederfinden, so kann man sich das Wort BK schreiben (im Listing auf Block 3), das die Cursorposition sichert. Gibt man eine Zeile ein, die mit BK beginnt, so kann der Cursor wandern, wohin er will: nach Abarbeitung der Zeile ist er wieder zurück. Wie funktioniert nun BK? Denkbar einfach. Am Anfang wird die Cursorposition auf den Returnstack gebracht und am Ende wieder zurückgeholt ... Und dazwischen wird mittels INTERPRET der Rest der Zeile ausgeführt, der dadurch gewissermaßen im Inneren von BK liegt, also auch unter dessen Kontrolle.

Die Version für den nicht-interpretativen Gebrauch wird auf die genau gleiche Weise benutzt. Der einzige Unterschied besteht darin, daß einige Wörter jetzt anders heißen: INTERPRET heißt LATER, >R heißt >LATER und R> heißt LATER>. Und aus dem BK unseres Beispiels wird das Wort STAY (Name aus /2/).

Warnung!

Leider arbeitet LATER nicht gut mit dem Debugger zusammen: Das Tracen von LATER und Worten, die LATER benutzen (in unserem Fall also STAY), führt zu einem Systemabsturz. Worte, die solche Worte aufrufen, können aber wieder normal untersucht werden. Stößt man beim Debuggen unerwartet auf LATER, dann hilft nur noch der Notausstieg mit RESTART (QUIT und ABORT helfen nicht!).

Arbeitsweise von LATER

Um Bezeichnungen zur Verfügung zu haben, sei angenommen, ein Wort X rufe STAY auf. STAY sichert die Cursorposition, dann muß LATER in Analogie zu INTERPRET den Rest von X ausführen und danach wird - nun wieder von STAY - der Cursor zurückgesetzt. LATER muß also den Rest von X vorzeitig in den Interpreter "mogeln", damit er innerhalb von STAY ausgeführt wird.

Weil STAY von X aufgerufen wurde, befindet sich zu seiner Laufzeit an der Spitze des Returnstacks ein Verweis auf die Stelle in X, wo nach Ende von STAY der Interpreter fortfahren soll. Zur Ausführungszeit von LATER liegt darüber noch ein entsprechender Verweis auf STAY. LATER tut nun nichts mehr, als diese beiden Werte zu vertauschen. Und schon wird nach Schluß von LATER zuerst X fortgesetzt und dann erst mit STAY weitergemacht. Oder - von STAY aus gesehen: LATER nimmt einen Wert vom Returnstack, um ihn abzuarbeiten.

Dieser Wert, der vor der Ausführung von LATER auf dem Returnstack liegt, muß berücksichtigt werden. >LATER legt deshalb seinen Parameter auch an die zweite (vom eigenen Standpunkt die dritte) Position des Returnstacks, weil der oberste Wert ja von LATER konsumiert wird. Dagegen könnte man statt LATER> auch R> schreiben - er trägt seinen Namen nur aus Gründen der Datenabstraktion.

Nachträge

Obwohl ich hier vom interaktiven Konzept ausgegangen bin und daraus LATER entwickelt habe - so läßt es sich am einfachsten darstellen - stammt die Grundidee zu LATER eigentlich aus dem ROM der Schneider-CPC-Computer; in /1/ ist sie sehr gut dargestellt. Woher der INTERPRET-Trick stammt, weiß ich nicht mehr. Er ist so einfach, daß er eigentlich nicht neu sein kann - aber in der mir bekannten Literatur wird er nicht erwähnt.

LATER.SCR Scr 2

```
0  \ Later
1
2  : Later r> r> swap >r >r ; restrict
3
4  ' r> Alias later>
5
6  : >later ( n — )  r> r> rot >r >r >r ;
7
8
9  \\  volksFORTH-Definition von PUSH:
10
11  | Create: pull r> r> ! ;
12
13  : push  ( addr — )  r> swap dup >r @ >r pull >r >r ;
14                      restrict
15
```

LATER.SCR Scr 3

```
0  \ Einige Beispiele
1
2  : rv      rvson  interpret  rvsoff ;        \ reverse
3  : inverse rvson  Later      rvsoff ; restrict
4
5  : bk
6    at? >r    >r    interpret  r>    r>   at ; \ back
7  : stay
8    at? >later >later Later later> later> at ; restrict
9
10  : pr [ printer ] print interpret display ; \ print
11
12  \\  Neue Version
13  : push ( addr — )
14    dup >later @ >later Later later> later> ! ; restrict
15
```

Die Namen der Wörter `>LATER` und `LATER>` sind eigentlich nicht sehr glücklich gewählt; mir fiel leider nichts Gutes ein. Wer bessere Ideen hat, möge sich melden.

Zum Schluß möchte ich mich noch bei den Teilnehmern der FORTH-Tagung im April in Aachen entschuldigen, denen ich einen Artikel über `LATER` "für die nächste Ausgabe der *Vierten Dimension*" versprochen habe. Es dauert halt alles etwas länger als man denkt...

Literatur:

/1/ Kapitel 2.3.2 aus: Jörn W. Janneck, Till Mossakowski. ROM-Listing CPC 464/664/6128. Haar bei München: Markt und Technik, 1985.

/2/ Ulrich Hoffman. Interaktive Hinweiszettel auf dem PC. Vortrag auf der FORTH-Tagung 1989. (Listing abgedruckt im Tagungshandbuch.)

Patch fürs volksFORTH (auf Atari ST)

von Robert Epprecht

Beim Arbeiten auf meinem Atari habe ich das volksFORTH als ein Programm von erstaunlicher Qualität schätzen gelernt. Insbesondere die Politik, sämtliche Quelltexte zum System gleich mitzuliefern, ist mir sehr sympathisch, wenn ich auch den verwendeten Targetcompiler schmerzlich vermisse. Diese offene Informationspraxis hat es mir auch ermöglicht, einen Fehler im Atari volksFORTH genau zu lokalisieren und zu beheben:

Im von mir verwendeten Atari volksFORTH arbeitet das Wort `up!` fehlerhaft. Testen Sie selbst:

`up@ up!` führt zum Systemabsturz!

(Eigentlich sollte diese Sequenz ja gar nichts verändern...)

Die Anweisung `disw up!` oder das Studium des Quelltextes (Screen 7 FORTH_83.SCR) offenbart rasch die Ursache: wegen eines Tippfehlers wird dort zweimal das Datenregister D0 statt D6 angesprochen.

Quelltext

```
Code up!   SP )+ D0 move
           $FFFE D0 andi
           D6 UP R#) move
           Next end-code
```

Dies ist natürlich Unsinn! Richtig muß es heißen:

```
Code up!   SP )+ D6 move
           $FFFE D6 andi
           ...
```

Glücklicherweise wird das fehlerhafte Wort vom System nirgendwo weiterhin verwendet, insbesondere auch nicht im Tasker, so daß der Fehler vorerst ohne weitere Folgen bleibt.

Patchvorschlag

Da der Befehl `sp!` mit der gleichen (richtigen) Code-Sequenz beginnt, mache ich mir die Korrektur des Fehlers einfach, und kopiere die fraglichen Befehle einfach von dort:

```
' sp!  @   ' up!  @  6 move
```

Nach Eingabe dieser Wortfolge, können Sie sich nun vom einwandfreien Funktionieren des Wortes up! überzeugen und das korrigierte System mit `savesystem` abspeichern.

Schnelles FORTH für den MC68000 - Teil II

Vorschau auf F68K

von Jörg Plewe,
Großenbaumer Str. 27,
4330 Mülheim

Stichworte

- M68000,
- Native Code,
- Verschiebbarkeit

In der letzten VD habe ich beschrieben, wie man auf dem M68000 ein schnelles FORTH implementieren kann, ohne die wesentlichen Eigenschaften der virtuellen FORTH-Maschine aufgeben zu müssen. Noch einmal zur Erinnerung:

```
: NIP SWAP DROP ;
```

sollte folgenden Code erzeugen:

```
Header von NIP
JSR <Adresse von SWAP>
JSR <Adresse von DROP>
RTS
```

Der Vorteil dieses Codes sollte seine im Vergleich zu seinem gefädelten Gegenstück erheblich kürzere Laufzeit sein.

Dabei habe ich aber ein kleines Problem, den dieser Code mit dem gefädelten gemeinsam hat, vornehm verschwiegen; er ist nicht verschiebbar!

Dies wäre nicht weiter schlimm, wenn man auf Worte wie SAVESYSTEM o.ä. verzichten wollte. Will man aber nicht! Speichert man den Code nun ab, so wie er ist, so kann es passieren, daß das Betriebssystem beim nächsten Start das FORTH-System an eine andere Stelle im Speicher lädt, als es vor SAVESYSTEM noch war. Dies hat dann zur Folge, daß '<Adresse von SWAP>' gar nicht mehr die Adresse von SWAP ist. Das Gleiche gilt natürlich auch für eine gefädelte Implementation, die absolute Adressen ins Vokabular einträgt.

In Betriebssystemen wie GEMDOS behilft man sich hier so, daß das Programm bei seinem Start an die aktuelle Adreßlage angepaßt wird. Dazu ist im Programmfile eine Tabelle enthalten, die angibt, an welchen Stellen im Programm Adressen stehen, die anzupassen sind. SAVESYSTEM oder der Compiler müssen eben diese Tabelle erzeugen. Systeme wie FFORTH oder FORTHmacs bieten solche Mechanismen (bzw. der Target-Compiler zu 32FORTH, *Anm. der Red.*).

Achtung: FORTHmacs Benutzer! Bradley hat es sich etwas zu leicht gemacht:

```
' DUP CONSTANT PTR
: TEST .. PTR EXECUTE .. ;
```

ist nach einem SAVE-REL kein sicherer Tip mehr, da CONSTANT nicht wissen kann, daß die Konstante ein Adresse ist, die reloziert werden muß. Pointer müssen *immer* zur Laufzeit initialisiert werden! FFORTH schreibt hierfür ein ACONSTANT vor. Aber das nur nebenbei.

Die Mimik zur Erzeugung dieser Tabellen ist aber nicht gerade einfach und es muß immer der passende Lader eines Betriebssystems vorhanden sein. Unter OS9 z.B. wäre solch ein System gar nicht ohne weiteres lauffähig.

Ich hätte diesen Artikel gar nicht angefangen, wenn ich nicht auch eine Lösung anzubieten hätte. Ich will nur vorweg sagen, daß diese Lösung eine Spur aufwendiger ist als die oben genannte.

Also Ziel ist es, Adressen so zu behandeln, daß sie von der effektiven Adreßlage des Systems unabhängig werden. Adressen müssen für diesen Zweck *relativ* sein, d.h. die effektive Speicheradresse berechnet sich aus einer Basis und einem Offset. Der erste Schritt wäre also, sich ein Adreßregister auszugucken, sagen wir A5, dem beim Programmstart ein fester Wert, nämlich die Anfangsadresse des FORTH-Systems, zugewiesen wird und alle Adressierungen sich dann relativ auf dieses Register beziehen. Der Code könnte dann in einer ersten Vorstellung so aussehen:

```
Header von NIP
JSR <Offset von SWAP auf
     den Programmstart>(A5)
JSR <Offset von DROP>(A5)
RTS
```

Sieht doch ganz gut aus!

Kleiner Haken: für diesen Code braucht man bereits einen M68020! Da ein FORTH ja beliebig wachsen können soll, müssen diese Sprünge den gesamten Adreßraum erreichen können. Dazu muß der Offset eine Breite von 32 Bit haben. Der M68000 aber ist ein 16-Bit-Prozessor, d.h. Offsets umfassen nur 16 Bit! Schade.

Mit 16 Bit kann man 64 KByte adressieren. Ergo, man teile den Speicher in 64 KByte große Segmente ein (schon mal gehört?). Unser Adreßregister A5 lassen wir nun ins erste Segment zeigen. Nun muß das System aber noch wissen, wo sich die anderen Segmente befinden. Dazu richtet man eine Tabelle ein, in die man die effektiven Startadressen der einzelnen Segmente beim Programmstart einträgt. Diese Tabelle liegt dann im 1. Segment, so daß man sie direkt durch A5 adressieren kann. Der Code könnte dann wie folgt aussehen:

```
Header von NIP
MOVE.L <Offset in die
     Segmenttabelle>(A5),a2
JSR <Offset von SWAP ins
     Segment>(A2)
MOVE.L <Offset in die
     Segmenttabelle>(A5),a2
JSR <Offset von DROP ins
     Segment>(A2)
RTS
```

Wenn man sich beim Kompilieren noch merkt, welche Adresse (welches Segment) gerade in A2 steht, kann man sich den zweiten MOVE-Befehl auch noch sparen, wenn SWAP und DROP im gleichen Segment liegen. Und Daten? Daten könnte man i.A. auf diese Weise nur adressieren, wenn die Segmenttabelle den gesamten Adreßraum abdeckt. (Codeadressen sind i.d.R. > A5, bei Daten muß das nicht so sein) Dies hätte eine 256 Byte große Segmenttabelle zur Folge, die dann wiederum nicht mehr direkt durch A5 adressiert werden könnte, ... Also nicht gerade praktikabel!

Aus dem obigen Ansatz lernt man aber zweierlei; Code und Daten lassen sich nicht immer einheitlich adressieren, und das erste 64kB Codesegment ist wertvoll, da man hier direkt durch A5 adressieren kann.

Folgerung: Trennung von Code und Daten! Dies hat einleuchtende Vorteile. Zum einen sind 64 KByte reiner Code schon ganz schön viel Holz, so daß man in der Regel auf die oben genannte, kompliziertere Adressierung verzichten kann, was natürlich nicht heißt, daß der Compiler sie nicht 'drauf' haben muß. Man kommt aber auch für den Fall großer Codemengen mit relativ wenigen Segmenten aus (für DOS-Freunde am besten 10).

Daten können in beliebiger Größe adressiert werden:

```
MOVE.L <32 Bit Offset>,d0
MOVE.L (A3,d0.l),-(A6)
          \ A6 = Datenstack
```

Man sieht sofort den Nachteil: man braucht ein weiteres Adreßregister, im obigen Beispiel ist es A3, das auf den Start der Daten zeigt. Aber eigentlich hat der M68000 genug von der Sorte.

Also:

```
Header von NIP
JSR <16 Bit Offset von
     SWAP>(A5)
JSR <16 Bit Offset von
     DROP>(A5)
RTS
```

Exemplarisch für einen Datenzugriff:

```
Header von @
    ( Adreßoffset – Wert )
MOVE.L (A6)+,D0
MOVE.L (A3,D0.l),-(A6)
RTS
```

Zusätzlich gewinnt man durch die Trennung noch einiges an Flexibilität, daß man z.B. den Code separat in schnelles statisches RAM (sehr teuer) laden kann, um den Programmlauf zu beschleunigen, und die Daten im dynamischen RAM (nur teuer) beläßt.

Man erkennt, daß man nun nur noch beim Programmstart zwei Adreßregister auf die geeigneten Werte einzustellen hat, und schon ist der Rest von der tatsächlichen Adreßlage völlig unabhängig! Das Wissen um die Möglichkeit, ein adreßunabhängiges FORTH zu erstellen, wird wahrscheinlich die meisten M68000 Anwender nur wenig befriedigen. Deshalb eine freudige Nachricht:

Das Ganze gibt es schon! Jedenfalls fast. Zum Zeitpunkt des Entstehens dieses Artikels ist der neue Kernel, F68K, bis auf die Massenspeicherschnittstelle bereits lauffähig. Wenn Sie den Artikel lesen, ist vielleicht auch das 'bis auf' schon beseitigt. F68K habe ich auf der Basis meines FFORTH-Kerns entwickelt. F68K geht noch etwas weiter als oben beschrieben, in dem es nicht nur unabhängig von der Adreßlage, sondern zusätzlich auch unabhängig von der I/O sein soll. Dies wird erreicht, in dem alle von Hardware oder Betriebssystem abhängigen Funktionen in einen Lader verlagert werden (unter GEMDOS ca. 300 Byte). Damit sollte F68K ohne Neuassemblierung auf jedem M68000-System lauffähig sein, lediglich der Lader muß jeweils neu angepaßt werden.

F68K soll für private Anwender nichts kosten. Interessenten können sich also mit mir in Verbindung setzen. Ich bin stark an Ladern für andere Systeme und F68K-Quelltexten für unseren Quelltextpool (siehe Leserbrief Jörg Staben) interessiert, damit F68K leichter Verbreitung finden kann.

Hinweise für Autoren

Auch in Zukunft möchten wir Beiträge veröffentlichen, die Sie uns hoffentlich in großer Zahl liefern werden. Schicken Sie Ihre Manuskripte bitte an die Redaktion der 'Vierten Dimension' D.LUDA Software, Gustav-Heimann-Ring 42, 8000 München 83, Tel. 089/6708355, FAX 089/6792271 oder legen Sie sie in der FORTH-Mailbox München 'Konferenz Vierte Dimension' ab (8N1 Tel. 089/7259625).

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

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

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

Datenübertragung von Commodore-Plus4 auf den PC

von Claus Vogt

Der Autor hatte das Problem, seine in ultraFORTH auf Plus4 entwikkelten Quelltext-Screens auf einen IBM-kompatiblen PC zu transportieren, um sie dort unter volksFORTH 3.8.1-2 weiterzuverarbeiten. Dieser technologische Sprung erfordert ein gewisses Maß an Soft- und Hardware, das es erstmal zu entwickeln gilt. Die vorgestellten Lösungen sind weder besonders intelligent noch besonders schwierig, können aber sicher dem einen oder der anderen helfen, die Zeit lieber für wichtigere Dinge zu verwenden. Teile der Lösungen sind auch für den Transport vom C64 aus und für den Transport auf andere Zielrechner (z.B. Atari ST) zu verwenden. Es werden Hard- und Software für die Plus4-Seite und geringe Softwareänderungen für die PC-Seite vorgestellt.

Zur Weiterverarbeitung von C16- und C64-Screens auf anderen Rechnern stellen sich folgende Probleme:

- Die Hardware-Verbindung der beiden Rechner
- Ein Übertragungsprogramm auf dem Plus4
- Die Anpassung der Software auf dem PC auf das Commodore-Screen-Format von 25 Zeilen à 41 Buchstaben. Dies erfordert geringe Änderungen im volksFORTH-System, im Editor und im Druckertreiber.

Die Hardware-Verbindung:

Sie ist ganz einfach. (s.Tabelle 1 und 2). Einerseits ist ein geeignetes Kabel erforderlich, ein User-Port-Stecker für den C64, ein RS-232-Stecker für die andere Seite. Dazwischen muß aber noch ein wenig Elektronik her, zur Konvertierung der TTL-Pegel der CBM-Rechner (0 und +5 Volt) auf die vorgeschriebenen RS232-Pegel (+/- 3 Volt bis +/- 15 Volt). Die im User-Port vorhandene 9 Volt-Wechselspannung läßt sich meiner Ansicht nach hierfür nicht verwenden, da sie nicht potentialfrei von der TTL-Versorgungsspannung ist. Ein zusätzliches Netzteil wollte ich mir sparen, so daß die Wahl auf den MAX232 (kostet ca. 10 DM) fiel, der keine hohe externe Versorgungsspannung erfordert. Der Inverter 7406 wurde gewählt, da das DCD- und DSR-Signal bereits im Plus4 mit einem Kollektorwiderstand versehen ist. Da dieses Signal von unserer Schaltung aber gar nicht verändert wird (es liegt immer auf High), ist ein 7404 sinnvoller, der keine Kollektorwiderstände erfordert.

Ein Übertragungsprogramm für den Plus4. (s.CBM-TERM.SCR)

Im Handbuch für den Plus4 ist ein kurzes Terminalprogramm vorgestellt. Es arbeitet mit oben vorgestelltem Kabel durchaus zusammen. Allerdings kann es zum Filetransfer kaum eingesetzt werden, da die zugrundeliegenden Betriebssystemroutinen keinen korrekten Hardwarehandshake machen. Ein XON/XOFF-Protokoll ist im Betriebssystem zwar implementiert, aber nachdem noch nicht einmal das Handbuch dies zu erwähnen wagt, erscheint es auch nicht grade als zuverlässige Variante. Die selbst erstellte Software lehnt sich ein wenig an SERIAL.SCR der PC-Version an. Sie greift direkt auf den Chip des Plus4 zu (ein 6551, der bis zu 19.200 Baud erlaubt, also nix für C64!). Es werden elementare Wörter (`RX TX RX? TX?`) definiert, ankommende Zeichen werden in einem 128 Byte tiefen Buffer per Interrupt zwischengelagert. Höhere Wörter erlauben eine einfache Teletype-Emulation sowie die Übertragung von FORTH-Screens.

Quelltext Service

Notwendige Änderungen an der PC-Software. (s.CBM.SCR)

Dieser Teil bezieht sich auf volksFORTH3.8.1-2. Einzelne Teile sind aber sicher auch auf andere FORTH-Systeme übertragbar (oder eventuell sogar auf die neueren, immer zahlreicher erscheinenden volksFORTH-Versionen). Die Änderungen sind ausschließlich deshalb erforderlich, weil ultraFORTH ein etwas exotisches, dem kleinen Bildschirm angepaßtes Screenformat verwendet. Im Gegensatz zum Standard (der 16 Zeilen à 64 Buchstaben vorschreibt), werden 25 Zeilen benutzt, von denen die ersten 24 je 41 Buchstaben haben, die 25te nur 40 Buchstaben.

Zunächst muß dem FORTH-System das neue Format mitgeteilt werden. Dies ist nötig, da Worte wie: \ und \needs sonst fehlerhaft arbeiten. Dies kann sehr einfach durch Patchen

Stichworte

- Commodore-Plus4
- serielle Datenübertragung
- Screenformat 40mal25
- volksFORTH

Datenübertragung von Commodore-Plus4 auf den PC

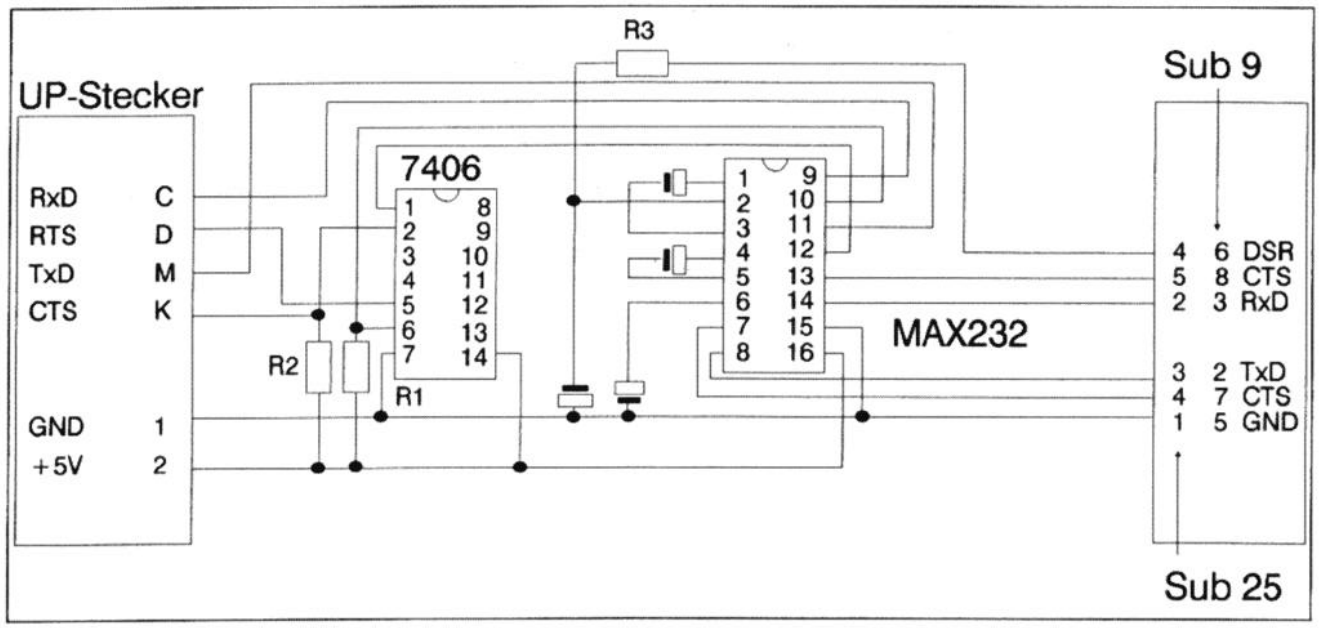


Bild 1: Verdrahtung CBM+4-auf-PC-Kabel

von `C/L` und `L/S` geschehen. Nun ist auch bereits ein Angucken der Screens mit `LIST` problemlos möglich. Hierbei macht sich der fehlende 41te Buchstabe der 25ten Zeile lästig bemerkbar. An seiner Stelle erscheint ein Zufallszeichen, da `LIST` für jede Zeile 41 Buchstaben ausgibt. Ein Schönheitsfehler, mit dem sich leben läßt.

Die Ausgabe auf den Drucker erfordert noch eine kleine Änderung. Da die Screens etwas länger sind, wird es enger auf der Seite. Dies läßt sich durch Umschalten auf kleinen Zeilenabstand (Wort: `8/"`) erreichen. Zumindest auf Endlos-Papier passen damit 6 Screens pro Seite. Die Stelle, an der dieses Wort ausgeführt werden muß, ist je nach Drucker verschieden. Dem EPSON.PRN-Treiber, den ich verwendet habe, reicht es völlig, ein einziges Mal umgestellt zu werden. Der GRAPHICS.PRN-Treiber ist in dieser Beziehung etwas problematischer, da das Wort `NORMAL` jedesmal auf den Standard-Zeilen-Abstand von 6-Zeilen-per-Zoll zurückschaltet. Hier sollte das `8/"` anschließend in `NORMAL` ausgeführt werden. Auch das Betrachten und Verändern von ultraFORTH-Screens mit dem Editor ist durch geringfügige Änderungen in EDITOR.SCR, die in CBM.SCR, Scr 5 aufgeführt sind, einfach zu erreichen. Auch beim Editor ignorieren wir kleinere Schönheitsfehler: Die Statuszeile verdeckt am Anfang die letzte Zeile, die Fehlermeldungen kann man nicht mehr sehen.

An zusätzlicher Software enthält CBM.SCR noch spezielle Worte zum Behandeln von ASCII-Nullen im Quelltext. Commodore-Disketten enthalten auf nicht benutzten Screens gerne allerlei binäres Gewimmel. Sei es vom Formatieren übriggeblieben oder von alten Files: es stört. Es äußert sich beim Ausdrucken als große schwarze Kästchen und kostet Zeit und Farbband. Binäre Nullen sind ein sicheres Indiz für solche binär-Screens. Die Worte `SEARCH0 REPLACE0` bzw. `WIPE0` sind dafür, solche Screens zu finden, die Nullen durch Blanks zu ersetzen bzw. den gesamten Screen mit Blanks zu überschreiben.

Verdrahtung (Bild 1)

Die aus dem User-Port kommenden Anschlüsse C,D,M,K werden (teils über Inverter 7406 durch den (ebenfalls invertierenden Pegelwandler MAX232 geführt) und anschließend nach Bedarf an Sub-d25 oder Sub-d9 geführt.

Der 7406 erfordert Kollektorwiderstände (wer sie sparen möchte, kann einen 7404 nehmen und sie weglassen).

Der MAX232 braucht noch eine Reihe 22μF-Kondensatoren für seine Ladungspumpe (C1-4). Die folgende Tabelle zeigt welche Teile benötigt werden:

Abkürzung	Anzahl	Bezeichnung
UP	1	User-Port-Stecker C64 (paßt auch auf Plus4)
SG	1	Stecker-Gehäuse (passend zu UP, mit Platz für 2 IC's drin)
KA	2 m	9-poliges Kabel
Sub 9 Sub 25	1	Sub-d25 bzw. Sub-d9-Stecker
MAX232	1	Max232-Pegelwandler
7406	1	7406-Inverter (oder 7404)
R1, R2	2	Widerstände 10k (nur, wenn 7406)
R3	1	Widerstand 3k
C1..C4	4	22μF-Kondensatoren
LP	1	kleine Lochraster-Platine

Tabelle 1: Stückliste CBM+4-auf-PC-Kabel

Claus Vogt;
Bülowstr. 67,
D-1000 Berlin 30

Datenübertragung von Commodore-Plus4 auf den PC

SCR# 0

```
\\eer                                clv 04jul89

Serielle Schnittstelle für
Commodore Plus4 über den 6551-Chip
Angelehnt an SERIAL.SCR des PC-volksFORTH
das PC-seitig benutzt werden kann.

Empfangene Zeichen werden in einer 128
Byte tiefen Queue per Interrupt Routine
zwischengespeichert.
Parity und Framing-Errors werden nicht
geprüft.

Die DTR Leitung wird
bedient, je nachdem, ob weitere Zeichen
empfangen werden können.
Der Sender beachtet CTS, sodaß ein
vollständiger Hardwardshake-Handshake
implementiert ist.
Xon/Xoff Protokoll
mit ^S/^Q ist nicht implementiert

Höhere Routinen zum Übertragen von Blocks
und ganzen Disketten sind ebenfalls
implementiert.
```

SCR# 1

```
\ TM:Terminal Loadscreen      cclv28feb89



    1 &11 +thru  \ serial interface
      &12 +load  \ cbm ascii conversion
      &13 +load  \ tx more than byte
      &14 +load  \ dumb terminal
```

SCR# 2

```
\ TM:Serielle Schnittstelle   clv28feb89


| Variable INT \ location of interrupt

| $fd00 Constant Port@   \ 6551
| $fd10 Constant uport@  \ cts-port
| Variable ctrl
| $314 Constant sint@
| sint@ @ >Label oldSint
| $fcbe >Label intEnd    \ system-depend
                \ interrupt end routine

  Create Queue 0 , $80 allot


\\ 0      1     2          130 byte adr
   len    out   128-byte-queue
  len ::= number of characters queued
  out ::= relative address of next
          output character
  (len+out)mod 128 = relative address
              of first byte empty
```

SCR# 3

```
\ TM:Serielle Schnittstelle    clv8feb89

: tx  ( c - )  port@ c! ;

: tx? ( - f )
   port@ 1+ c@  $10 and 0<>
   uport@ c@  $02 and 0<>
   and ;

\\

| Label no  0 # lda
| Label pushaa  sp 2dec  sp )y sta
   puta jmp  end-code

Code tx? ( - f )
   port@ 1+ lda  $10 # and  no beq \ ok?
   uport@  lda  $02 # and  no beq \ cts?
| Label yes $ff # lda  pushaa bne
   end-code

Code tx  ( c - )
   port@ sta  pop jmp  end-code
```

SCR# 4

```
\ TM:Serielle Schnittstelle   clv28feb89

: +dtr port@ 2+ dup c@   1 or  swap c! ;

: -dtr port@ 2+ dup c@ $fe and swap c! ;

: +rts port@ 2+ dup c@ $04 or  swap c! ;

: -rts port@ 2+ dup c@ $f3 and swap c! ;

\\

Code +dtr
   port@ 2+ lda   $1 # or   port@ 2+ sta
   next jmp  end-code

Code -dtr
   port@ 2+ lda  $fe # and  port@ 2+ sta
   next jmp  end-code
```

SCR# 5

```
\ TM:Serielle Schnittstelle    clv8feb89

| Label s-int
    port@ 1+ lda
    0>= ?[ oldSint jmp ]?
    $08 # and  0= ?[ intEnd jmp ]?
    tya pha
    queue lda  $68 # cmp
    0>= ?[ port@ 2+ lda  $f2 # and
           port@ 2+ sta ]? \ -dtr -rts
    clc  queue 1+ adc
    $7f # and  tay  queue inc
    port@ lda  queue 2+ ,y sta
    pla tay  intEnd jmp  end-code
```

SCR# 6

```
\ TM:Serielle Schnittstelle    clv8feb89

: rx? ( - f ) queue c@ ;
Code sei  sei next jmp end-code
Code cli  cli next jmp end-code

: rx  ( - c )
   queue c@ 0=
   IF +dtr +rts  BEGIN queue c@ UNTIL
      THEN
   sei  queue dup  1+ c@  +  2+  c@
   -1 queue +!  1 queue 1+ +!  cli ;

\\
Code rx?  ( - f )
  queue lda  pushaa jmp  end-code

Code rx   ( - c )
  queue lda
  0= ?[ port@ 2+ lda  5 # or
        port@ 2+ sta          \ +dtr +rts
        [[ queue lda  0<> ?] ]?
  sei  queue 1+ ldy  queue 2+ ,y lda
  queue dec  queue 1+ inc
  cli  push0a jmp  end-code
```

SCR# 7

```
\ TM:Serielle Schnittstelle  clv228feb89

: s-init  s-int sint@ ! \ new interrupt
   ctrl @ port@ 2+ !  +dtr +rts ;

: bye  0 port@ 2+ c!  bye ;
  \ -dtr -rts -rxint
```

SCR# 8

```
\                              clv228feb89
```

SCR# 9

```
\ TM: ctrl:                    clv228feb89

| : ctrl: ( adr 8b - ) Create not c, ,
     does> ( 8b - )
           Create dup c@ c, 1+ @ , c,
      does> dup c@  over 1+ @ c@ and
            over 3+ c@ or swap 1+ @ c! ;

\ The only meta-defining-word I ever saw
```

SCR# 10

```
\ TM: baudrate                 clclv8feb89

| ctrl 1+ %00011111 ctrl: bd:

   $11 bd:    50baud
   $12 bd:    75baud
   $13 bd:   110baud
   $14 bd:   135baud
   $15 bd:   150baud
   $16 bd:   300baud
   $17 bd:   600baud
   $18 bd:  1200baud
   $19 bd:  1800baud
   $1a bd:  2400baud
   $1b bd:  3600baud
   $1c bd:  4800baud
   $1d bd:  7200baud
   $1e bd:  9600baud
   $1f bd: 19200baud
```

SCR# 11

```
\ TM: bits stopbits            clv228feb89

| ctrl 1+ %01100000 ctrl: ln:

   $00 ln: 8bits
   $20 ln: 7bits
   $40 ln: 6bits
   $60 ln: 5bits

| ctrl 1+ %10000000 ctrl: st:

   $80 st: 1stop
   $00 st: xstop

\\ xStop gives: 8 bit+1 parity -> 1stop
                5 bit+0 parity -> 1.5
                    other      -> 2stop
```

Datenübertragung von Commodore-Plus4 auf den PC

SCR# 12

```
\ TM: parity                    clv228feb89

| ctrl    %11100000 ctrl: pa:

   $00 pa: noParity
   $80 pa: oddParity
   $c0 pa: evenParity
   $a0 pa: 1Parity
   $e0 pa: 0Parity

| ctrl    %00010000 ctrl: ec:

   $10 ec: +echo \ only if -rts !!
   $00 ec: -echo


\\ 1parity means:
    send and receive high parity
    but no parity-check
```

SCR# 13

```
\ TM: cbm>asc asc>cbm          cclv28feb89

: cbm>asc ( c – c' )
   dup $41 $5b uwithin
   IF $20 + exit THEN
   dup $61 $7b uwithin
   IF $60 + exit THEN
   dup $c1 $db uwithin
   IF $7f and exit THEN
   $dc case? IF $7c exit THEN
   $7c case? IF $dc exit THEN
   $14 case? IF $08 exit THEN
   $08 case? IF $14 exit THEN ;

: asc>cbm ( c – c' )
   dup $41 $5b uwithin
   IF $80 or exit THEN
   dup $61 $7b uwithin
   IF $20 - exit THEN
   dup $c1 $db uwithin
   IF $60 - exit THEN
   $dc case? IF $7c exit THEN
   $7c case? IF $dc exit THEN
   $14 case? IF $08 exit THEN
   $08 case? IF $14 exit THEN ;
```

SCR# 14

```
\ TM: txType txblocks          clv28feb89

$1a Constant #EOF

| : ?break key? abort" ***aborted***" ;


| : ?tx ( c – )
     BEGIN ?break tx? UNTIL
     cbm>asc #EOF case? IF $ff THEN tx ;

: txType ( adr count – )
   bounds DO I c@ ?tx LOOP ;

: txblocks ( fromBlk toBlk – )
   2dup u> abort" nein!"  s-init
   1+ swap DO I block b/blk txType LOOP
   #EOF ?tx ;

: txDisk 0 $a9 txBlocks ;
```

SCR# 15

```
\ TM: dumb Terminal            clclv8feb89

\ Sample for an easy Terminal

\needs #LF      $0a Constant #LF

| : ?rx  pause rx? 0= ?exit rx asc>cbm
      #LF case?  IF cr exit THEN
      #CR case?  IF row 0 at exit THEN
      #BS case?  IF del exit THEN emit ;


| : ?tx ( c –)
    cbm>asc BEGIN ?rx tx? UNTIL tx ;

  300baud noparity 1stop 8bits -echo

  : dumb s-init
     BEGIN BEGIN ?rx key? UNTIL
       key $1b case? IF -dtr exit THEN
           #cr case? IF #CR ?tx #LF THEN
       ?tx REPEAT ;
```

SCR# 0

```
\ Zum Bearbeiten von C64-Screenclv 04jul89
       clv 04jul89
Das Format der C64 & c16- screens ist 40 x 25, wobei
Spalte 41 noch 24 blanks enthält, die allerdings
vom Editor nicht erreichbar sind (40*25+24=1024).

Um dieses Fileformat verarbeiten zu können, sind folgende
Änderungen erforderlich:
- System: Die Layout-orientierten Kommentarzeichen (insb. \)
  erfordern Änderung, ebenso LIST
- Editor
- Drucker-Treiber
- TESTBLOCK zum testen dieser Änderungen
- SEARCH0 REPLACE0 WIPE0 zum Bearbeiten sinnloser Binärdaten
```

SCR# 1

```
\ c/l l/s patchen:  60x25/scr 64x16/scr          clv 04jul89

include editorCB.scr ( s.Screen 5 folgende )

: 40x25      &41  ['] c/l >body !   &25  ['] l/s >body !
             [ printer ]  8/"   [ editor ]  0 ['] dy >body ! ;
: 64x16      &64  ['] c/l >body !   &16  ['] l/s >body !
             [ printer ] 6/"    [ editor ]  1 ['] dy >body ! ;

include EPSON.PRN   3 load
' 40x25 Is 'cold
full  savesystem cbm.com

\\ bei Druckertreibern,
   die 8/" selbst setzten (z.B. GRAPHIC.PRN),
   muß 8/" in NORMAL aufgenommen werden.
```

SCR# 2

```
\ testblock                                    clv 04jul89

  : testblock  ( blk – )
     base push  c/l base !
     b/blk 0 DO
               I extend <# # #> drop c@
               over block  I +  c!
               LOOP
     b/blk 0 DO I c/l /  extend <# Ascii : hold  # #> drop @
               over block  I +  !
               c/l +LOOP drop  update ;

\ zum Testen verschiedener Screenformate. Ausgabe s.SCR 4
```

SCR# 3

```
\ search0 replace0                                clv 28jun89

   : ?abort  stop? abort" aborted" ;
   Defer (action0
   : action0
      capacity 0 DO  I block b/blk bounds DO  ?abort
        I c@ 0= IF J I (action0 THEN LOOP LOOP ;

   : (search0   ( scr adr – ) drop                ?cr . ;
   : (replace0  ( scr adr – ) bl swap c!          ?cr . ;
   : (wipe0     ( scr adr – )
      drop  dup block b/blk bl fill  update   ?cr . ;
   : search0    ['] (search0  Is (action0  action0 ;
   : replace0   ['] (replace0 Is (action0  action0 ;
   : wipe0      ['] (wipe0    Is (action0  action0 ;
```

SCR# 4

```
0:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_1:23456789ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_2:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_3:234
56789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_4:23456789ABCDEFGHIJKLMNOPQR
STUVWXYZ[\]^_5:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_6:23456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_7:23456789ABCDEFGHIJKLMNOPQRSTUVW
XYZ[\]^_8:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_9:23456789ABCDE
FGHIJKLMNOPQRSTUVWXYZ[\]^_A:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\
]^_B:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_C:23456789ABCDEFGHIJ
KLMNOPQRSTUVWXYZ[\]^_D:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_E:
23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_F:23456789ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_G:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_H:23456
789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_I:23456789ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_J:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_K:23456789AB
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_L:23456789ABCDEFGHIJKLMNOPQRSTUVWXY
Z[\]^_M:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_N:23456789ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_O:23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^
```

SCR# 5

```
\\ EDITORCB.SCR enthält gegenüber EDITOR.SCR       clv 04jul89
   folgende Änderungen (volksFORTH 3.8.1-2)

Screen 11:
     $0E # W add   W W add   A I xchg   c/l # C mov
->   $0E # W add   W W add   A I xchg   ' c/l >body #) C mov

     dup 1+ c/row * swap c/l * 'start + video@ (.line ;
->   dup dy + c/row * swap c/l * 'start + video@ (.line ;

Screen 12:
     c/l # C mov   [[  byte lods   stos   C0= ?]
->   ' c/l >body #) C mov  [[  byte lods   stos   C0= ?]

     dup 1+ c/row * swap c/l * 'start + video@ (.line ;
->   dup dy + c/row * swap c/l * 'start + video@ (.line ;
```

Lokale FORTH-Gruppen, die sich regelmäßig treffen:

1000 Berlin Claus Vogt, Tel.: 030/2168938. Treffen am letzten Donnerstag des Monats um 19.30 Uhr in der Technischen Universität Berlin, Mathematikgebäude, 6.Stock im Raum MA 621

4130 Moers 1 Friederich Prinz, näheres Tel: 02841/583 98

Rhein-Ruhr Jörg Plewe, Tel: 0208/423514, Treffen nach Absprache. Der nächste Termin kann bei Jörg Plewe erreichbar unter obiger Telefonnummer erfragt werden.

6100 Darmstadt Andreas Soeder, Tel. 06257/2744. Treffen an der VHS an einem Mittwoch in der Mitte des Monats (Termine: 13.9, 11.10, 15.11 13.12 im alten Pädagog, Raum 3-1, 3.Stock).

6800 Mannheim Lokale Gruppe Rhein-Neckar, Thomas Prinz, Tel.: 06271/2830, Ewald Rieger, Tel.: 06239/8632. Treffen jeden ersten Mittwoch im Monat im Vereinslokal des Segelflugvereins Mannheim e.V. Flugplatz, Mannheim-Neuostheim.

7000 Stuttgart Lokale Gruppe Stuttgart, Wolf-Helge Neumann Tel.: 0711/882638 und Ulf Katzenmaier, Tel.:0711/268293.

8000 München Heinz Schnitter, Tel. 089/3103385 oder Christoph Krinninger 089/7259382. Treffen jeden 4. Mittwoch im Monat 19 Uhr 30 im Vereinsraum 2 im Bürgerhaus Unterschleißheim am Rathausplatz (S-Bahnhaltepunkt S1 Unterschleißheim).

DDR-Leipzig FORTH-Gruppe Leipzig, Michael Balig, Lützner Plan 17, DDR-7033 Leipzig oder Dr. Jürgen Hesse, Lieselotte-Herrmann-Str. 40, DDR-7050 Leipzig, Tel.: 041-69 56 02

FORTH-Fachgruppen:

8000 München RTX 2000 Gruppe, Koordinator Herr Krämer, Treff- und Zeitpunkt wie oben bei der lokalen Münchner Gruppe.

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

Es möchten in ihrer Region eine Gruppe gründen:

3300 Braunschweig Martin Holzapfel, Bassestr.17.
8500 Nürnberg 20 Thomas G. Bauer, Fichtestr. 31, Tel. 0911/538321.
5000 Köln 60 Michael Heycke, Boltensternstr.
4830 Gütersloh 1 Ludwig Röver, Holzheide 145A

Eine Fachgruppe will gründen:

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

Hier kann man um Rat fragen:

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

Ansprechpartner zu bestimmten Interessengebieten:

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

FORTH-Gesellschaft e.V. - Postfach 1110 - D-8044 Unterschleißheim
Tel.089/3173784, FORTH-Mailbox Tel.: 089/7259625 oder 08841/5880
Postgiroamt Hamburg, Kontonr.: 563211-208 BLZ 20010020

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

EDV-Beratung – Software-Design – Goppold

Bgm. Germeierstr.4 – 8011 Poing – Tel.: 08121-82710

Wir haben das Forth Know-How:

* Consulting, Projekt-Management, Beratung, Schulung.
* Auf 68000er, SUN-Workstations (SPARC&68k), PC-Systeme, Forth-Prozessoren.
* FORTH unter UNIX (alle Systeme).
* Eigen-Entwicklung fortgeschrittener Software-Technologie: Objekt-Programmierung, Datenbanken, Hypertext.
* Vermittlung von US-Software zu US-Preisen: LMI, Harvard Softworks, Forthmacs.
* Und natürlich Leibniz, das System nach Forth.

Das FWD-Team Ludwig Richter bietet:

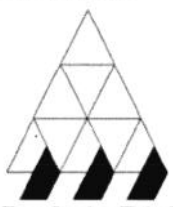
Kommunikation Automation Meßwerterfassung Industriesteuerungen

Hard/Soft-Lösungen, auch unorthodoxe

Vollständige Lösungen vom Sensor über die Elektronik, bis zum Aktor

Unsere Effizienz und Flexibilität beruht auf den Faktoren:

- Wir sind ein Team aus Spezialisten verschiedener Richtungen
- Wir nutzen das Know-How auf Ihre Anwendungen bezogen
- Wir regen Ihre Mitarbeiter und Maschinen an
- 10 Jahre Erfahrung Assembler — PC 68000 Z80 6809 6502
- 10 Jahre Erfahrung physikalische Meßtechnik / Nachrichtentechnik
- 7 Jahre Erfahrung Forth — Metacompiler UR/FORTH PC/FORTH CFORTH
- 5 Jahre Erfahrung analoge und digitale Schaltungsentwicklung
- 3 Jahre Erfahrung C — PC AMIGA ATARI Crosscompiler



FWD-Team
Bernhard Emese Reinhard Fenger Jürgen Gerhard Ludwig Richter

Ludwig Richter, Essenheimerstr.94, D-6500 MZ-Bretzenheim, Tel.06131-368274

FORTH-SYSTEME

ANGELIKA FLESCH

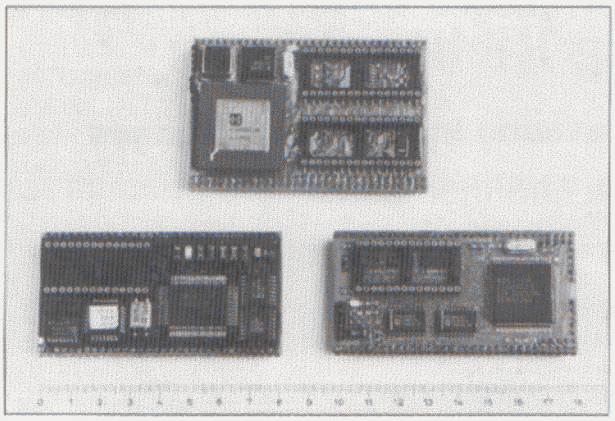
Postfach 1103
D-7814 Breisach

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

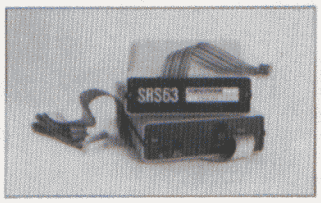
Entwickeln Sie romfähige Programme ?

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