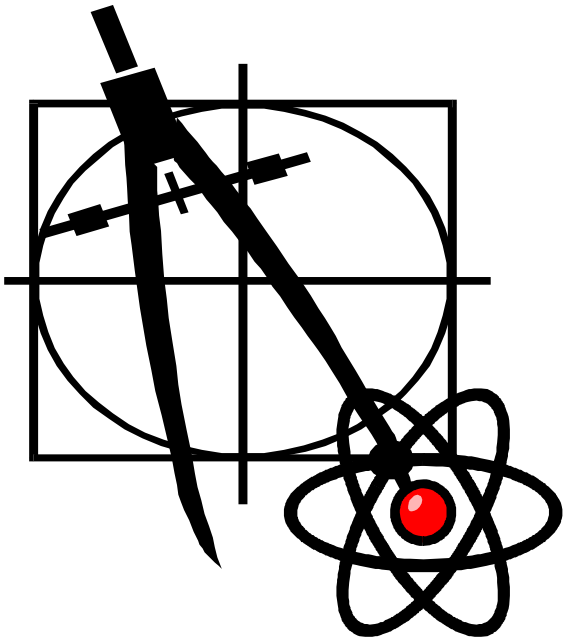
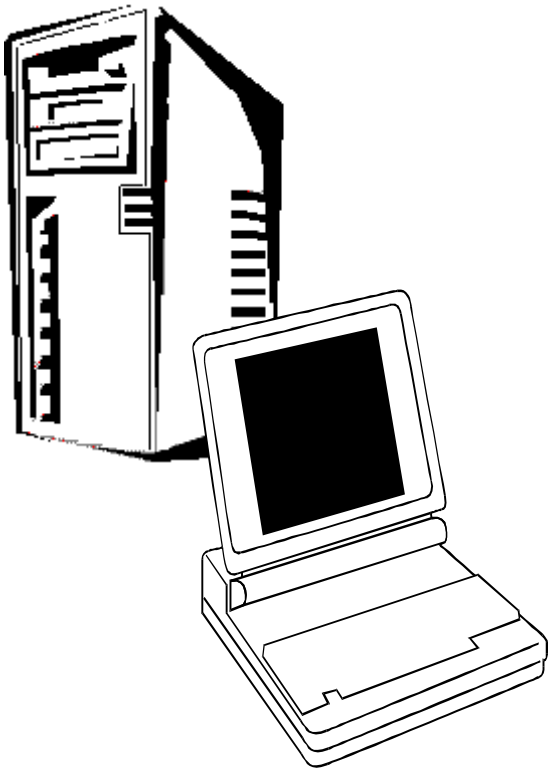
VIERTE DIMENSION

**für Wissenschaft und Technik, für kommerzielle EDV,
für MSR-Technik, für den interessierten Hobbyisten.**





In dieser Ausgabe:

15. Jahrgang - **Ausg. 4/’99** - 10,- DM

Organ der FORTH-Gesellschaft e.V.

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

Inhalt

In der nächsten Ausgabe finden Sie voraussichtlich:

- den zweiten Teil des Aufsatzes über das 'Hashing'; von Friederich Prinz
- einen Artikel CFA2NAME; von Wolfgang Allinger
- einen weiteren Beitrag über Sprünge bis 64 kByte mit ZF's Assembler; von Fred Behringer
- einen Artikel über ein Graphikpaket für F-PC; von Gert Bretschneider
- und was immer Sie uns und den Lesern der VD mitteilen wollen

Editorial

IMPRESSUM

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth-Gesellschaft e.V.
Postfach 16 12 04
D-18025 Rostock
Tel.: 0381-400 78 28
E-Mail:
SECRETARY@FORTH-EV.DE
DIREKTORIUM@FORTH-EV.DE

Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208

Redaktion & Layout

Friederich Prinz
Hombergerstraße 335
47443 Moers
Tel./Fax:: 02841-58 3 98
E-Mail:
VD@FORTH-EV.DE
FRIEDERICH.PRINZ@T-ONLINE.DE

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluß 1999

März, Juni, September, Dezember
jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

DM 10,- zzgl. Porto u. Verp.

Liebe Leser,

Die letzte VD in diesem Jahr ist fertig, rechtzeitig zum versprochenen Termin. Die Zeitschrift ist voll und die unterschiedlichen Beiträge, Aufsätze und Zuschriften bieten sicher jedem unserer Leser etwas. Für dieses Jahr bedanken wir uns für das Engagement jedes einzelnen ‘Schreibers’. Alle Autoren haben dazu beigetragen, unsere Zeitschrift zu füllen und dabei gleichzeitig ein wenig von dem wiederzuspiegeln, was unser ‘Vereinsleben’ ausmacht.

Das Rätsel um den ‘schlampigen Elektriker’ hat bei Ihnen großen Anklang gefunden. Im Vergleich zu der ‘Anzahl’ Zuschriften, die uns gewöhnlich zu einzelnen Artikeln erreichen, hat Georg Beierleins Beitrag sogar regelrecht ‘eingeschlagen’. Das freut nicht nur den Rätselsteller. Vielleicht ermuntert es Sie, ähnliche Rätsel oder Aufgaben vorzulegen.

Vielleicht hat Sie das Rätsel aber auch zu ähnlichen philosophischen Betrachtungen animiert, wie den diesjährigen Hüter des Drachen, Fred Behringer, der uns gleich einen Anstoß zu Überlegungen inbezug auf *Voraussetzungen* ‘geliefert’ hat (siehe dieses Heft). Fred Behringer erhofft sich davon eine Diskussion über Voraussetzungen im allgemeinen und beim Programmieren im besonderen. Wir hoffen natürlich, daß diese Diskussion sich in entsprechenden Beiträgen für die VD niederschlägt.

Eine noch viel regere Beteiligung Ihrerseits ist nicht nur aus Sicht der ‘VD-Macher’ wünschenswert. Die jeweils nächste VD soll eben nicht nur voll werden, sondern voll mit Themen, Anregungen, Hinweisen und Beiträgen sein, die Sie uns geben. So wünschen wir uns zum Beispiel ‘Reviews’ zu Zeitschriften und/oder Artikeln, die Sie besonders lesenswert fanden. ‘Threads’ im Netz, die Ihr Interesse geweckt haben, sind ganz sicher auch eine Meldung in der VD wert. Geben Sie es uns einfach...

fep

VD im Internet

Die VD auch im Internet wiederfinden zu können, wird in den Netzwerkforen, in E-Mails an die Redaktion und auch in persönlichen Gesprächen immer wieder als Wunsch geäußert. Unsere Zeitschrift ist nicht nur für die Mitglieder der Forthgesellschaft interessant ! Es spricht auch wenig dagegen, die VD - mit entsprechender, zeitlicher Verzögerung - ‘allgemein zugänglich’ zu machen. Woran es scheitert ist einmal mehr der Wille zur Tat, sprich: Wir brauchen Jemanden, der uns 4 Mal im Jahr die dazu notwendige Arbeit machen will !

fep

Quelltext Service

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können diese Texte auch direkt bei uns anfordern und sich zum Beispiel per E-Mail schicken lassen. Schreiben Sie dazu einfach eine E-Mail an die Redaktionsadresse. *fep*

Georg Beierlein's Rätsel um den 'schlampigen Elektriker' hat offensichtlich vielen Lesern der VD großen Spaß bereitet. Lösungsvorschläge kamen weniger, als wir in der Redaktion zunächst gedacht hatten. Georg's Aufgabe war wohl doch recht 'kniffelig'. Das freut nicht nur die Direktoren, denen allesamt keine 'vernünftige' Lösung eingefallen ist, was man vielleicht mit dem Genuß geistiger Getränke erklären mag, die anläßlich des Treffens der Direktoren im Harz 'gereicht' wurden. Dort wurde uns das Rätsel erstmals vorgestellt.

Immerhin sind aber einige Lösungen eingegangen, die man als korrekt bezeichnen muß. Gratulation !

fep

Betreff: Der schlampige Elektriker
Datum: Mon, 19 Jul 1999
Von: joerg@jpohl.de (**Joerg Pohl**)

Hi Friederich,

gerade die niegelnagelneue VD aus dem Kasten gezogen und gleich (weil von hinten nach vorn gelesen) über den Elektriker gestolpert.

Also ich weiß zwar nicht, wie es der Hausbesetzer gemacht hat, ich würde die Aufgabe jedoch folgendermaßen lösen.

3 Schalter, 3 Lampen, angenommen es funktionieren alle, reicht es ja aus 2 zu detektieren. Da (normale) Glühlampen die Eigenschaft haben, warm zu werden, wird dies ausgenutzt. Also beliebiger Schalter ein, Bier getrunken (Prost!), Schalter wieder aus und einen anderen Schalter ein. Dann, oh Qual, die Treppe hochsteigen, sollte zu schaffen sein, bevor Lampe
sich abkühlt und nachsehen bzw. -fühlen.
Ergo Problem gelöst.

Joerg Pohl

Betreff: VD 3/99: Rätsel
Datum: Tue, 20 Jul 1999
Von: **Guido Quick** <quick@uni-wuppertal.de>

Liebe Redaktion,

hier schnell mein Lösungsvorschlag:

Schalter a, b und c müssen Lampen 1, 2 und 3 zugeordnet werden. Vorgehen:
Schalter a einschalten
1-2 Minuten warten (eventuell auch das Bier trinken)
Schalter a auschalten, Schalter b einschalten
zum OG gehen und folgende Erkenntnisse umsetzen:
die leuchtende Lampe wird mit Schalter b bedient
die nichtleuchtende warme Lampe wird mit Schalter a bedient
die nichtleuchtende kalte Lampe wird mit Schalter c bedient

G. Quick

...und bei Guido's Lösung wird der Zweck des Bieres glasklar. Das Erwärmen einer Glühbirne muß abgewartet werden. Wie lange soll gewartet werden ? Eine 'Zigarettenlänge' ist als zeitliche Einheit ein wenig 'out'. Eine Flasche Bier ist eine recht passable Einheit mit ausreichend zeitlichem Spielraum...

Sub: 25Jul99
Datum: Mon, 26 Jul 99
Von: v.vinerts@genie.com (**Henry Vinerts**)

1. To Georg Beierlein (is he another professor from Mittweida?):
My answer to his puzzle (on page 34) goes something like this--
Let's start with all three light switches in the OFF position.
Turn on Nr.1, then drink the beer slowly, with enjoyment; turn on Nr.2; grab the empty beer mug and run quickly upstairs (hopefully, the beer has not taken its effect yet, and you can do it safely);
use the mug (upside down), if necessary, to step up on and to reach the lightbulbs; feel the area around the two light bulbs that are lit--the one around which the ceiling, the shade, or what have you, is hotter, was turned on by switch
Nr.1;
switch Nr. 2 turned on the other; the one that is not turned on belongs to switch Nr.3.
I forgot to tell you to bring a piece of paper and pencil and to write this down, because by now the beer may have hit you, you might lie down, fall asleep and forget the solution.

Henry

...und eine Lösung der ganz besonderen Art hat Fred Behringer beigesteuert. Die befindet sich auf der Seite 7.

Betreff: Beitrag für die VD
Datum: Thu, 29 Jul 1999 16:06:45 +0200
Von: **Klaus Schleisiek** <xxxxxxx@xxxxxxx>

...daß ich "die Gift Culture" nicht so richtig verstanden hätte, lasse ich nicht auf mir sitzen. Hier also mein Beitrag dazu.
Zu Freeware und ihre Anwendung auf synthetisierbare Hardwarebeschreibung.
Die Lizenzbedingungen von GNU haben zu einer Blüte kooperativen Austausches und permanenter Produktpflege geführt. Das brachte im Falle Linux Software hervor, die alle anderen breit verfügbaren Betriebssysteme an Zuverlässigkeit und Effizienz übertrifft.
Es wäre schön, wenn es auch frei verfügbare, synthetisierbare Hardware gäbe, die, wenn es die Performance erfordert, angepaßt/verändert werden kann. Das setzt zwar zusätzlichen Lernaufwand gegenüber einem als Hardchip erhältlichen Prozessor voraus, dafür ist man aber nicht den schwarzen Käfern unbeeinflußbarer Hersteller ausgeliefert, die mit ihren ver-

druckst zugegebenen Macken eine Portierung immer wieder in ein Abenteuer verwandeln.
Die GNU-Bedingungen können zu einem kommunalen Pool von Software führen, weil GNU auf Basis des Copyrights für geschriebene Werke (Autorenrechte) eine rechtlich starke Position hat. Wer gegen die Pflicht zur Rück-Veröffentlichung angefügten Codes verstößt, dem kann vor Gericht mit dem Urheberrecht der Garaus gemacht werden. Und das Urheberrecht greift deshalb, weil es verdammt viel Arbeit ist, diesen ganzen Code zu erstellen und zum Laufen zu bringen. Selber Nachprogrammieren ist irgendwie keine sinnige Alternative, wenn man die Bedingungen der Free-Software-Foundation nicht akzeptieren will.
Bei der synthetisierbaren Beschreibung eines Prozessorkerns ist die Situation so, daß ein Profi sie nach Studium der Architektur in 1/2 Jahr nachgebaut hat. Eine Größenordnung, die das Copyright als rechtliche Basis für die Durchsetzung der Lizenzbestimmungen dahinwelken läßt. Wenn denn das Programm, das die Architektur realisiert, nicht viel hergibt, dann bleibt halt die Architektur selbst. Und die ist als Idee nicht schutzfähig. Von der Idee kann man aber Logikgatterstrukturen ableiten, die zusammengelötet etwas für den Menschen "maschinieren" und als Erfindung patentierbar sind. Das Patent gibt dann der Hardware eine vergleichbare rechtliche Basis, wie das Copyright der Software. Und die rechtliche Basis wird z.B. auch gebraucht, um MicroCore im Erfolgsfalle nicht im Inkompatibiltätsmodus vermicrosoften zu lassen.
DESHALB habe ich für die MicroCore Architektur ein Patent angemeldet.
Zu den Lizensierungsbedingungen habe ich folgende Vorstellungen, die für mich noch nicht zuendegedacht sind:
MicroCore ist im Sourcecode öffentlich verfügbar, so daß nach Herzenslust damit experimentiert und entwickelt werden kann. Wenn der Anwender den synthetisierbaren Code, den er um den Core herum entwickelt, seinerseits wieder frei veröffentlicht ("derivative work"), dann darf er den Core ohne Lizenzzahlung nutzen und damit Produkte realisieren und vermarkten. Will er seinen eigenen Code nicht frei veröffentlichen, dann muß er Lizenzgebühren zahlen. An wen? Nun, da bietet sich im deutschen Rechtsraum ein eingetragener Verein an, dem ich die Nutzungsrechte am Patent übertragen werde. Dieser Verein hätte dann die Aufgabe, zu beschließen, was mit den Lizenzeinnahmen geschehen soll. (Ich möchte z.B. die Kosten der Patentanmeldung erstattet bekommen!)
Und solange es noch keinen Verein gibt, und solange es die synthetisierbare Hardwarebeschreibung für MicroCore noch nicht gibt, tausche ich nur mit denjenigen Designinformationen aus, die sich bereiterklären, dies solange vertraulich zu behandeln, bis der erste MicroCore "steht".

Hamburg, 28.7.99
Klaus Schleisiek

Betreff: Tools zum Download
Datum: Fri, 02 Jul 1999
Von: **Ulrich Paul** <upaul@paul.de>

Hi,

ich weiß nicht, ob Ihr es schon wißt: Auf meiner Homepage
www.paul.de
gibt es unter Download 2 Erweiterungen für FPC. Sie sind frei (GNU Public License).

Die eine räumt mit SWAP, DROP, ROLL, PLUCK, SUCK ... auf. Man schreibt einfach

```
REORDER ( otto fritz karl friedrich -- friedrich otto karl
                fritz otto )
```

und danach sieht der Stack auch so aus. Solange links im Stackkommentar jeder Bezeichner nur einmal vorkommt (Bezeichner = ASCII-String, beliebig, gilt nur lokal) und rechts keiner, der nicht links vorkommt, ist alles erlaubt. REORDER erzeugt Assembler-Code, ist also verdammt schnell. Man kann es aber auch interaktiv verwenden, es ist state-smart.

Die andere Erweiterung ist mehr syntaktischer Zucker: SWITCH:

Beispiel:

```
SWITCH: rechtsrum? linksherum rechtsherum \ deklarieren
            \ des Schalters und Benamung seiner Zustände

linksherum rechtsrum? . --> 0
rechtsherum rechtsrum? . --> -1
```

Die Erweiterung habe ich einmal geschrieben, da ich mich in einem größeren Projekt mit dem ON und OFF so verzettelt hatte, daß es einfach zu mühsam wurde. (Ein weiterer Grund war der, daß ich auf dem Target-Prozessor nicht eine ganze Speicherstelle für jeden Schalter opfern wollte. Das SWITCH: für FPC ist ein Port des ursprünglich für den Target-Compiler gebauten.)

Bin ja gespannt, wer mir jetzt als erstes die Lynchjustiz androht wegen schwerster Vergehen gegen Forth.

CU
Ulrich Paul

Die Stricke zu denen wir greifen, sind die Hash-Threads im Dictionary. Damit lassen sich vortrefflich viele Stunden der Freizeit 'erwürgen' (siehe Beitrag in diesem Heft, Seite 23), aber keinesfalls Forther, die sich ihre Arbeit erleichtern ;-) Und unsere Zeitschrift lebt (auch) von 'provokanten' Zuschriften. Also bitte, weiter so !

fep

Rätsel - Versuch einer Lösung

von Fred Behringer
<behringe@mathematik.tu-muenchen.de>
Planegger Str. 24, 81241 München

Ein großes Lob dem Sohnemann, dem wir diesen "Mindcracker" aus der VD 3/99 verdanken! In Amerika hat sich Martin Gardner mit Fragen zur Unterhaltungsmathematik einen Namen gemacht. Das Ziel des Rätsels ist klar formuliert. Ein Beweis ist nicht nötig. Die Angabe eines Lösungsalgorithmus zur Erreichung des Ziels IST der Beweis. Wie steht es aber bei diesem Rätsel mit den Voraussetzungen? Zugegeben, Rätsel leben von offengelassenen Voraussetzungen und man sollte bei seinen Forderungen nicht zu streng sein. Viele Voraussetzungen verstehen sich von selbst. Fliegen kann der Mensch im Rätsel natürlich nicht. Und mit dem Kopf durch die Wand auch nicht. Darf er blind sein? Oder kleinwüchsig? Wie klein? Und wenn er sich beim Kegeln beide Hände verstaucht hat? Sind die Lampen normal? Was heißt normal? Ich verwende nur noch Sparlampen in Birnenform. Es soll schon welche geben, die keine merkliche Wärme mehr abstrahlen. Hat die erste Etage vielleicht eine Falltür als Zugang? Eine Tür mit eingelassenem Glas? Schließlich nicht ganz ungewöhnlich! Wie steht es mit den Schaltern? Weiß man bei jedem Schalter, in welcher Stellung er eingeschaltet ist? Kann die Möglichkeit ausgeschlossen werden, daß der schlampige Elektriker Kippumschalter verwendet hat, in welchen er zufallsbedingt mal die eine, mal die andere Hälfte als Schaltbahn an die Drähte geschlossen hat? Ich würde als "Elektriker" in die Bastelteilekiste greifen und auf solche Feinheiten nicht achten.

Ein mathematischer Satz ohne genaueste Angaben der Voraussetzungen ist undenkbar. Man weiß dann nicht, auf was man sich beim Beweisen beziehen darf. Ein physikalisches Experiment ist ohne Angabe der Voraussetzungen wertlos. Es kann dann nicht nachvollzogen werden. Ein Algorithmus ist ohne Angabe der Voraussetzungen unbrauchbar. Man kann ihn dann nicht maschinell überprüfen.

Man merkt, das Thema "Voraussetzungen" ist mir diesmal ein Anliegen. Das Rätsel selbst war nur Anlaß. Ein willkommener Anlaß. Mein Dank dem Rätselsteller!

Ich versuche im folgenden, verschiedene Voraussetzungskombinationen sauber zu formulieren, und gebe verschiedene Lösungen des Rätsels an. Ich numeriere die Voraussetzungen hexadezimal durch und stelle sie dann je zu je als Hexadezimalzahl dar.

Mögliche Voraussetzungen

Vollständigkeit oder Systematik wird nicht angestrebt.

(1) Aus den Stellungen der Bedienele mente läßt sich eindeutig erkennen, welcher Schalter ein- und welcher ausgeschaltet ist.
(2) Der Rätselmensch ist in seinen Bewegungsabläufen nicht behindert.
(3) Der Mensch ist klein, aber mit dem Krug als Untersatz schafft er es, an die Lampen zu kommen.
(4) Der Mensch ist blind, sein Tastsinn und sein Wärmeempfinden sind ungetrübt.
(5) Die Lampen sind Kaltstrahler (Biolampen mit Glühwürmcheneffekt).
(6) Die Lampen sind "normale" Glühlampen. An der Wärmeabstrahlung läßt sich (durch Anfassen) erkennen, ob die Lampe eingeschaltet ist oder ob sie mehr als t_1 sec ausgeschaltet war.
(7) Allein durch Anfassen, bei geschlossenen und mit einer lichtundurchlässigen Kapuze verdeckten Augen, läßt sich erkennen, ob eine Lampe länger als t_2 sec eingeschaltet war, ob sie gerade eben erst eingeschaltet wurde oder ob sie länger als t_1 sec ausgeschaltet war.War sie nur geringfügig länger als t_2 sec angeschaltet, so ist sie handwarm und bleibt weitere etwa t_1 sec lang so.
(8) Es herrscht Fliegeralarm. Die Fenster sind verhangen.
(9) Man kann den Schein einer eingeschalteten Lampe der ersten Etage vom Parterre aus sehen.
(A) Man kann den Schein einer eingeschalteten Lampe der ersten Etage nur dann sehen, wenn die Tür zu den Gemächern der ersten Etage einen Spalt offen ist.
(B) Die Tür zur ersten Etage ist eine Feuerschutztür, die von selbst zufällt.
(C) Die Frau des Rätselmenschen ist gerade woanders. Er hat keine möglichen Helfer.
(D) Die Lampen lassen sich auch in eingeschaltetem Zustand "herausdrehen".
(E) Der Mensch kann die Lampen, jede für sich, herausdrehen.
(F) Der Mensch ist hinreichend groß, so daß er an die Lampen kommt.

Was ich hier entwickle, hat große Ähnlichkeit mit den Sprechblasen in den Abenteuerdenkspielen für den Bildschirm. Mathematik und logisches Denken sind wesentlich interessanter, als die Lehrer uns das in unserer Schulzeit klarmachen konnten. Es muß aber beides erst gelernt werden. Selbstverständlich ist hier gar nichts.

Mögliche Handlungen des Rätselmenschen

Vollständigkeit oder Systematik wird nicht angestrebt. Ich verwende "nachsehen" im Sinne von "in die erste Etage gehen und überprüfen".

(1) Er klemmt den Krug nach Verlassen der ersten Etage zwischen Tür und Pfosten, um sie einen Spalt offen zu halten.
(2) Er benetzt die agierende Hand (Links- oder Rechtshänder, einarmig amputiert?) mit Bier, um sich nicht zu verbrennen.
(3) Er trinkt das Bier aus (ex!), stellt den Krug umgekippt hin und steigt drauf, um an die Lampen zu gelangen.
(4) Er beobachtet über die Fenster der ersten Etage den Lichtschein.
(5) Er beobachtet über den offenen Spalt der Tür zur ersten Etage den Lichtschein.
(6) Er schaltet Schalter 1 etwas länger als t_2 sec lang an, dann wieder aus, und dann Schalter 2 an. Schließlich geht er nachsehen.
(7) Er schaltet Schalter 1 an, geht nachsehen und dreht eine der nichtangeschalteten Lampen heraus.
(8) Er schaltet Schalter 1 und 2 an, geht nachsehen und dreht eine der angeschalteten Lampen heraus.
(9) Er schaltet Schalter 1 aus und probiert an den bis dato noch nicht eingeschalteten Schaltern, welcher davon die erste Etage bei Einschalten erhellt.
(A) Er schaltet die beiden eingeschalteten Schalter wieder aus und probiert an ihnen durch einzelnes Einschalten aus, welcher Schalter Licht ins Dunkel der ersten Etage bringt.
(B) Er fühlt an der Lampentemperatur, welche Lampe länger als t_2 sec angeschaltet war.
(C) Er fühlt an der Lampentemperatur, welche Lampe gerade brennt.
(D) Er sieht mit seinen Augen nach, welche Lampe brennt.

Lösungen

Ich gebe hier für verschiedene Kombinationen von Voraussetzungen mögliche Lösungen an. Ich schreibe links die jeweilige Voraussetzungskombination, rechts die Kombination von Lösungsschritten an. Die genaue Ausformulierung der Lösungswege bleibe dem Leser überlassen. Vollständigkeit strebe ich nicht an.

Es ist äußerst schwierig, alle Voraussetzungen oder Nichtvoraussetzungen und alle möglichen Einzelhandlungen elementar aufzuspalten und genau anzugeben. Die Schwierigkeiten der Textaufgaben aus den Schulbüchern und den Semestralen sind von dieser Art. Ich achte bei den kompakten Angaben der Einzelhandlungen nicht auf die Logik der Reihenfolge und lasse auch zu, daß sie sich überlappen.

Natürlich läuft meine Darstellungsweise auf die Anfangsgründe einer aussagenlogischen Behandlung hinaus. Das war mein eigentliches Anliegen.

(127F) —» (6BD)
(12378ABC) —» (1236BD)
(12467CF) —» (6BC)
(1259CDEF) —» (48A)
(1258ABCEF) —» (1579)

Weitere solcher Implikationen aufzustellen, sei den verehrten Leserinnen und Lesern zur Übung empfohlen.

Vom Rätselsteller wurde wahrscheinlich die Antwort aus der ersten Zeile erwartet: Alle Schalter seien zunächst ausgeschaltet. Er schaltet Schalter 1 länger als t_2 sec, aber nicht wesentlich länger, an und dann wieder aus. Dann schaltet er Schalter 2 an und geht nachsehen. Die handwarme dunkle Lampe gehört zu Schalter 1, die helle zu Schalter 2, die kalte dunkle zu Schalter 3. - Blind darf er dabei nicht sein und mit Verbänden an den Händen auch nicht aufwarten. Er darf auch nicht allzu kleinwüchsig sein. Ansonsten beachte man, wie viele die Handlungsfähigkeit einschränkende Zusatzvoraussetzungen bei diesem Lösungsweg noch hätten gemacht werden können (siehe Zeile 2 der die verschiedenmöglichen Voraussetzungen und Lösungen darstellenden Implikationen, aber auch die anderen Zeilen)!

Ein neues Rätsel

Wieviele Lampen kann der Rätselmensch unter den Voraussetzungen (12367CD) durch einmaliges Nachsehen in der ersten Etage eindeutig den Schaltern im Parterre zuordnen?
Hier sei nur eine einzige Möglichkeit der Voraussetzungskombination angegeben.Weitere Rätsel über weitere Voraussetzungskombinationen anzugeben, sei dem geneigten Leser anempfohlen. Man bringe in Tabellenform oder über einen Metaalgorithmus Systematik ins Geschehen.

Noch'n Rätsel

Unter welchen, nach der Girlandenpartie der durchzechten vorausgegangenen Nacht, auf der er sich beim Grillen beide Hände verbrannt hat, er trägt Verbände, naheliegenden Zusatzvoraussetzungen kann der Mensch durch einmaliges Nachsehen in der ersten Etage sieben Lampen sieben Schaltern eineindeutig zuordnen?

Nota bene:

Irgendwie habe ich jetzt beim Korrekturlesen den Überblick verloren. Ich hätte die Voraussetzungen und Teilhandlungen wohl doch noch etwas sauberer formulieren sollen, jeden einzelnen umgangssprachlichen Begriff. Vielleicht versucht es mal jemand anders! Aber ist es nicht beim Programmieren ähnlich? Stehen wir nicht immer wieder vor dem gleichen Dilemma? Auch in Forth! Gerade in Forth! Man denke nur an die Stackakrobatik, die man laufend zu bewältigen hat. Akribisches Aufzeichnen der getroffenen Voraussetzungen hilft da sehr.

Fred Behringer

Forth in Rußland

Zusammengestellt von Fred Behringer

Seit kurzem gibt es den Begriff der FIG RU, vorerst allerdings nur in Form einer Webseite, nämlich http://www.forth.org.ru . Webmaster ist Andrey Cherezov. E-Mail: ac@eserv.ru . Es handelt sich um einen losen Zusammenschluß von Forth-Enthusiasten aus ganz Rußland. Dieser besteht hauptsächlich darin, daß den "Mitgliedern" (Mitgliedschaft kostenlos) Platz auf dem Webserver reserviert wird. Außerdem können sie sich dort einen E-Mail-Anschluß einrichten und haben Zugang zu den Newsgroups und sonstigen Foren. Auf der Webseite sind auch Links zu allen gängigen Forth-Adressen, in Rußland und sonst in der Welt, zu finden. Wir sind auch darunter. Allerdings nur in Form unserer alten Adresse. Ich werde dafür sorgen, daß die neuere aufgenommen wird. Eine Zeitschrift gibt es (noch) nicht. "Es gibt im Internet", sagt Andrey, "viel bessere Möglichkeiten, als sie eine gedruckte Zeitschrift bieten könnte." Na ja, diese Argumentation kennt man inzwischen. Wieviele Forth-Freunde sind denn schon per E-Mail zu erreichen? Nein. Es gibt Berechtigung für beides, Elektronisches und Gedrucktes. Da muß man dann aber erst mit der Zeit dahinterkommen. Computer-Bücher verschwinden auch nicht schon deshalb vom Markt, weil viele von ihnen sich auf einer beigelegten CD selbst enthalten. Außerdem muß man das elektronische Material erst einmal haben. Aber dann kann man es auch gleich drucken. So herum könnte man auch argumentieren. - Andrey und mindestens ein anderer der auf der Webseite vertretenen Forth-Freunde aus Rußland sind auch Mitglied bei der FIG US. Lange haben wir auf Forth-Aktivitäten aus Rußland gewartet. Jetzt ist es soweit. Ich werde den Kontakt pflegen und weiterhin berichten. Andrey bietet von einem Klickfeld auf der Webseite aus sein SP-Forth zum Herunterladen an. Ich werde es mir gelegentlich ansehen. Auf der Webseite sind neben interessanten Informationen in lateinischen Buchstaben (Englisch) auch viele undefinierbare Zeichen aus dem oberen ASCII-Bereich zu finden. Von meinen Übersetzungskontakten zu dem russischen Forth-Dimnsions-Parallel-Artikel-Autor (1995) her habe ich in Erinnerung, daß das die ASCII-Codes sind, die in einem russischen Zeichensatz den kyrillischen Buchstaben freigemacht werden. Mal sehen, ob es mir gelingt, meinen Netscape Navigator mit einem russischen TTF zusammenarbeiten zu lassen.

Aufruf

Eine russische FIG ? Was es nicht alles gibt ! Die Welt wird wirklich täglich bunter, auch ohne Windows ;-)

Haben Sie selbst Kenntnis von einer 'exotischen FIG', oder einer Gruppe, die sich - warum auch immer - mit Forth oder verwandten Themen auseinander setzt ? Bitte berichten Sie uns doch davon.

fep

Gehaltvolles

zusammengestellt und übertragen
von Fred Behringer

FORTHWRITE der FIG UK, Großbritannien

Nr. 103, August 1999

1 Editorial

Chris Jakeman begrüßt zwei neue Mitglieder, die über das Internet zu FIG UK gestoßen sind, und stellt fest, daß es doch interessant sei, in den E-Mail-Unterschriften ("Signatur") einen Hinweis auf die Mitgliedschaft bei der FIG UK aufzunehmen, um so ein Mehr an Reklame anzufügen. Eine Idee, die uns auch gut anstünde?

2 Forth News
Dave Abrahams

Platinen für das FIGUK-Hardwareprojekt können bestellt werden - comp.lang.forth hat ständig steigenden Zuspruch (Anton Ertl), Eiffel und ADA lassen nach - John Verne hat die Forth-FAQs auf den neuesten Stand gebracht - die 15. euroForth-Konferenz wird vom 17.- 20. September 1999 in Petersburg abgehalten - Forth-Web-Site nun auch in Japan - ShBoom für 10 Dollar: www.ptsc.com - Ficl - Forth für Lego - Minotaur - TpForth - API-Programmierung in Win32Forth.

4 Forth and Genetic Programming
Chris Ramsay

Es wurde Zeit, daß auch dieses Thema von Forth-Enthusiasten aufgegriffen wird. Die Genetische Programmierung, als Methode, die der Natur abgeguckt wurde, aber auch fast als Paradigma, existiert nun schon seit geraumer Zeit. Dies ist höchstwahrscheinlich der erste Artikel weltweit, der die Genetische Programmierung mit Forth zusammenbringt. Gene, Überkreuzreproduktion, Mutation, Selektion, Überleben des Bestangepaßten - und das alles in Forth, wo man interaktiv miterleben kann, wie sich das Programm Muster und Zusammenhänge selbst erkundet, die dem Programmierer zur Zeit der Programmierung nicht bekannt sind oder gar nicht bekannt sein können.

Als Beispiel geht der Autor der Frage nach einem Programm nach, das aus dem vorgegebenen Monat heraus die Anzahl der darin enthaltenen Tage "berechnet". Vorgehen: Eine Untermenge von zulässigen Forth-Worten, als "Gene" nur solche Wortgruppen, die den Stack ausgeglichen halten, ein Kriterium, das die Zielerreichung mißt (Auswahlkriterium) - und einen Genetischen Algorithmus, der auf das Problem losgelassen wird. Die Einschätzung des Autors: Bei größerer Komplexität enttäuschend.

Bemerkungen des Rezensenten: Wenn schon die zu suchende Lösung eines Problems über Evolutionstechniken erreicht werden soll, warum nicht dann auch die Bestandteile der Genetischen Programmierung selbst? Hat sich mal jemand mit Genetischer Metaprogrammierung oder Metagenetischer Programmierung beschäftigt? Und wie sollen die Metakriterien aussehen? Interessant die kritischen Aufsätze zum Darwinismus, die man kürzlich im Spektrum der Wis-

senschaft lesen konnte. Sind es wirklich unsere unseligen "Stärksten", die gute Chancen haben zu überleben? Die "Bestangepaßten" kommen dem schon eher näher. "The fittest" umfaßt beides. Aber manchmal schaffen es ja auch die Nichtangepaßten am besten, die Individualisten. Es kommt auf das gerade geltende Auswahlkriterium an. Wer aber wählt das aus (siehe oben)? Es bleibt wohl doch bei der Suche nach der allesbestimmenden Metakraft als Hauptaufgabe des Menschen (?)

12 Keeping in Touch
Chris Jakeman

Alle FIGUK-Mitglieder, die eine E-Mail-Adresse haben, bekommen jetzt regelmäßig Offerten von Firmen durchgestellt, die Forth-Programmierer suchen und sich derobhalben an die FIGUK-Redaktion gewandt haben. Gibt es bei unseren Jüngeren eigentlich für sowas auch Interesse?

13 Figuring it out with Win32Forth
Dave Pochin

Und weiter geht's mit Daves Eintauchen in die möglichen Fallen von Win32Forth, ein Unterfangen, das den Lesern und Leserinnen helfen soll, sich gar nicht erst in solchen Schwierigkeiten zu verfangen. Dave gehört zu denen, die bis dato ständig geäußert hatten "wozu brauche ich denn Graphik in meinem Forth?", bloß "weil sie zu bequem waren, sich mal ein bißchen mit den sich anbietenden Möglichkeiten zu beschäftigen". Win32Forth hat mit seinen schnell begreifbaren Graphik-Beispielen sein Interesse geweckt und jetzt zeigt er uns, wie man seinen Text ganz schnell mit Win32Forth-Graphiken beleben kann. Es geht um den "Windows Device Context" (als ein Paket von Objekten) und das Graphic Device Interface (GDI). Er "create"t einen Schreibstift (pen) als ein "gutes Beipiel, Windows nach Win32Forth zu übertragen". Daves Anstrengungen sind prima, aber ... seit zwanzig Jahren benötige ich darstellbare Beispiele für mittelpunktskonvexe Kurven, die strikt p-quasikonvex, aber nicht p-konvex sind. Kann mir Windows da helfen? Vielleicht in Verbindung mit einem genetischen Algorithmus à la Seite 4? Das ist das Schöne an Forth: Forth ist überall. Man wird nicht unbedingt auf eine bestimmte Richtung festgenagelt.

19 Deutsche Forth-Gesellschaft

Da sind wir wieder. Alle vier Monate. Unsere Anzeige zur Mitgliederwerbung und zur allgemeinen Reklame für uns. Mit allen inzwischen eingetretenen Adreßänderungen. Prima! Danke, Chris.

20 Finite State Machines - 3a, Test Vectors
Graeme Dunbar

Das ist der dritte Teil (und wie es aussieht, die erste Hälfte davon) einer Serie von Forthwrite-Artikeln über Automaten mit endlichvielen Zuständen. Von CAD-Programmen her kennt man den Begriff eines Testvektors. Mit Testvektoren kann man einen solchen Automaten eine vorgegebene und wiederholbare Sequenz von Einzelzuständen durchlaufen lassen. Graeme verwendet für seine Testvektoren (4 Seiten Forth) selbstmodifizierende Datenstrukturen, so wie sie von Leo Brodie und Alan Winfield beschrieben werden. Weiterhin nimmt er Bezug auf Julian Noble und Anton Ertl. Julian Noble, so sagt Graeme, verwendet Automaten mit endlichvielen Zuständen zum Durchforsten (Parsen) von Gleitkommazahlen in wissenschaftlich-technischer Notation. Ich (der Rezensent) werde mir das für Arbeiten, die ich gerade vorhabe, merken.

27 Did you know ...

Zusammenhänge zwischen Postscript, das von vielen Druckern verstanden wird, und Forth. Mit einem Beispiel zur Fassung von Postscript-Befehlseinheiten als Forth-Worte nebst Syntaxübertragung.

28 FIG UK Hardware Project
Jeremy Fowell

HC11-Einplatinen-Computer mit Pygmy-Forth nahezu fertig. Preis und Informationen über die bestehende Mailing-List. Näheres in Forthwrite 100. Wird auch in die FIGUK-Webseite gestellt.

30 Forth Interest Group UK :: Revenue Account for year to 31 March 1999

Hier wird der volle Bericht des Kassenwarts Keith Metthews veröffentlicht. Seine Feststellung: Vierzehn Jahre lang lag der Jahresbeitrag bei 10 englischen Pfund (etwa DM 30,-). Um die gestiegenen Kosten zu decken, schlägt er vor, ihn auf 12 Pfund zu erhöhen - oder mehr Mitglieder zu werben.

31 AGM

Die Allgemeine Mitgliederversammlung der FIG UK findet am 4.9.99 in Morden (in London am südlichen Ende der U-Bahn Northern Line - soweit habe ich mich nie aus Großlondon hinausgetraut, wahrscheinlich, weil mein Visitors Ticket nicht soweit hinausreichte) statt.

32 ByteForth for MCS51 cpu's
Willem Ouwerkerk

Über das laufende "Igel"-Hardware-Projekt unserer niederländischen Forth-Freunde wurde im Vijgeblaadje, in der Forthwrite und bei uns in der VD bereits berichtet. Hier ist das Forth dazu: Der Prozessor AT89Cx051 - optimierender Macro-Compiler - interaktives Austesten - Flash-Programmierer - beträchtlicher Vorrat an ausgetesteten Worten - optimierender Compiler - Programmbeispiel - so sieht die Entwicklungsschleife aus - weitere Vorhaben.

37 The FIG UK Awards

Die Gewinner sind: (Soweit sind wir, daß wir solche im Deutschen überflüssige Wendungen übernehmen. Was ich sagen wollte, ist: Den Preis haben gewonnen:) Philip Preston für seine gründlichen Beiträge zum WebForth-Projekt und Paul Bennett für seine Serie "Reading/Writing the World" in der Forthwrite, die auch im Web veröffentlicht wird.

38 Letters

Zwei Briefe an die Redaktion über Schwierigkeiten mit wohl doch ein bißchen zu "genial" programmiertem MAX und MIN als Antwort auf eine Mitteilung aus FW 102, der eine Brief vom Rezensenten, der andere von Willem Ouwerkerk, dem sehr aktiven Redakteur des holländischen Vijgeblaadjes.

Bitte sehen Sie auch die Anzeige der
FIG UK
auf der Seite 16 !

Hallo, Friederich!
Zwei Monate sind seit meinem letzten Brief vergangen, und das liegt teilweise daran, daß das April SVFIG Treffen auf den dritten Samstag im Monat verlegt wurde und ich an diesem Wochenende nicht in der Stadt war.
Ich nehme an, daß Jeff Fox seinen angekündigten Vortrag über die letzten Entwicklungen mit Chuck Moore's F21-Chip hielt, und daß Dr. Ting so ziemlich den Rest des Tages auffüllte, wie er es üblicherweise tut, wenn nicht genug Vorträge vorliegen. Skip Carter war nicht in der Lage teilzunehmen, aber er kam zum 22. Treffen. Ich werde darüber als nächstes berichten (hoffentlich noch in diesem Brief).
Traurige Neuigkeiten erreichten die Forth Gemeinschaft Anfang Mai: Robert Reiling starb am 5. Mai. Er war der frühere President der FIG (1984) und die treibende Kraft hinter den FORML Konferenzen in Asilomar. Wir werden ihn vermissen.
Das Mai-Treffen zog ein große Menge an, über 30 im Laufe des Tages, und ich denke daß es nicht nur an Dr. Tings Gegrilltem zu Mittag lag. Es waren zumindest 25 von uns gleich zu Beginn da, um 10 Uhr morgens, die John Rible's Vorlesung in Chip Design hörten; Jeff Fox's Kamera und Laptop bauten einen Link mit der Zuhörerschaft (etwa ein halbes Dutzend) in fernen Plätzen der Welt, einschließlich Kanada und Ost-Deutschland. John lehrt einen Kurs an einer Zweigstelle der Universität von Kalifornien und nutzt ein Xilinx-Board und Verilog als Werkzeuge. Weitere Details könnt Ihr unter "*http://www.ultratechnology.com/cpuclass.htm*" erfahren. Es scheint, daß John einem Trend in der gegenwärtigen Ausbildung entgegenwirken will, die nach seinen Worten, Absolventen produziert, die gern gewaltige Programme schreiben anstatt dazu angehalten werden, kleine Module zu programmieren, zu testen und zusammenzufügen (wie z.B. in Forth).
Nach dem Mittagessen gab uns Skip Carter, in seiner üblichen effizienten Art, eine Zusammenfassung von Forthsystemen die unter Unix im allgemeinen und insbesondere unter Linux laufen. Hier ist die Liste der Systeme mit ihren Autoren, die ich aus meinen Notizen zusammengestellt habe (alle Schreibfehler stammen wahrscheinlich von mir):

iForth von Marcel Hendrix, das einzige kommerzielle System, Editor;
ThisForth von Wil Baden, läuft auf allen möglichen Systemen, manchmal etwas langsam;
PFE von Dirk Zoller (Portable Forth Environment);
GForth von Ertl, Paysan, & Wilke, ANSI comform, gutes Manual;
BigForth von Bernd Paysan, ANSI, Linux, Windows, GUI Generator;
pForth von Phil Burk, Unix, Mac, Win95/NT, Sadler benutzt für Ficl;
kForth von Krishna Myneni, Linux, Win95, x-y plot für Win95;
eForth von Andy Valencia & Francois R..(?), für viele Plattformen;
DynOOF von Andras Zsoter, Linux.

Alle Compiler sind auf "*http://www.forth.org/compilers.html*" verfügbar.
Auf die ganze Diskussion zurückblickend muß ich sagen, daß "kein Meister komplett ist, solange er nicht seine eigenen Werkzeuge geschaffen hat"
("No master is complete unless he makes his own tools").
Und dann kam der ursprüngliche Meister von Forth, und er macht immer noch seine eigenen 'Werkzeuge. Chuck bekam eine Stunde, um soviele Fragen wie möglich, die er vorher erhalten hatte, zu beantworten. Auch wenn er in der ersten Stunde des Tages begonnen hätte, anstelle der letzten, hätte er
wahrscheinlich nicht die ganzen Neugierigen zufrieden stellen können.
Jeff Fox's Batterien waren um diese Zeit schon leer, aber ich glaube, er hat es hinbekommen ein JPEG-Bild von Chuck früher am Morgen um die Welt zu schicken. (Ich hoffe eines Tages eine Grafik anzufertigen, die aufzeigt wie lange es dauert, das Bild einer Person über den Ozean zu schicken, begonnen mit der Zeit von Kolumbus. Wieviel Zeit haben wir noch auf der Asymptote?)
Ok, ich werde meine "Plauderei" kurz fassen und Euch einige von Chucks Antworten und Gedanken zusammenfassen, bevor ihr noch einige Seiten an die VD anhängen müßt und der freundliche Professor aus Sachsen dazu übergeht die "Les hommes de bonne volonte" zu übersetzen.
Chuck sagte, daß einige Gelehrte aus dem Silicon Valley da waren, um einen Blick auf die TV-Box zu werfen, die die iTV Company herstellt, und sie "ästhetisch elegant und anachronistisch" nannten. Würden solche Adjektive heute auch auf Forth passen? Ja, er ist immer noch bei iTV, aber die Arbeit an seinen eigenen neuen Werkzeugen ist interessanter. ColorForth ist das neue Werkzeug, Chuck wird es wahrscheinlich bald ins Web legen, die gesamten 1 oder 2 kByte, die es braucht! Ihr werdet wahrscheinlich ebenfalls in Kürze über sein neues Konzept von ICE hören (ist wirklich "cool"!), Interpretation, Compilation und Execute in einem "Atemzug". Das ist es, was Forth von C unterscheidet. Die C Leute haben die Fähigkeit zur Compilation zur Laufzeit verloren.
Ein paar Worte zum Standard:
"Entsprechend Goedel kann ein selbst-bezügliches, formales System nicht komplett sein, somit kann man Forth nicht in Forth beschreiben. Der Standard läßt Forth zu kompliziert erscheinen."
Chuck sagte, daß er den Standard umschreibt, so daß er seinem Geschmack entspricht, und er wird viel einfacher sein. "Schon, der Standard bringt Glaubwürdigkeit und erzeugt bedeutungsvolle Debatten und, außerdem, ohne Standard wäre ich nicht hier, um über ihn zu verpfuschen." "Er muß noch mehr ausgeputzt werden. Wir können die 'Double Words' loswerden. Die meisten Maschinen haben sowieso 32-Bit."
"Wenn ich reich wäre, würde ich das tun, was ich jetzt tue. Ich würde meinen letzten Chip in Produktion bringen."
"Jeder sollte seinen eigenen personalisierten Editor für Programme haben."
"Ich denke nicht, daß es der Natur von Software innewohnt, daß sie sehr umfangreich sein muß, es liegt stattdessen in der Natur unserer Gesellschaft 'floatware' zu produzieren. Wir

können die Microsoft Industrie durch 0,1 % des Codes ersetzen, den sie produziert hat. Wir sollten nicht mit einer Welt voller mittelmäßiger Programme zufrieden sein."
"Falls Y2K schlecht ausgeht, werden wir neue Verordnungen haben, es mag einen Druck zu mehr Kompaktheit geben, darüber wieviel jeder Chip enthalten darf. Es könnte möglich sein Forth für diese Zukunft zu nominieren. Da gab es 1989 ein dänisches Buch "Die Nutzer Illusion". Vielleicht sollten wir die Illusion der Nutzer über Forth zerstreuen. Sollten wir Forth-Uniformen, Logos, T-Shirts, Wimpel u.ä. herstellen, um ein neues Image aufzubauen?"
So viel für den Moment, meine Freunde, Auf Wiederhöhren!
Henry
(übersetzt v. T. Beierlein)

Forth Interest Group International (FIG USA)

Wollen Sie mit der ganzen Welt verbunden sein und dabei Ihr Englisch perfektionieren ? Amerika ist ein wesentlicher Teil der ganzen Welt. Zumindest, was Forth betrifft. Über tausend Mitglieder aus allen Ländern sind bei uns.
Werden Sie auch ein Mitglied in der Amerikanschen Forth-Gesellschaft (FIG-USA).
Für 45 Dollar im Jahr (Studenten zahlen 18 Dollar) bekommen Sie 6 Hefte unserer Vereinszeitschrift Forth Dimensions und genießen auch sonst verschiedene Vorteile. In den Heften erfahren Sie Forth-Neuigkeiten aus aller Welt, neue Produkte, Literatur, Forth-Ideen, fundiertes Wissen, Artikel auch für Einsteiger, Projekte, Leser-Diskussionen, Quelltexte, Hinweise auf Internet-Verbindungen, kostenlose Forth-Systeme und vieles mehr. (Für Übersee-Porto müssen wir leider noch 15 Dollar hinzurechnen).

Unmittelbare Informationen über uns bekommen Sie, wenn Sie auf der Homepage der Deutschen Forth-Gesellschaft “Links zu anderen Forth-Organisationen” und dann “Forth Interest Group (USA)” anklicken.

Ansonsten bekommen Sie Auskünfte über das amerikanische Forth-Büro:

Forth Interest Group
100 Dolores Street, suite 183
Carmel, California 93923
USA

oder auch vom Redakteuer, Marlin Ouverson, unter der E-mail Adresse:

E-Mail: **office@forth.org** oder
editor@forth.org

Hallo, Friederich,
Ich bin froh, daß Du und Marlin direkt Ideen über Eure Beilage zur Forth Dimensions ausgetauscht habt. Ich hoffe, daß das FIG-Büro zumindest allen interessierten Teilnehmern ein zustimmendes Kopfnicken gibt.
Ich schrieb Dir schon ein wenig über das Juni-Treffen der FIG, aber vielleicht sollte ich etwas mehr hinzufügen, sonst werde ich mich wohl nicht so fühlen als ob ich meinen 'Job' gemacht habe.
Wir trafen uns am 26. Juni wie üblich im Cogswell College. John Rible hielt seine zweite Lektion über CPU Entwurf, so waren etwa 15 bis 18 'Studenten' ganz zum Anfang da. Vier oder fünf hatten das Xilinx-Board gekauft und machten Ernst mit praktischem ('Hands-on') Lernen. Zu weiteren Details über die Lektionen empfehle ich "*http:// www.sandpipers. com/cpuclass2.html*" zu besuchen, da mein Bericht der Vorlesung sicher nicht gerecht wird.
Der einzige andere Sprecher war Dr. Ting, aber er hatte, wie üblich, über genug zu sprechen, um den ganzen Nachmittag auszufüllen. Als erstes hatte er den Xilinx XC4005XL FPGA benutzt, um einen 7-stimmigen Synthesizer zu bauen, den er auch vorführen konnte. Danach bschrieb er seine neue Version 2.0 von eForth. Version 2.03 ist für die 8086-Platform. Die Worte READ, LOAD und SEND wurden hinzugefügt, um das Firmware Paket zu vervollständigen.
Schließlich, um eForth's Arbeitsfähigkeit auf dem P16-Prozessor zu testen, hatte Dr. Ting an einem P16 Simulator gearbeitet, dessen technische Beschreibung einige Zeit brauchte. Doch die meisten Zuhörer blieben trotzdem dabei.
Unsere Treffen sind fast wie in der Schule: die meisten 'Schüler' scheinen die Pausen-Aktivitäten den Lektionen selbst zu bevorzugen. Während der Pausen findet man Unterhaltungen und Debatten nahezu über jedes Thema, und manchmal muß unser Tagungsleiter George Perry seine Stimme mehrmals erheben, um die Leute zurück in die Reihen der unbequemen Schulstühle zu bewegen. Aber, im Namen von Forth, wir versammeln uns und werden das hoffentlich noch oft tun.
Also, alles Gute bis zum nächsten Mal!

Henry

Vormerken !

Forthtagung 2000 in Hamburg

Reed-Solomon-Fehlerkorrektur

Teil 1

von Glenn Dixon <Dixong@iomega.com>, Roy, Utah, USA

Mit freundlicher Genehmigung der amerikanischen Forth Interest Group
übersetzt von Fred Behringer, München.
Der Originalaufsatz erschien in der Forth Dimensions, Band XX, Nummer 4,
vom November/Dezember 1998.

Fehlerkorrekturen nach Reed-Solomon sind Vorwärtskorrekturen. Sie werden in Diskettenlaufwerken, CDs, Satelliten und anderen Übertragungskanälen verwendet. Die Daten werden vor dem Absenden redundant erweitert. Am Bestimmungsort läßt sich erkennen, ob ein Fehler vorliegt, und dieser läßt sich korrigieren, ohne daß die Daten erneut übertragen werden müssen.

Stichworte: Fehlererkennung, Fehlerkorrektur, redundanter Code, endlicher Körper, irreduzibles Polynom

Zur Übersetzung: "Galois field" = "Galoisfeld", "(finite) field" = "(endlicher) Körper". Die mengentheoretischen Bestandteile eines Körpers heißen im Deutschen "Elemente". Spezielle Elemente sind "Nullelement" und "Einselement". Wir sprechen von "n-elementigen Körpern". In der Codierungstheorie ist die Bezeichnung "Codewort" oder "Wort" üblich. Die Bezeichnung "symbol" aus dem amerikanischen Original wird je nach Zusammenhang mit "Element" oder "Wort" wiedergegeben. Hier haben wir es speziell mit "Binärwörtern" zu tun: "n bit symbol" = "n-stelliges Binärwort".

In diesem Artikel wird beschrieben, wie man die Reed-Solomon-Fehlerkorrektur in Forth programmiert. Das Reed-Solomon-Verfahren ist eine Vorwärtsfehlerkorrektur und wird in Diskettenlaufwerken, CDs, Satelliten und anderen Kommunikationskanälen eingesetzt. Vorwärtskorrektur heißt, daß eine gewisse Redundanz (beispielsweise Zusatzbytes) blockweise in die Daten eingefügt wird, bevor diese losgeschickt werden. Am Bestimmungsort werden die Extradaten dazu benutzt, herauszufinden, ob Fehler aufgetreten sind, und diese nach Möglichkeit zu korrigieren. Vorwärtsfehlerkorrektur vermindert die Notwendigkeit, fehlerhafte Daten zurückschicken zu müssen. In manchen Fällen, Diskettenlaufwerke gehören dazu, stehen die ursprünglichen Daten bei Entdekkung eines Fehlers gar nicht mehr zur Verfügung und es kommt nur Vorwärtskorrektur in Frage.

Reed-Solomon-Fehlerkorrekturcodes (FKCs) arbeiten mit einer besonderen Arithmetik, die man Arithmetik der endlichen Körper oder Galoisfeld-Arithmetik nennt. Galois war ein französischer Mathematiker, der das Gebiet der endlichen Mathematik in einer einzigen Nacht um eine ganze Generation vorantrieb. Zum Duell um eine Frau herausgefordert, schrieb er in der vorausgehenden Nacht so viel er konnte über dieses Thema auf und starb am nächsten Morgen. Die Mathematiker haben oft darüber spekuliert, was er noch alles hätte entwickeln können, wenn er überlebt hätte.

Der Artikel ist in zwei Teile gefaßt. Im ersten Teil wird eine Einführung in die Arithmetik der endlichen Körper gegeben und gezeigt, wie man diese erzeugt und wie man mit ihnen umgeht. Im zweiten wird auf die Erzeugung und die Verwendung der Reed-Solomon-FKCs eingegangen.

Endliche Körper heißen so, weil sie im Gegensatz zu den rationalen oder den reellen Zahlen nur endlich viele Elemente haben. In der Codierungstheorie werden die Elemente eines endlichen Körpers Codewörter oder Wörter genannt und die heutzutage meistens verwendeten Wörter sind Binärwörter, also Zahlen zur Basis 2 oder Binärzahlen. Alle endlichen Körper mit Binärwörtern haben 2^n Elemente, wobei n eine natürliche Zahl ist. Ein 2^8-Körper hat also 256 Symbole, und das sind genau die 256 Bytes, die uns nur allzu vertraut sind. Die heutzutage üblichen Algorithmen werden für Körper größer als 2^{16} unüberschaubar und die meisten Körper haben eine Größe von 2^6-2^{12}.

In endlichen Körpern gibt es die Addition, Subtraktion, Multiplikation und Division. Also definieren wir für diese Verknüpfungen Forth-Worte:

FF+ (n1 n2 -- n3)
FF- (n1 n2 -- n3)
FF* (n1 n2 -- n3)
FF/ (n1 n2 -- n3)

(**n1, n2** und **n3** sind Elemente des betreffenden endlichen Körpers).

Das Ergebnis einer jeden einzelnen dieser Operationen liegt wieder im betrachteten endlichen Körper (wobei die Division durch 0, wie in den üblichen Zahlensystemen, nicht definiert ist). Wir können also einen endlichen Körper durch Addition, Subtraktion, Multiplikation und Division, ja sogar durch Logarithmen- oder Antilogarithmentafeln charakterisieren. Abbildung 1 zeigt Tafeln für einen 2^4-Körper. In diesem einfachen Körper lassen sich vielleicht noch Muster in den Zahlen erkennen, in größeren Körpern sieht alles wie ein einziges großes Durcheinander aus. Die Multiplikation mit 0 oder 1 liefert zwar immer noch das, was man von ihr erwartet, die Verteilung der anderen Tafelelemente ist aber alles andere als einsichtig.

Wie man sieht, gibt es keine Subtraktionstafel. In der Arith-

metik endlicher Körper sind Addition und Subtraktion ein und dieselbe Verknüpfung: XOR ! Man kann sich davon überzeugen, indem man zwei beliebige Zahlen auswählt, sie mit Hilfe der Additionstafel addiert und dann eine davon vom Ergebnis abzieht. Das funktioniert immer. (Sollte es auch, denn wir wissen, daß **n1 n2 XOR n2 XOR** uns **n1** als Ergebnis liefert: In der Sprache der endlichen Körper addieren wir **n2** zu **n1** und subtrahieren dann wieder **n2** vom Ergebnis.) Man probiere ein paar Multiplikations- und Divisionsbeispiele aus.

Kommutativität, Assoziativität und Distributivität gelten auch in der Arithmetik der endlichen Körper. Hinzu kommt eine weitere Eigenschaft: Die Subtraktion ist ebenfalls kommutativ, da sie ja nichts anderes als die Addition ist.

Ähnlich der Magischen Quadrate (Matrizen, deren Zeilen, Spalten und Diagonalen sich zur selben Zahl aufsummieren), mit denen wir als Kinder herumgespielt haben, ergeben nur ganz bestimmte Elementanordnungen einen endlichen Körper. Für einen Körper mit 4-stelligen Binärwörtern gibt es, läßt man Rotationen außer Acht, nur 6 mögliche verschiedene Anordnungen. Für einen Körper mit 8-stelligen Binärwörtern sind es derer schon 38. Je größer der Körper, desto größer die Anzahl von Anordnungen, mit denen es geht.

Wie läßt sich ein endlicher Körper erzeugen? Der Körper wird mit Hilfe von **XOR** und Verschiebeoperationen mit irreduziblen Polynomen erzeugt (die Mathematik solcher endliche Körper erzeugenden irreduziblen Polynome ist ein wesentlicher Bestandteil der FKCs). Es ist nicht ganz einfach, irreduzible Polynome aufzuspüren, sie sind aber in Tabellenform vorzufinden. In Listing 1 finden sich einige für Binärwörter von bis zu 12 Stellen. Jedem irreduziblen Polynom ist ein endlicher Körper zugeordnet. Diese Körper können noch "gedreht" werden, indem man in den Log/Antilog-Tafeln gewisse Offset-Werte vorgibt.

Das Programm in Listing 1 erzeugt, ausgehend von einem irreduziblen Polynom, einen endlichen Körper. Es liefert außerdem die drei arithmetischen Operationen (**FF-** wurde ausgelassen, man verwende **FF+**), sowie Log/Antilog und Potenzoperationen.

Listing 2 enthält ein paar Hilfsworte, mit denen Sie die Tafeln aus Abb. 1 für den von Ihnen erzeugten Körper am Bildschirm darstellen können. Für größere Körper lassen sich die Tafeln natürlich wegen ihrer Größe nicht mehr richtig ausgeben.

Zur Erzeugung eines endlichen Körpers lege man zunächst einmal in der Konstanten MASK die gewünschte Anzahl von Bits (Stellen in den Binärwörtern) fest. Dann wähle man aus der gegebenen Liste das erzeugende Polynom aus. Je nachdem, welches Polynom man wählt, ergeben sich kleinere Unterschiede in der Ausführung. Für unsere Zwecke ist jedes der angegebenen Polynome geeignet.

Sobald Sie den Code geladen haben und alles gut ging, können Sie Arithmetik auf einem endlichen Körper betreiben. Gehen Sie dabei genau so vor wie mit Ganzzahlen und verwenden Sie den Stack. Ohne spezielle Hardware erledigt man **FF*** und **FF/** am effizientesten über Nachschlagetafeln für Logarithmen und Antilogarithmen. In einem endlichen Körper ist der Logarithmus eines Binärwortes eine gewöhnliche Ganzzahl. Zur Multiplikation verwendet man also die übliche Addition (modulo der Größe des Körpers) der Logarithmen. Sehen Sie sich die Definition von **FF*** und **FF/** an.

Im nächsten Artikel werden wir mit Hilfe der Operationen auf endlichen Körpern einen Reed-Solomon-Codierer und -Decodierer aufbauen und zeigen, wie man mit dieser Form von Mathematik Fehler entdeckt und korrigiert.

Glenn Dixon ist bei der Firma Iomega beschäftigt. Er wirkte bei der Entwicklung des ZIP 100 und des ZIP 250 mit. Forth verwendet er für alles, was ihm seine Firma erlaubt. Wenn er sich nicht mit Forth beschäftigt, schreibt er Kriminalgeschichten.

Abbildung 1

POLYNOMIAL (OHNE ERSTEN TERM)= 3 MASK= F
U=Undefiniert

ANTILOGARITHMENTAFEL

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	2	4	8	3	6	C	B	5	A	7	E	F	D	9	U

LOGARITHMENTAFEL

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U	0	1	4	2	8	5	A	3	E	9	7	6	D	B	C

ADDITIONSTAFEL

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

MULTIPLIKATIONSTAFEL

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

DIVISIONSTAFEL

ZÄHLER:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
3	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
4	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
5	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
6	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
7	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
8	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A
9	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
A	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
B	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
C	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
D	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
E	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
F	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1

^NENNER

Listing 1. (Elektronische Kopien der Listings sind auf E-Mail-Anfrage beim Autor erhältlich.)

```
\ Datei FiniteField.t zur Implementation und Aus-
\ testung endlicher Körper mit GF(2^K) Binär-
\ wörtern.
\ Zur Bearbeitung von bis zu K=12, abhängig von der
\ Größe Ihres Speichers.
\ Sollte auf jedem ANSI-Forth ohne Implementations-
\ abhängigkeiten, 16 oder 32 Bit, arbeiten.

DECIMAL

\ EINE KLEINE LISTE VON IRREDUZIBLEN POLYNOMEN [1]:
\ $ bedeutet hex
\ Die untenstehenden Zahlen stellen Polynome dar.
\ Das Polynom selbst ergibt sich zu
\ x^k + x^(k-1)*Bit(k-1)+x^(k-2)*Bit(k-2)+ . . .
\ +x^0*Bit0
\ Beispiel: Das 5-stellige Polynom $1D lautet
\ x^5+x^4+x^3+x^2+1
\ 3-stellig: 3 ; 4-stellig: 3  $17  ;
\ 5-stellig: $5 $1D $17 ; 6-stellig: 3 $17 $27 ;
\ 7-stellig: 9 $F $1D   ; 8-stellig: $1D  $77  $F3
\ 9-stellig: $11 $59 $131 ; 10-stellig: 9  $F  $10D
\ 11-stellig: 5  $125  $8D
\ 12-stellig: $53  $45B  $4D

\ Den gewünschten endlichen Körper erzeugen Sie wie
\ folgt: Wählen Sie die Anzahl der Stellen für Ihr
\ Binärwort. Dann setzen Sie MASK auf das Binärwort
\ mit lauter Einsen.
\ Beispiel: MASK=$1F für ein 5-stelliges Binärwort,
\ MASK=$FF für ein 8-stelliges.
\ Dann setzen Sie POLYNOMIAL auf eine der oben
\ angegebenen Polynomzahlen.

\ HEX F DECIMAL CONSTANT MASK \ MASK=2^k-1 mit
\                               k=Binärwortlänge
\ 3 CONSTANT POLYNOMIAL       \ Aus der obenstehen-
\                   den Liste, 4-stelliges Binärwort

\ Der Körper kann auch mit einem Offset-Wert
\ versehen sein, welcher im wesentlichen eine
\ Element-Rotatation bewirkt:
0 CONSTANT M0  \ M0 ist ein Offset-Wert. In manchen
               \ Fällen wird zur wirksameren Berech-
               \ nung ein nichtverschwindender
               \ Offset-Wert verwendet. Ein nicht-
               \ verschwindender Offset-Wert dreht
               \ den Körper.

\ Zunächst stellen wir eine Antilog-Tafel her. Das
\ erreichen wir über Shift-XOR-Operationen,
\ angewandt auf das ausgewählte Polynom.

CREATE ANTILOG MASK 1+ CELLS ALLOT
                                  \ Eine Antilog-Tafel
: FILL-ANTILOG
   1 MASK 0 DO DUP I CELLS ANTILOG + !
                         \ Wert in Tafel abspeichern
       MASK U2/ MASK XOR OVER AND IF
                                  \ Oberstes Bit prüfen
         2* POLYNOMIAL XOR
                      \ Shift und XOR, falls gesetzt
       ELSE 2*
       THEN MASK AND
            \ Nächsten verschobenen Wert berechnen
       LOOP DROP ;
FILL-ANTILOG
                                     \ Tafel auffüllen

\ Durch Auslesen der Antilogarithmentafel erzeugen
\ wir uns jetzt die Logarithmentafel.

: >ANTILOG  ( n1--n2 )
\ Auslesen des Antilogs von n1 aus der Tafel
\ Anmerkung: Multiplikation und Division können
\ Eingaben erzeugen, die größer als die
\ Tafelindizes sind. Ein Aufwickeln ("Wraparound")
\ über die MOD-Operation beseitigt
\ diese Schwierigkeit.
   MASK MOD CELLS ANTILOG + @ ;

CREATE LOG MASK 1+ CELLS ALLOT

\ FILL-LOG geht einfach die Antilogarithmentafel
\ durch und setzt den jeweiligen Index in die
\ Logarithmentafel ein.

: FILL-LOG
      MASK  0 DO I >ANTILOG 1- CELLS LOG +  I
SWAP ! LOOP ;
FILL-LOG

: >LOG ( n1--n2 )
\ Bestimme Log von n1. Ergebnis ist eine Ganzzahl
\ (natürliche Zahl)
   DUP 0= ABORT" VERSUCH, LOG VON 0 IN >LOG
                 ZU FINDEN!!!"
    1- CELLS LOG + @ ;

: FF+  ( n1 n2--n )
\ Addition (und Subtraktion!) in endlichen Körpern
  XOR ;
\ Wenn Ihr System SYNONYM enthält, ist es
\ effizienter, das für FF+ zu verwenden

: FF*  ( n1 n2--n )
\ Multiplikation in endlichen Körper über
\ Logarithmen
   DUP 0<> >R
```

```
\ Wenn n1=0 oder n2=0, funktioniert die
\ Log-Methode nicht.
   OVER 0<> R> AND IF
   >LOG SWAP >LOG +
\ Zum Multiplizieren einfach Logarithmen addieren
   >ANTILOG
   ELSE 2DROP 0 THEN ;

: FF/  ( n1 n2--n1/n2 )
\ Division in endlichen Körpern über Logarithmen
   DUP 0= ABORT" VERSUCH, IN FF/ DURCH 0 ZU TEILEN!"
   OVER 0<> IF
\ 0 im Nenner erfordert Extrabehandlung.
   >LOG SWAP >LOG SWAP -
   >ANTILOG
   ELSE 2DROP 0 THEN ;

: FF^  ( n1 potenz--n2 )
\ n1 zur Potenz "potenz" erhoben:
\ potenz ist eine natürliche Zahl
   DUP 0= IF 2DROP 1 ELSE
    1 SWAP 0 DO  OVER FF* LOOP SWAP DROP THEN ;
\ mit roher Gewalt.

\ Wenn ein Offset-Wert vorgegeben wurde, müssen die
\ Log/Antilog-Tafeln jetzt überarbeitet werden:
M0 [IF]
\ Das ersetzt die Antilogtafel b^i (Offset 0) durch
\ die Tafel A^i (Offset M0)
CREATE NANTILOG MASK CELLS ALLOT
: FILL-NANTILOG
   MASK 0 DO I >ANTILOG M0 FF^  I CELLS NANTILOG + !
LOOP ;
 FILL-NANTILOG
 NANTILOG ANTILOG MASK 1+ CELLS CMOVE
\ ersetze die Antilogtafel
 FILL-LOG
  [THEN]
```

Listing 2

```
\ Hilfsprogramme zur Anzeige von Tafeln
\ endlicher Körper
\ Die Hilfsprogramme funktionieren nur bei
\ Binärwortlängen von 8 Bits oder weniger.

\ (Anmerkung des Übersetzers: Ich habe die
\ englischen Meldungen in den Ausgabe-
\ programmteilen stehen lassen, um bei der
\ Anzeige nichts durcheinanderzubringen.
\ Das erspart mir das akribische Nachprüfen
\ der Programme.)

: H.R ( n1 n2--)  \ Hex .R
  BASE @ >R HEX .R R> BASE ! ;

: .TOPLINE
   CR 4 SPACES
   MASK 1+ 0 DO I 2 H.R SPACE LOOP
   CR 4 SPACES
   MASK 1+ 0 DO ." ===" LOOP  ;

: +TABLE
   CR ." ADDITION TABLE    "
   .TOPLINE
   MASK 1+ 0 DO
    CR I 2 H.R 2 SPACES
   MASK 1+ 0 DO I J FF+ 2 H.R 1 SPACES LOOP
    LOOP CR CR ;

: *TABLE
   CR ." MULTIPLICATION TABLE    "
   .TOPLINE
   MASK 1+ 0 DO
    CR I 2 H.R 2 SPACES
   MASK 1+ 0 DO I J FF* 2 H.R 1 SPACES LOOP
    LOOP CR CR ;
```

```
: /TABLE
   CR ." DIVISION TABLE    "
   CR ." NUMERATOR:"
   .TOPLINE
   MASK 1+ 0 DO
    CR I 2 H.R 2 SPACES
   MASK 1+ 0 DO I J DUP 0<> IF FF/ 2 H.R ELSE
2DROP ."  U" THEN 1 SPACES LOOP
    LOOP
    CR ."  ^DENOMINATOR" CR ;

: ANTILOG-TABLE
     CR ." ANTILOG TABLE"
     .TOPLINE
     CR 4 SPACES
   MASK 0 DO
    I >ANTILOG 2 H.R SPACE LOOP
    ."  U " CR CR ;

: LOG-TABLE
     CR ." LOG TABLE "
     .TOPLINE
     CR ."      U "
   MASK 1+  1 DO
     I >LOG 2 H.R SPACE LOOP CR CR ;

: TABLES.
    CR  ." POLYNOMIAL (LESS TOP TERM)=" POLYNOMIAL 2
H.R
    ."  MASK=" MASK 2 H.R
    CR ." U=Undefined"
    CR CR
   ANTILOG-TABLE
   LOG-TABLE
   +TABLE
   *TABLE
   /TABLE ;
```

Patriot Scientific PSC1000
Ein Java, Forth und C Prozessor

Jens Wilke

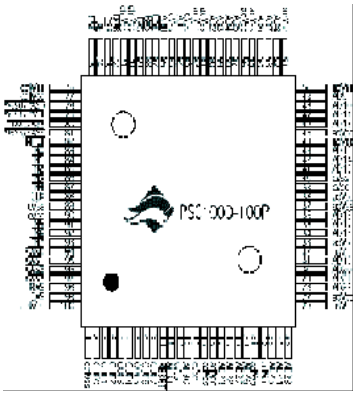
JW-Datentechnik GmbH
Brunhamstr. 21
81249 München
wilke@jwdt.com
Tel.: +49 89 897689 - 0
Fax.: +49 89 8714548

Forth-Tagung 1999 (23.-25. April), Oberammergau

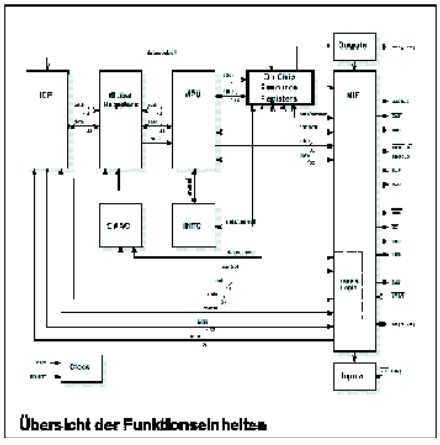
Der PSC1000 ist der jüngste und wohl auch einzige Forth-Prozessor, der über ein internationales Distributionsnetz vermarktet wird. Wohl aus Marketinggründen wird er vom Hersteller lieber als Java-Prozessor bezeichnet als als Forth-Prozessor.

Der vorliegende Text soll einen groben Überblick über den Prozessor und die Befehlssatzarchitektur geben. Außerdem wird auf das Portieren des Gforth-Systems auf den Prozessor eingegangen.
Abschließend findet man einen kurzen Performance-Vergleich zu anderen Prozessoren dieser Kategorie.

Funktionseinheiten



Eigentlich ist der PSC1000 eine zwei-Prozessor Architektur. Die MPU (Microprocessing Unit) erledigt die eigentliche (Rechen-) Arbeit, während der IOP (I/O Processor) sich - wie der Name schon sagt - nur um die Ein- und Ausgaben kümmert. Das Instruktionsformat des IOP ist ähnlich der MPU (siehe unten). An Instruktionen stehen aber nur Lade- und Speicher-, Interrupt-, DMA-Transfer-Befehle und keine Rechenbefehle zur Verfügung.



Besonderes Augenmerk beim IOP wurde auf das exakte Zeitverhalten gelegt. Um dies zu gewährleisten, hat der IOP das Oberkommando auf dem Systembus. Nur wenn sich der IOP mit der delay Instruktion schlafen legt, darf die MPU ihrer Arbeit nachkommen. Ein Speicherrefresh, der sich in normalen CPU-Architekturen als störend erweist, wenn es um exaktes Timing geht, kann dem PSC1000(-Programmierer) durch einen einfachen Trick nicht dazwischenfunken: Der Refresh wird auch vom IOP initiiert und kann damit an die richtige Stelle (unkritische) plaziert werden.

Damit ein IOP-Programm bis in den letzten Zyklus berechenbar wird, gibt es noch eine weitere Besonderheit. Alle Instruktionen und Speicherzugriffe von MPU und DMA Controller werden nur ausgeführt, wenn sie bis zum nächsten geplanten Einsatz des IOP beendet sind.
Für diesen Zweck gibt es im MIF (Memory Interface) vier konfigurierbare Speichergruppen, in denen die Zugriffszeiten exakt festgelegt und damit im voraus kalkulierbar sind. Waitstates gibt es nicht. Will man langsame Einheiten ansprechen, und reichen die vier Gruppen nicht aus, so kann/muß man das Timing einer Speichergruppe für diesen Zugriff einmalig abwandeln. Vom MIF werden 8-Bit und 32-Bit-Speicher in DRAM, SRAM, VRAM und EPROM Technologie unterstützt.

Der DMAC (DMA Controller) hat 8 Prioritätsstufen/Kanäle, die entsprechend mit dem INTC (Interrupt Controller) gekoppelt sind. An einer 1024-Byte-Grenze kann der entsprechende Interrupt ausgelöst werden, um die CPU über den Fortschritt zu informieren oder “Nachzuladen”.

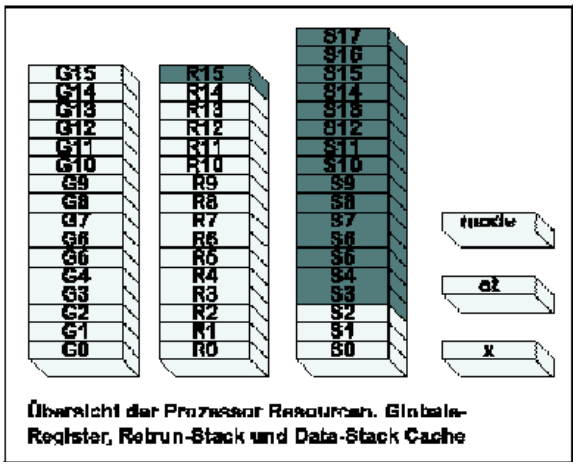
Soweit in Kürze zu den einzelnen Funktionsträgern des Prozessors. Im folgenden möchte ich auf die Architektur der MPU näher eingehen.

Ressourcen

Der Prozessor enthält 16 globale Register (G0 bis G15) die mit den Befehlen *push* und *pop* angesprochen werden. Die globalen Register teilen sich MPU und die IOP, was zum Datenaustausch zwischen den zwei Prozessoren nützlich ist. Weitere Register sind die obersten Zellen von Daten und Return Stack, die in der CPU gecached werden. Für beide Stacks sind die Befehle *sframe/rframe, scache/rcache, sdepth/rdepth* realisiert. sframe/rframe dient zum Reservieren von Platz. scache/rcache zum expliziten Rausschreiben oder Nachladen. Der Aufruf von *depth* gibt die Größe des gecachten Bereichs. Außerdem stehen Befehle für das Auslesen und setzen der Stack Pointers zur Verfügung.

Vom Datenstack werden die obersten 17 Elemente im Prozessor gehalten. Drei Speicherzellen sind allerdings nur direkt

adressierbar. Alle forth-typischen Stackoperationen sind ausführbar, z.B. *xcg (swap), pop (drop), rev (rot), push s0 (dup), push s1 (over).* Push und Pop bezieht sich immer auf den Datenstack. Also ein Push auf den Return (bzw. Locals) Stack ist ein *pop lstack*.



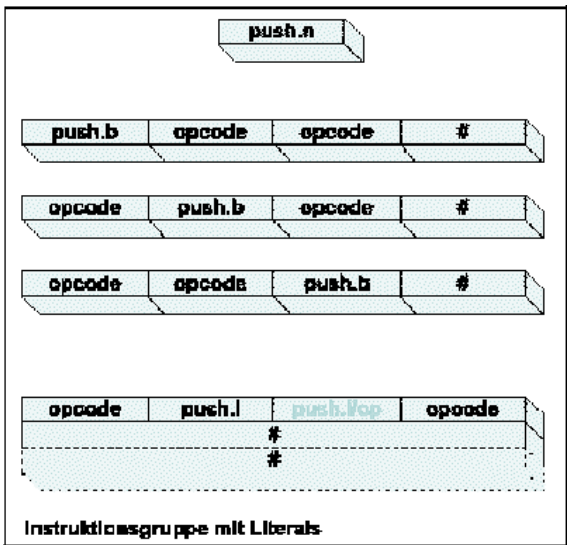
Übersicht der Prozessor Ressourcen. Globale-Register, Return-Stack und Data-Stack Cache

14 der 15 Return Stack Speicherzellen im Prozessor sind direkt ansprechbar. Mit *push* und *pop* kann schreibend und lesend darauf zugegriffen werden. Nach PSC1000 Nomenklatur handelt es sich dabei um sog. lokale Register. Damit können diese Zellen als effektive Implementierung eines C Framestack oder als lokale Variablen (für Forth) dienen. Es stehen Befehle wie *push lstack (r>), pop lstack (>r) und pop r0 (r@)* zur Verfügung. Eine Sonderrolle spielt r0. Über r0 ist ein indirektes Load und Store mit optionalem Predecrement oder Postincrement in den Hauptspreicher möglich. Über das Register x ist genauso wie über r0 in indirektes Load und Store mit optionalem Predecrement oder Postincrement in den Hauptspreicher möglich. Zusammen mit r0 lassen sich damit Kopierschleifen effektiv realisieren.

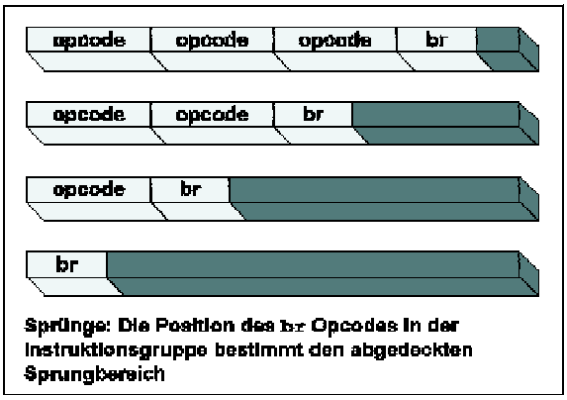
Befehlscodierung

Beim PSC1000 ist eine Instruktion in 8-Bit codiert. Vier Instruktionen werden zu einer Instruktionsgruppe in einer 32-Bit-Zelle zusammengefaßt. Eine Sonderrolle spielen Literale. Ein 4-Bit-Wert wird direkt als normale 8-Bit-Instruktion codiert. Das Datum wird sign-extended, also sind Werte von 8 bis 7 darstellbar. 8-Bit-Literale werden mit dem Opcode push. b codiert. Taucht ein push.b in einer Instruktionsgruppe auf, so wird das letzte Byte der Gruppe als 8-Bit-Wert (nicht sign-extended) auf den Daten-Stack gelegt. Der Opcode push.l legt das der Instruktionsgruppe folgende 32-Bit-Wort auf dem Datenstack ab. push.l kann in einer Instruktionsgruppe mehrmals und an beliebiger Stelle vorkommen.

Einer weiteren Spezialbehandlung bedarf es bei den Sprüngen. Wird ein Sprungopcode erkannt, sind die restlichen Daten der Instruktionsgruppe (der noch nicht abgearbeitete Teil), zusammen mit den 3 niederwertigsten Bits der Sprunginstruktion selbst, Sprungoffset. Der Offset wird dabei nur in Zellen relativ zur Instruktionsgruppe angegeben. Es kann also nur auf Zell-Adressen (durch vier teilbar) gesprungen werden. Auf diese Weise codiert, stehen 4 Sprungarten zur Verfügung: br, call, dbr (decrement ct und branch not zero) und bz (branch if tos zero, ?branch in Forth).



Instruktionsgruppe mit Literals



Sprünge: Die Position des br Opcodes in der Instruktionsgruppe bestimmt den abgedeckten Sprungbereich

Befehle

Für Integerarithmetik stehen die üblichen Befehle bereit. *Add* und *sub* können, wie bei normalen Prozessoren auch, über das
Carry-Flag kaskadiert werden. Dazu stehen sie in unterschiedlichen Varianten bereit. Außerdem gibt es Befehle für increment und decrement um 1 oder um 4. Für Logik gibt es neben *and, or* und *xor* ein *iand* (invert and) und obendrein noch eine große Auswahl an Shift-Befehlen. Es kann entweder bitweise (eine Zelle oder Double) oder byteweise (nur eine Zelle) nach links oder rechts geschoben werden. Shift-Left füllt immer mit Null auf, Shift-Right mit Carry. Die Instruktionen *shift* und *shiftd* schieben um eine variable Anzahl von Bits. Übergibt man einen negativen Wert wird nach rechts geschoben. Die Verschiebung wird optimiert erst byteweise, also 8-Bit in einem Taktzyklus, und dann bitweise durchgeführt.

Die Multiplikation gibt es in einer signed und unsigned, die Division nur in einer unsigned Variante. Multiplikation und Division werden (unparallelisiert) in 32 Takten ausgeführt.

Floating Point Arithmetik wird nicht direkt unterstützt. Stattdessen sind notwendige Grundkomponenten im Prozessor integriert, damit man sich eine effektive Floating-Point Arithmetik zusammenstellen kann.

Für den Speicherzugriff gibt es die Befehle *ld[] (@), ld.b[] (c@), st[] (!)* und *replb*. Das Forth c! muß man sich über replb, der ein Ersetzen eines Bytes in einer Zelle erlaubt, und fetch und store zusammenbauen. Praktisch sind die Befehle, die sich auf die Abarbeitung einer Instruktionsgruppe beziehen. *mloop* wiederholt die aktuelle Instruktionsgruppe, *skip* überspringt den Rest und fährt mit der nächsten Instruktionsgruppe fort. skip und mloop können unconditional oder mit den Bedingungen *c* (Carry), *nc* (not Carry), *p* (tos positive), *n* (tos negative), *z* (tos zero) und *nz* (tos not zero) verknüpft sein.
Somit lassen sich Kopierschleifen innerhalb eines Instruktionswortes realisieren.

Gforth-Port, der Cross-Compiler

Mit der Arbeit am Gforth für den PSC1000 bin ich wieder um eine entscheidende Erfahrung reicher: Die Portierung eines Forth-Systems auf einen Forth-Prozessor ist wesentlich arbeitsintensiver als auf einem normalen, register-orientierten Prozessor. Warum dies so sein soll, ist zwar nicht sofort einzusehen, liegt aber auf der Hand: Um ein gefädeltes Forth-System auf einen neuen Prozessor anzupassen, braucht man lediglich next und ein paar Befehle in der Maschinensprache der Ziel-CPU zu codieren. Für den Cross-Compiler, der den Fädelcode erzeugt, bleibt alles fast gleich: Befehle, Sprünge, Literale. Bei einem Forth-System für einen Forth-Prozessor muß man dies alles stets neu anpassen.

Um dies einfach zu gestalten und den Cross-Compiler weitestgehend für alle Implementationsvarianten (also Fädelcode, Assembler und Forth/Stack-CPUs) zu nutzen wurde eine Plug-In Schnittstelle eingeführt. Somit kann der Standardcode für das Compilieren von Colon-Calls, Sprünge, Literals usw. CPU-spezifisch ersetzt werden. Praktisch beim Portieren des
Gforth-Systems ist, daß nahezu alle benötigten Wörter auch als entsprechende high-level Forth Definitionen zur Verfügung stehen. Ein spezieller Modus im Cross-Compiler erlaubt es, nur die Wörter hinzuzufügen, die noch nicht definiert sind. So ist es möglich, sich an ein optimales Forth-System heranzutasten. Für ein lauffähiges System kann man die komplizierten Befehle wie IF, DO, /, usw. erstmal einfach weglassen.

Das Grundkonzept

Da die primitiven Forth-Befehle direkt vom Prozessor unterstützt werden gibt es keinen expliziten Assembler Modus. Forth-Befehle und CPU-Instruktionen können direkt gemischt werden. Um Namenskonflikten vorzubeugen, haben die CPU-Instruktionen ein Komma hintenangestellt.

Ist ein CPU-Befehl äquivalent zu einem Forth-Befehl, so ist er unter diesem geführt. Die CPU-Befehle werden als ganz

adda	+
sub	-
eqz	0=
call[]	execute
neg	negate
pop	drop
pop lstack	>r
push	dup
push s1	over
push lstack	r>
pev	rot
shl #1	2*
swap	xcg
or	or
xor	xor
and	and
inc #1	1+
inc #4	call+
ld[]	@
ld.b[]	c@

In Forth abbildbare PSC1000 Befehle

normale Wörter geführt, die als Code nur den jeweiligen Befehl enthalten. Damit hat so ein Befehl auch ein gültiges Execution Token, das für Execute und Defer benötigt wird. Dafür, daß beim Compilieren eines CPU-Befehls nicht ein Call sondern der Befehl codiert wird, ist die Inline Funktion zuständig. Kurze Instruktionssequenzen und damit auch einzelne CPU-Befehle, werden direkt in das neue Wort übernommen.

Gemeinheiten

Zeit kostet beim Portieren kleine Feinheiten. So ist die PSC1000 Instruktion st[] eigentlich das Store (!) von Forth, aber halt nur eigentlich. Um das von in Forth bekannte Verhalten zu erzielen, bedarf es noch einem zusätzlichen *drop*.

Ein anderes, größeres Problem ist die Implementation von *pick*. Wie man es auch dreht, eine effektive gibt es nicht. Die Problematik ist der Stack-Cache im Prozessor, den man nur zum Teil und schon gar nicht indiziert direkt adressieren kann. Im wesentlichen gibt es zwei Varianten um dem Problem zu begegnen. Entweder man spillt den Cache in den Speicher und greift dann indirekt über den Stackpointer zu (wie man es kennt), oder man schaufelt die Stack-Elemente z. B. auf den Return-Stack und wieder zurück, um an die gewünschte Zelle heranzukommen.

Durch diesen Umstand sind Forth Programme auf dem PSC1000 weniger effektiv als C Programme. Während Forth Programme den Stack reorganisieren müssen arbeiten C Programme mit lokalen Variablen auf dem Return-Stack Frame.

Erfreuliches

Praktisch lassen sich die Doer codieren. Doer nennen wir in Gforth die kleinen Programme die das Verhalten von Variablen, Konstanten, User-Variablen, usw. bestimmen. Im Normalfall steht bei einer Variable zum Beispiel an der xt-Adresse (die CFA) eine Adresse (indirekt gefädelt) oder ein Call (direkt gefädelt) einer Routine, die die body-Adresse (die PFA, in Gforth CFA+8) zurückgibt.

```
nonisline versions:
: (docon   r> cell+ >r ; isdoer
: (dovar   r> cell+ @ >r ; isdoer
: (douser  r> cell+ @ up@ + >r ; isdoer

inline versions:
: (docon _inline 7 addpc, >r ; isdoer
: (dovar _inline 7 addpc, @ >r ; isdoer
: (douser _inline 7 add_pc, @ up@ + >r ; isdoer
```

Die doer lassen sich in den inline Versionen direkt in das Code-Field "quetschen"

Dieses Stück Code läßt sich beim PSC1000 locker in die zwei Zellen also 8 Byte bzw. Instruktionen quetschen, die im Code-Field Platz sind. Möglich macht dies die Instruktion *addpc*, eine Addition die als eine Operanden den aktuellen PC-Inhalt hat. Das Literal sieben läßt sich ebenfalls in einem Ein-Byte Opcode codieren. Eine weitere Variante, z.B. für *docon* ist der Einsatz von *skip* und *push.l*.

Performancevergleich

Im Kasten sieht man das typische Beispiel von Patriot. Bei gleichem Effekt ergibt sich gegenüber einer Registermaschine nur die Hälfte an Codegröße in Bytes. Was aber auch deutlich sichtbar ist, ist das doppelt soviele Instruktionen und damit auch Taktzyklen notwendig sind. Außerdem wurden hier bewußt Rechenausdrücke gewählt (x+1, x*2), die sich im PSC1000 in einer Instruktion abhandeln lassen. Das Beispiel repräsentiert außerdem natürlich kein reales Programm. Auch sind Ausdrücke in der Länge in normalen Programmen eher selten, so daß der Vorteil des Stack zum effektiven Ausrechnen von Ausdrücken sich nicht voll entfalten kann.

g5 = g1 - (g2 +1) + g3 - (g4 * 2)

	push g1
	push g2
add g2, #1, g5	inc #1
sub g5, g1, g5	sub
	push g3
add g5, g3, g5	add
	push g4
shl g4, #1, temp	shl #1
	sub
sub temp, g5, g5	pop g5
20 Bytes	10 Bytes

Vergleich von (3-adress) RISC-Befehlen und den "zero operand" (Stack) Befehlen des PSC1000 (Quelle: Patriot Scientific)

Pro echte Operation sind damit zwei bis drei Operationen Overhead nötig, um Operanden zu holen oder Ergebnisse zu speichern. Dies spiegelt sich in der Dhrystone MIPS Gegenüberstellung wieder. Etwa ein Drittel der Taktfrequenz holt der PSC1000 an MIPS heraus.

- **ARM7**
  - 30 Mips @ 33 Mhz = 0,91 Mips / Mhz
  - 55000 Transistors, 4 Kbyte Cache
- **Motorola Coldfire**
  - 13,5 Mips @ 33 Mhz = 0,41 Mips / Mhz
- **Intel 386**
  - 11,5 Mips @ 40 Mhz = 0,29 Mips / Mhz
- **Intel 486**
  - 18 Mips @ 33 Mhz = 0,55 Mips / Mhz
- **PSC1000**
  - 28 Mips @ 100 Mhz = 0,28 Mips / Mhz

Gegenüberstellung der Dhrystone MIPS

Vergleicht man allerdings mit einem nackten i386 (kein DMAC, kein IOP,...) ist bei seiner Komplexität (137,500 Transistoren) die Leistung schon beachtlich.

Fazit

Auch mit dem PSC1000 als moderner Prozessor, werden Stack-CPUs in der Nische bleiben. Erweiterte Konzepte wie Pipelining, Parallelisierung und on-chip Cache fehlen. Durch seine straffe Architektur hat der PSC1000 aber auch Vorteile: Schnelle, auf den Taktzyklus genaue Reaktion machen den PSC1000 in dieser Leistungsklasse einmalig. Eine andere Nische für den PSC1000 ist die Integration auf ASICs. Gegenüber Standard-CPUs macht sich seine geringe Komplexität hier positiv bemerkbar.

Sucht man lediglich einen Allerweltsprozessor, hat man beim PSC1000 lediglich den Vorteil daß er Forth schnell ausführen kann. Komplexere, Main-Stream CPUs haben zum gleichen Preis schon Cache integriert (und diverse Schnittstellen) und ermöglichen so die Verwendung von langsameren Hauptspeichern.

Infos/Bezugsquelle

Weitere Informationen zum PSC1000, und auch das Prozessor Manual, findet man auf der WEB-Site der Patriot Scientific Cooperation unter www.ptsc.com. In Deutschland ist der PSC1000 über die Firma Ineltek, Heidenheim (07321/9385-0) zu beziehen. Die lokalen Vetriebsbüros findet man unter www.ineltek.com. ***Jens Wilke***

Wollen Sie interessante Vorträge 'life'
hören, sehen und erleben ?

Die nächste
Jahrestagung der Forthgesellschaft e.V.
findet
vom **14. bis 16.04.2000** in
Hamburg statt !

BEGIN--UNTIL über 32 K ab 80386 im ZF-Assembler

von Fred Behringer <behringe@mathematik.tu-muenchen.de>
Planegger Str. 24, 81241 München

Sprünge in den Kontrollstrukturen konnten bisher nur 128 Byte weit gehen. Mit einem kleinen Handgriff läßt sich dieses Manko nachträglich, auch im laufenden Betrieb, beseitigen.

Stichworte: ZF, F83, Turbo-Forth, BEGIN--UNTIL, BEGIN--AGAIN, BEGIN--WHILE--REPEAT, IF--THEN; IF--ELSE--THEN.

Strukturiertes Assemblieren ist eine feine Sache. Leider gehen die zulässigen Sprünge in ZF, Turbo-Forth und den anderen F83-Derivaten nur bis +127 oder -128. Es werden die bedingten Sprungbefehle, die 7xh als Opcode haben, eingesetzt. Andere gab es bis zum 80286 nicht. Und die haben nur 8-Bit-Operanden.

(FIND) komfortabler, "genialer" oder mit Zusatztricks auszustatten, führt beispielsweise leicht zur Überschreitung dieser Grenze. Der ZF-Assembler hat nicht einmal eine Fehlerwarnung ("Sprung zu weit") eingebaut. Das Operandenbyte wickelt sich einfach einmal um sich selbst auf (wrap around). Einfach so.

Natürlich gibt es Tricks (siehe Assembler-Bücher). Wenn ich aber in einer CODE-Definition einen bedingten Sprung über eine Länge von 2 Kilobyte haben möchte - und was ist das schon? - dann möchte ich nicht an 20 verschiedenen Stellen, deren genaue Lage ich nur im Trial-and-Error-Verfahren ermitteln kann, Hilfs-**JMP**s einbauen müssen.

Zum Glück geht es einfacher. Es sagt einem nur niemand. Und man sage nicht, das stehe dort und dort! Da, wo es hingehört, steht es jedenfalls nicht. Und es geht sogar ganz leicht. **?>MARK** , **?>RESOLVE** und **?<RESOLVE** müssen per **IS** neu eingebunden werden, und flugs laufen dann alle Assembler-Kontrollstruktursprünge über Strecken von bis zu ungefähr 32 Kilobyte nach oben und unten. Die Neueinbindung kann sogar im laufenden Betrieb vorgenommen werden.

Ab dem 80386 können bedingte Sprünge (auch im Real-Mode) bis zu etwa 32 Kilobyte (vorwärts und rückwärts) lang sein (bei den üblichen 64K-Segmenten - im Protected-Mode geht es noch wesentlich weiter).

Mit 0F beginnen die bedingten Sprungbefehle in der Version mit 16-Bit-Operanden und in den Opcodes haben sie 8x statt 7x. Außerdem muß bei der Sprungweite noch ein Byte zugelegt werden. Deren Berechnung erfolgt ja (in weiser Voraussicht oder schludrigerweise?) sowieso schon in (der üblichen) 16-Bit-Arithmetik, wobei dann per C, nur immer die untere Hälfte übernommen wird. Nun wird stattdessen einfach "," eingesetzt. Das läßt sich aber alles, auch in einem schon laufenden System, leicht "nachträglich" ändern.

Ein klein bißchen verkompliziert werden die Überlegungen noch dadurch, daß bei **ELSE** und **AGAIN** bisher der unbedingte relative Sprungbefehl **JMP** mit dem Opcode **EB** verwendet wurde. Das kann aber mit einer vorherigen Abfrage in **IF** und **UNTIL** und bei Bedarf Ersetzen durch den 16-bittigen **JMP**-Befehl **E9** aufgefangen werden.

Getestet habe ich die oben angeführten Strukturen in ZF mit ähnlichen Testprogrammen wie dem untenstehenden für **BEGIN--UNTIL** . Überlegung: Fällt der Sprung, falls das Ganze nicht funktionieren sollte, um auch nur ein einziges Byte kürzer aus als geplant, so daß das **AX INC** am Anfang nicht mehr mit erfaßt wird, so gerät das Testprogramm in eine Endlosschleife.

Patchen wird man das, was ich hier tue, nicht nennen wollen. Aber warum eigentlich nicht? Ich lade den Leser dazu ein, in ein System lokal einzugreifen, ohne ihm die Gesamtübersicht oder auch nur einen begrenzten Teil von tieferem Einblick abzuverlangen. Ein Gen wird verändert, und aus der Tomate wird eine Riesentomate. Der Bauplan bleibt weiterhin ein Geheimnis des Allmächtigen. Daß man das auch in Forth so ohne weiteres kann, liegt an dessen ins Unendlichkleine gehende Modularisierung. Das Patchen vollzieht sich bei meinem Vorgehen auf dem Stack und im Dictionary, beim Übergang von Wort zu Wort. Da mogle ich mich frecherweise hinein, ohne bemerkt zu werden. Und was das eigentliche Patchen betrifft, so sollte man mit dessen Ablehnung vorsichtig sein. Wir teilen unsere Welt in Begriffe ein, um sie besser begreifen zu können. Wir bauen uns eine Begriffswelt auf. Die Einteilung selbst bleibt uns überlassen. Alles ist eine Frage der Betrachtungsweise. Man ordne dem Patchen ein geeignet wohldefiniertes Bedeutungsfeld zu, und schon wird es hoffähig. Hier ein Versuch: Patchen ist ein Eingriff in ein System mit dem Ziel, es wesentlich umzugestalten, ohne es neu erzeugen zu müssen, geschweige denn zu können. Ärzte und Genetiker sind die größten Patcher vor dem Herrn. Forthler kommen gleich danach. Natürlich hätte ich die Sprungmechanismen bei einer Neumetacompilation (Übung für die Fleißigen) eleganter (direkter und damit wesentlich kürzer) konstruiert. Man kennt ja (gedankt sei der offenen Informationspolitik der internationalen Forth-Gemeinde) den Bauplan.

Programm-Listing ===>

```
HEX  ASSEMBLER DEFINITIONS
```

BEGIN--UNTIL über 32 K ab 80386

```
: A?>MARK-386
   [ FORTH ]          \ AND und OR aus FORTH
   HERE 1- C@
   DUP 0EB =          \ JMP mit 8-Bit ?
   IF
     DROP 0E9 HERE 1- C!
   ELSE
     0F HERE 1- C!    \ 16-Bit-Opcode
     0F AND 80 OR C, \ Opcode 8x statt 7x
   THEN
   TRUE HERE 0 ,      \ 2 Bytes reservieren
   [ ASSEMBLER ] ;    \ Eigentlicher CONTEXT

: A?<RESOLVE-386
   [ FORTH ]
   HERE 1- C@
   DUP 0EB =
   IF
     DROP 0E9 HERE 1- C!
   ELSE
     0F HERE 1- C!
     0F AND 80 OR C,
   THEN
   HERE 2+ - ,
   [ ASSEMBLER ]
   ?CONDITION ;

: A?>RESOLVE-386
    HERE OVER 2+         \ Länge plus 1, da 16
Bit
   - SWAP ! ?CONDITION ;

' A?>MARK-386    IS ?>MARK
' A?<RESOLVE-386 IS ?<RESOLVE
' A?>RESOLVE-386 IS ?>RESOLVE

FORTH DEFINITIONS

CODE TEST
   AX POP
   BEGIN  AX INC
   NOP  NOP  NOP  NOP  NOP  NOP  NOP  NOP  NOP  NOP
NOP
   ..................... insgesamt 512 NOPs
   NOP  NOP  NOP  NOP  NOP  NOP  NOP  NOP  NOP  NOP
NOP
   AX AX OR  0= UNTIL  NEXT END-CODE
```

Lieber Friederich und lieber Thomas,

meinen persöhnlichen Dank an Euch beide für die außerordentliche Gefälligkeit die Ihr "Henry's Berichten" hinzufügt. Erstens habt Ihr mir die VD Nr.3/99 mit Luftpost geschickt, so daß ich sie zum SVFIG-Treffen am nächsten Tag mitnehmen konnte; weiterhin sehe ich (auf Seite 34), daß sogar Chris Jakeman in England feststellt, daß es für Thomas nicht leicht sein kann, meinen kalifonischen Slang zu übersetzen. Ich bin überwältigt.
Aber dann...Ich weiß nicht, wie ich "Voll das Verarschungsprogramm, ey!" übersetzen soll!
Leser mit 'Amerikaerfahrung' - wer hilft hier weiter ?

red

Ich stimme Fred Behringer zu, wenn er sagt: "Nichts ist für einen Autor schlimmer als keine Reaktion." Danke für die Reaktionen in der VD und, bei dieser Gelegenheit, laßt mich ein paar von meinen zurückgeben.

1. An Georg Beierlein (ist er ein weiterer Professor aus Mittweida?): Meine Antwort auf seine Knobelaufgabe (auf Seite 34) geht in etwa so --
Beginnen wir mit allen drei Lichtschaltern in der AUS Position. Schalte Nr.1 an, dann trinke langsam das Bier, mit Genuß; schalte Nr.2 an; schnapp Dir die leere Bierbüchse und renne schnell die Treppe hoch (hoffentlich hat das Bier noch nicht seine Wirkung entfaltet und Du kommst sicher hoch); benütze die Bierbüchse (Oberseite nach unten), wenn nötig, um daraufzusteigen und die Glühlampen zu erreichen; fühle die Umgebung der beiden leuchtenden Lampen ab -- die wärmere wurde vom Schalter 1 angeschaltet, Schalter 2 schaltete die andere an; die, die nicht an ist, gehört zu Schalter 3.
Ich vergaß zu sagen, daß man ein Stück Papier und einen Bleistift mitbringen und das Ergebnis aufschreiben sollte, denn das Bier mag Dich inzwischen erwischt haben, vielleicht fällst Du um, schläfst ein und vergißt die Lösung.

(Zitat Georg Beierlein: "Prima! Genau richtig gelöst! Gratuliere!")

2. An Fred Behringer, über ZF:
ZF Forth ist wahrscheinlich der Hauptgrund, aus dem wir "Henry's Berichte" heute überhaupt haben. Es ist eine lange Geschichte. Mein erster Brief an Friederich, am 29. Juni 1995, war eine Anfrage nach seinem "wirklich schönem Anfänger-Päckchen", welches auf Seite 18 der VD 1/95 beschrieben war. Ich war schon immer ein "Anfänger" in Forth, auf der Suche nach einem Forth-Paket mit dem ich lernen konnte. Friederich sandte es netterweise zu mir und dann erkannte ich, daß ZF Tom Zimmers Vorgänger von FPC war. Ich zeigte es Tom auf dem nächsten SVFIG Treffen. Sein Kommentar war, daß er sich nicht mehr erinnere, was ZF enthielt, nur das es klein war. Wie auch immer, es machte einen vollen Kreis, von den USA nach Deutschland und zurück zu mir.
Ich begann es zu erlernen und gab es sogar an andere weiter, doch da Friederich seinen Kurs nicht fertig geschrieben hatte, stoppte mein Lernprojekt. Stattdessen begannen Friederich und ich öfter miteinander zu korrespondieren, bis er eines Tages einen meiner Briefe an Claus Vogt sandte, den dieser in der VD 3/96 abdruckte. Die Berichte "über den Großen Teich" sind seitdem regelmäßig erschienen.

3. An Martin Bitter:
Ich fand Deinen Artikel über den Lee-Effekt sehr interessant. Ich mache gerade ein paar Bemerkungen darüber zu meiner Stieftochter, die Sprachtherapeutin ist und mit Kindern arbeitet.

4. An 'werimmer' die Bilder für die Zeitung machte:
Ich würde mich freuen die Namen all der Leute auf dem Bild zu erfahren. Dieses mal, glaube ich, habe ich Klaus Schleisiek in der hinteren Reihe gefunden.

Fortsetzung: Seite 29

Hashing

- Teil 1 -

von Friederich Prinz <Friederich.Prinz@t-online.de>
Hombergerstraße 335, 47443 Moers

Mit dem Hashing verhält es sich ein wenig wie mit Vergasern und Ventiltrieben in einem Auto. Die meisten Benutzer (eines Autos oder eines Forthsystems) wissen, ‘daß es so etwas gibt’. Einige wenige Nutzer kennen die jeweiligen Prinzipien, aber kaum jemand macht sich die Mühe, diesen Prinzipien wirklich auf den Grund zu gehen. Wozu auch ? Immerhin lehrt die Erfahrung, daß diese elementaren Grundmechanismen auch ohne tiefere Kenntnisse des Anwenders funktionieren.

Und doch gibt es immer wieder Zeitgenossen, denen an ‘Aufklärung’ gelegen ist, und die mit ihren Fragen dazu verführen, sich Aufgaben zu stellen, die man zuvor, wohlwissend um den damit verbundenen Arbeitsaufwand, abgelehnt hatte.

Stichworte: ZF, Assembler, BXDEBUG, Hashing

Der nachfolgende Aufsatz ist im Frühjahr 1994 entstanden als Zusammenfassung einer Reihe von E-Mails und Beiträgen im Z-Netz, als Antwort auf eine Anfrage zum Hashing. Seine Veröffentlichung war mehrfach vorgesehen, ist aber bisher wegen des Umfanges stets zurückgestellt worden.
Aufgrund mehrerer Anfragen, die mich zum Thema Hashing in den letzten Monaten erreicht haben, beschreibe ich nachfolgend in zwei bis drei Fortsetzungen, wie ich dieses Thema für mich selbst (und andere) aufgearbeitet habe.

fep

Vorwort

Daten ´ungeordnet´ ablegen und trotzdem schnell und einfach wiederfinden zu können, das soll, zumindest in Theorie, das HASHING ermöglichen. Hashing ist nichts anderes, als das Berechnen der Adresse eines Speicherplatzes, an dem abzulegende und/oder aufzugreifende Daten stehen sollen. In den verschiedensten FORTH-Systemen ist HASHING ein fester Bestandteil der Suchläufe in den Wörterbüchern. Wie das HASHING aber jeweils im Detail definiert und organisiert ist, ist nicht so einfach nachzuvollziehen. Das HASHING eines bestehenden FORTH-Systems ‘auseinander zu nehmen’ und im Detail zu betrachten, setzt verhältnismäßig tiefgreifende Kenntnisse des jeweiligen Systems ebenso voraus, wie sehr viel Geduld, die Bereitschaft zu umfangreichem Arbeiten und den ´Mut zu Fehlern´.

Im Zusammenhang mit einem im Herbst 1991 in Moers durchgeführten Kurs zur Programmierung von Dateiverwaltungen unter ZF, hatte ich mich bereits ‘im Ansatz’ mit Hashing beschäftigt. Mir wurden damals die Voraussetzungen sehr schnell klar. Vor allem die Aussicht auf die immense, zu leistende Arbeit bewog mich seinerzeit, das Thema Hashing bis auf weiteres wieder ‘zu den Akten’ zu legen. Ende April ’94 erreichte (oder ereilte) mich aber über das Netzwerk eine Nachricht, die Johannes Teich aus Murnau öffentlich im Z-NETZ ausgeschrieben hatte. Die Frage nach dem HASHING tauchte erneut auf. Diesmal lies ich mich auf das Thema ein, nicht ahnend, daß es mich einen ganzen Monat kosten würde.

Wie Alles angefangen hat

Am 30. April 1994 erreichte mich über die ´Heimatbox´ der Moerser Forthgruppe eine Nachricht, die Johannes Teich in das Z-NETZ gegeben hatte:

```
Empfaenger  : /Z-NETZ/SPRACHEN/FORTH
Absender    : J.TEICH@CBRA.zer.sub.org
              (Johannes Teich)
Betreff     : Hashing
Datum       : Do 28.04.94

Um ein großes Dictionary schneller zu verwalten, muß
man sich von der linearen Sucherei trennen. 'Hashing'
ist das Stichwort.

Hans Franke erwähnte mal, er bevorzuge 'fortgesetzte Teilung'. (Hallo,
Hans?) Nun ist das ja recht schön - mit maximal 12 Schritten kann ich
aus 4096 sortierten Elementen das gesuchte finden - aber wer sortiert mir
die Sachen? Ich müßte ja für jeden Eintrag das Dictionary neu ordnen.
(Wo ist da der Witz 'bei - bin doch sonst nich so humorlos... :)
Ich hänge einen (älteren) Text von Ray Duncan an. Er verrät einige inter-
essante Details, wie sie im UR/Forth Anwendung finden. Aber ich würde
auch gern mal konkreten Code sehen...
Ich habe mich bislang vor dem Hashing gedrückt, und nun stelle ich fest,
daß ich es nicht mit einem Klax erledigen kann. Wie optimiert man so-
was? Empirisch? Irgendwelche Tips? Publikationen?

                                        cu --Hannes
```

Den anhängenden, englischen Text von RAY DUNCAN “unterschlage” ich an dieser Stelle. Ray Duncan sagt darin von seinem UR/FORTH LMI, daß Suchläufe aufgrund eines aus-geklügelten Hashalgorithmus’ grundsätzlich mit einem einzigen Stringvergleich auskommen.

Humorlos ist der Hannes wirklich nicht. ‘Konkreten Code’ würde ich auch gerne einmal sehen. ‘Gedrückt’ hatte ich mich, wie eingangs beschrieben, bisher auch vor dem HASHING. Und wie man das Thema generell zu fassen bekommen sollte, davon hatte ich zu diesem Zeitpunkt keine Vorstellung. ‘Publikationen’, nach denen der Hannes gefragt hatte, standen mir auch nicht zur Verfügung. Also blieb, wie meistens in solch aussichtslosen Fällen, nur der Griff zum “DUDEN Informatik”. Dort steht, in der Auflage von 1989, unter dem Stichwort HASHING:

Stichwort: Hashing

Das Hash-Verfahren ist ein Speicherungs- und Suchverfahren, bei dem die Adressen von Datensätzen aus den zugehörigen Schlüsseln errechnet werden. Das Verfahren eignet sich auch für die Speicherung auf Massenspeichern und ist dem B-Baum dann vorzuziehen, wenn

die Daten nur eingefügt, aber nicht gelöscht werden. Formal definiert man eine Hash-Funktion **h:k-->A** von der Menge der Schlüssel K in die Menge der Schlüssel A. Ziel ist es, h durch möglichst einfache arithmetische Operationen zu realisieren. Die Suche nach einem Datensatz mit dem Schlüssel k beschränkt sich dann auf die Berechnung von h(k).

Abb.1: Suche nach einem Datensatz mit dem Schlüssel k

Beispiel: Sei K = (Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag),

A=(1,2,...,22).

f: (A,B,...,Z) --> (1,2,...,26)

ist eine in ähnlicher Form in vielen Programmiersprachen vorkommende Hilfsfunktion, die jedem Buchstaben seine Position im Alphabet zuordnet.

h: K --> A sei definiert durch

h(k) = f("1.Buchstabe von k") + f("2.Buchstabe von k") - 12

Es ist h(MONTAG) =16 h(DIENSTAG) = 1
h(MITTWOCH) =10
h(DONNERSTAG) = 7 h(FREITAG) =12
h(SAMSTAG) = 8
h(SONNTAG) =22.

Einfügung - Steht NICHT im DUDEN !!!

Soweit, so gut (oder auch nicht). Wenn man's ein paar Mal liest, und sich ein wenig Zeit nimmt, kann man das sogar verstehen. An dieser Stelle war ich fast soweit, den DUDEN einmal mehr 'in die Ecke zu pfeffern' und von meinem Schreibtisch zu verbannen. Als Techniker sehe ich die Notwendigkeit eindeutiger Terminologien und formalistischer Beschreibungen durchaus ein. Das muß doch aber nicht immer dazu führen, daß man alles dreimal lesen muß. Ich bin doch auch so bereit zu glauben, daß die Leute, die so etwas schreiben, auch selbst verstanden haben, was sie schreiben. Zumindest für mich selbst erkläre ich den ganzen Sermon noch einmal, gewissermaßen 'für Doofe'.

Also: Wir ordnen jedem Buchstaben des Alphabets eine Zahl zu. Das 'A' wird zur '1', das 'B' zur '2' usw... Dann nehmen wir von den zu verschlüsselnden Worten jeweils die ersten beiden Buchstaben, z.B. das '**MO**' von '**MO**ntag'.
Die Zahlenwerte der beiden Buchstaben addieren wir. Von der Summe ziehen wir dann 12 ab. Das Ganze sieht dann so aus:

M + O
13 + 15 -12 = 16

Jetzt geht's im DUDEN weiter...

Einfaches Nachrechnen zeigt, daß die Anwendung von h auf die englischsprachigen Wochentage zu einer Kollision geführt hätte, denn es ist

h(MONDAY) = h(WEDNESDAY)=16.

Von einer Kollision spricht man, wenn beim Einfügen eines neuen Schlüssels k' einer der beiden folgenden Fälle auftritt:

- An der Adresse h(k') steht ein Schlüssel k, der auch an diese Adresse gehört, d.h. für den h(k') = h(k) gilt (sog. Primärkollision).

- An der Adresse h(k') steht ein Schlüssel k, der bzgl. der Hashfunktion h nicht hierhin gehört, für den also h(k') ungleich h(k) gilt (sog. Sekundärkollision). k muß durch eine Kollisionsstrategie (s.u) an dieser Stelle eingetragen worden sein.

Das obige Beispiel bezieht sich auf einen festen Datenbestand und eine feste Schlüsselmenge und zeigt eine Hash-Funktion h, die auf der Schlüsselmenge K keine Kollisionen hat. Für große Schlüsselmengen ist dies nicht die Regel. Vielmehr führt bei der Anwendung des Hash-Verfahrens auf dynamische Dateien die Hinzunahme von neuen Schlüsseln oftmals zu Kollisionen. Um nicht ständig neue Hash-Funktionen h finden zu müssen, werden Strategien zur Behandlung von Kollisionen eingesetzt.

Der Entwurf einer Hash-Funktion und der zugrundeliegenden Datenstruktur gliedert sich daher in zwei Teilprobleme:

1) Bestimmung einer Hash-Funktion h, die einfach zu berechnen ist und die die Menge des Schlüssels K möglichst gleichmäßig und zufällig auf die Menge der Adressen A abbildet (um Kollisionen im Schnitt gering zu halten).

2) Festlegung einer Strategie zur Behandlung von Kollisionen.

Zu 1) Ein einfacher und guter Ansatz zur Bestimmung einer Hash-Funktion für A=(0,1,...,p-1) ist h(k) k mod p.

Falls k nicht numerisch ist, muß k geeignet umgewandelt werden. p ist eine beliebige, natürliche Zahl, die die maximale Größe des Speicherbereichs angibt. (Falls man die quadratische Verschiebung (s.u.) wählt, sollte p eine Primzahl sein, da dann mindestens p/2 Verschiebungsversuche stattfinden, bevor frühestens eine Adresse doppelt auftritt.)

Zu 2) Die einfachste Möglichkeit, Kollisionen zu behandeln, besteht darin, Datenobjekte mit gleichen Hash-Funktionswerten in einer linearen Liste zu verketten. Dieses Verfahren heißt direkte Verkettung. Die Suche nach einem Datensatz mit dem Schlüssel k besteht aus der Berechnung von h(k) mit anschließender sequentieller Suche nach k in der zu h(k) gehörenden Liste.

Abb.2 Direkte Verkettung bei Kollisionen

Beispiel: In Abb.2 kollidieren die Datensätze mit den Schlüsselworten "Lindemann", "Lottemann" und "Schneider"; sie sind direkt verkettet.

h(Lindeman) = h(Lottemann)
= h(Schneider)
= 2096.

Gängige Kollisionsverfahren, die keinen "Überlaufbereich" für lineare Listen verwenden, sondern den vorgegebenen Adreßraum A benutzen, sind die lineare und die quadratische Verschiebung.

Lineare Verschiebung: Falls h(k) bereits durch einen anderen Schlüssel besetzt ist, dann versuche, k in den Adressen h(k) + c, h(k) + 2c oder h(k + 3c,... unterzubringen (c ist eine Konstante).

Quadratische Verschiebung: Probiere bei einer Kollision nacheinander, ob h(k) + 1, h(k) + 4, h(k) + 9, h(k) + i^2,... noch frei ist. Die quadratische Verschiebung hat sich in der Praxis bewährt, da die Sekundärkollisionen hierbei kaum ins Gewicht fallen. Dagegen sollte diese quadratische Verschiebung nicht gewählt werden, wenn Schlüssel auch gelöscht werden müssen.

Theoretische Versuche und praktische Messungen haben ergeben, daß der durch A gegebene Speicher nur zu 80 % gefüllt werden sollte. Man muß dann beim Suchen im Durchschnitt mit höchstens drei Kollisionen rechnen. Bei mehr als 80 % steigt die Zahl der Kollisionen und damit die Zugriffszeit rasch an.

Inside ZF

Das waren die Ausführungen aus dem “DUDEN Informatik”. Aber was fange ich damit an ? Jetzt heißt es, “konkreter” zu werden. Die Frage nach dem “Wie” entbehrt im Augenblick noch jeder Antwort. Ein erprobtes Mittel ist das “Abgucken”. Am Besten wird es sein, wenn ich mich direkt im ZF “umsehe”. Dort müssen mehrere Worte mit dem HASHING arbeiten. Das Wort FORGET arbeitet zum Beispiel im Wörterbuch und mit dem Wörterbuch...

```
: FORGET ( -- )
    BL WORD ?UPPERCASE DUP CURRENT @ HASH @
    (FIND) 0= ?MISSING DUP >VIEW (FRGET) ;
```

Treffer: FORGET nutzt das Wort HASH. Aber was HASH “macht”, das weiß ich auch nicht. Die Stackbilanz gibt vielleicht Auskunft:

CODE HASH (str-addr voc-ptr -- thread)

'str-addr' entspricht nach dem Aufruf von FORGET der Adresse, die HERE liefert, könnte aber auch jede beliebige, andere Stringadresse sein. 'voc-ptr' ist der Inhalt von CURRENT, also die CFA des gerade aktuellen Vokabulars, was leicht zu überprüfen ist.

...aktuelles Vokabular sei FORTH

```
CURRENT @ BODY> >NAME .ID    FORTH ok
```

Dann will ich mal sehen, 'was HASH so macht'...

```
CODE HASH    ( str-addr voc-ptr -- thread )
                    CX  POP   \ 'Vokabularadresse'
                    BX  POP   \ Adr. von HERE, entspr.
                              \ Adr. der  Zeichenkette
                    BX  INC   \ COUNT wird überspr.
                0 [BX] AL  MOV \ 1. Zeichen nach AL
 #THREADS 1- #      AX  AND   \ 15 mit AX 'unden' :-)
                    AX  SHL   \ AX mit 2 multiplizieren
                CX  AX  ADD   \ Vokabularadresse auf AX
                              \ addieren
                  1PUSH       \ [AX] auf den TOS geben
END-CODE
```

HASH berechnet also einen Wert, der nach folgender Vorschrift erzeugt wird:

Nehme den ersten Buchstaben, verknüpfe diesen logisch UND mit der Zahl 15, multipliziere das Ergebnis mit 2 und addiere darauf die CFA des gerade aktuellen Vokabulars.

Wenn man die ASCII-Werte beliebiger Buchstaben mit 15 AND-verknüpft, erhält man Zahlen zwischen 0 und 15 (A=1...O=15) und (P=0...Z=10). Das heißt, das hier eigentlich gar kein so bedeutsames 'Hashing' stattfindet, jedenfalls nicht in dem Sinne der bisherigen Aussagen. Erstens wird

nur ein Zeichen 'gehasht', und zweitens gibt es nicht einmal für jeden Buchstaben eine separate 'Zahlenentsprechung'. Vielmehr irritiert mich aber im Augenblick das “Einbinden” der CFA des gerade aktuellen Vokabulars. Vermutlich liegt hierin die ´Ursache´ dafür, daß mir WORDS stets nur die Worte des gerade aktuellen Vokabulars anzeigt. WORDS *.* liefert dagegen alle Worte aller Vokabulare auf den Bildschirm. "Da muß noch etwas Anderes sein...". Zunächst will ich mich davon überzeugen, ob tatsächlich Worte mit unterschiedlichen Anfangsbuchstaben den gleichen Hashfunktionswert haben können.

```
: HashTest ( -- )
  BL WORD ?UPPERCASE CURRENT @ HASH @ ;

    HashTest ALLOC . 14488 ok
    HashTest QTYPE . 14488 ok
```

Das läßt nichts Gutes erahnen. Da hat der Tom Zimmer wohl geschummelt ?

```
:  .Namen ( thread -- )
  YSEG @                    \ Da liegen die Header
  SWAP 3 +                  \ Wenn thread = Offset auf
                            \ Def.Namen, dann
  BEGIN                     \ vermutlich 3 Byte weiter
                            \ (s. ZF-Intern)
    2DUP C@L                \ Byte aus Segment lesen
    DUP 128 AND             \ gegen MSB prüfen
                            \ (ENDE-Kennung)
    128 = IF  128 - EMIT    \ gegebenenfalls MSB entfernen
              2DROP EXIT    \ wenn ENDE erreicht, dann
                            \ Adressen wegwerfen
      ELSE EMIT             \ ansonsten Zeichen emittieren
      THEN
    1+                      \ Offset zeigt auf nächstes Byte
  AGAIN ;
```

Laut HASH's Stackbilanz ist die Ausgabe von HASH ein Thread, ein Faden. In der Konstanten #THREADS (Anzahl der Threads) ist die Zahl 16 festgeschrieben. Meine erste Vermutung: Das 'Minimalhashing' ist in Fäden oder Ketten organisiert, von denen es maximal 16 Stück gibt. Dann dürfte die 'Kollision' wirklich zur Regel werden ! Im Augenblick interessiert mich aber mehr, was man mit einem solchen 'Faden' anfängt.

Mal nachdenken: Im Aufsatz/Seminar ZF-Intern; Forthgruppe Moers; Herbst '91; Friederich Prinz (wie schreibt man einen ganz besonders dicken Smilie **8=()**) steht beschrieben, wie ZF's Worte im Headersegment aufgebaut und wiederzufinden sind. Sollte der 'Faden' etwa eine Kette aus Pointern in das Headersegment sein ? Auch das läßt sich leicht überprüfen !

Probieren wir's aus:

```
    HashTest Empty .Namen       EMPTY ok
Noch ein Treffer !!!

    HashTest Addeditor .Namen   APPENDEDITOR ok
```

...und schon liege ich auf der Nase **:-(** Ich hab's mit verschiedenen Worten ausprobiert - manchmal klappt es, manchmal nicht. Gelegentlich kommen auch ganz unsinnige Ergebnisse heraus. Aber klar, die Kollision ! Also scheint 'thread' wirklich so etwas wie ein Faden zu sein, sozusagen der Anfang einer 'Liste'. An diesem Anfang ist das angefragte Wort zu finden, wenn es das erste Wort seiner Art war ?

Dagegen stand in Hannes' Text von Ray Duncan etwas über den "most rescent entry", also über den jüngsten Eintrag...

Das jüngste Wort, das im Wörterbuch mit READ... beginnt, ist READEDITOR. Also noch ein Test:

```
    HashTest Readeditor .NAMEN READEDITOR ok
```

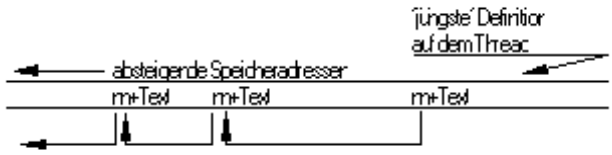
...klappt ja, dann aber richtig testen...

```
  : READMICH ;   \ einfach so...

    HashTest Readeditor .Namen READMICH ok
```

Also doch - HASH liefert das Ende eines Fadens, an dem alle Worte der gleichen, minimalen 'HashArt' hängen ! Wie geht es dann weiter ?

In FORGET arbeitet nach HASH @ das Wort (FIND). Vielleicht gibt mir das nähere Hinweise.



So ähnlich scheint ein “Faden” im ZF schematisch aufgebaut zu sein.

Laut dem "DUDEN Informatik" ist ZF's 'ListenHashing' die einfachste Strategie zur Vermeidung von Kollisionen. Einträge mit gleichen Hash-Funktionswerten werden zu einer linearen Liste (oder Thread, oder Faden) verkettet. Tom Zimmer hat hier noch eins drauf gegeben, bzw. ZF's Hashing noch einmal dahingehend vereinfacht, daß er nicht für jeden großen Buchstaben eine jeweils eigene Kette nimmt, sondern nur insgesamt 16 Threads zuläßt. Daß hierbei die Kollision zur Norm wird, bzw. daß relativ früh definierte Worte einen längeren Weg auf dem Thread beinhalten, sollte aus den bisherigen Überlegungen klar geworden sein. Womit muß ich nun aber in der Praxis rechnen ?

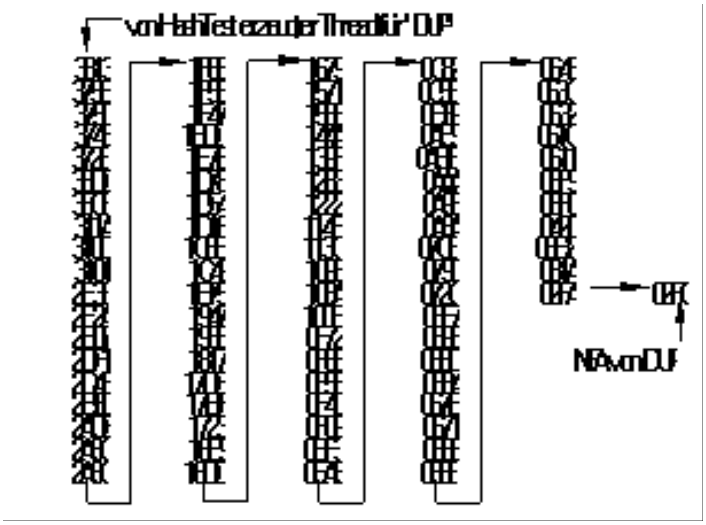
Der einfachste Weg mich zu vergewissern, daß meine Annahme bezüglich der Threads stimmt, schien mir ein Trace-Lauf

mit dem BXDEBUG zu sein. Wer's nicht kennt : BXDEBUG ist ein kleiner, sehr einfach zu bedienender und trotzdem ungewöhnlich komfortabler Debugger, der mit dem F-PC 'mitgeliefert' wird. BXDEBUG stammt nicht von Tom Zimmer, sondern von Sonam G. Gyato; Oasis Software, und ist das Einzige, was ich vom gesamten F-PC Paket behalten habe. Ich benutze das Programm immer dann, wenn ich mit ZF's Assembler arbeite, weil ich BXDEBUG aus ZF heraus aufrufen kann, ohne ZF 'abzumelden'. Das hat den Vorteil, daß ZF im Speicher bleibt. Ich muß mir lediglich vorher alle relevanten Adressen notieren (CS; SP0; HERE; Einstieg in die 'zu tracende' Routine etc.) und kann dann quasi 'von außen' gucken, wie ZF intern arbeitet...

Ich zeige hier zunächst, was durch BXDEBUG “sichtbar” wurde:

```
CODE (FIND) ( here thread -- cfa flag | here false )
            DX  POP     \ Offset in das Y-Segment
         DX DX  OR      \ Vergleich auf '00'
 0= IF   AX AX  SUB      \ AX '00' setzen
                1PUSH   \ und auf den TOS legen (FALSE)
 THEN
     YSEG #) ES MOV     \ Y-Seg. nach ES laden
 BEGIN DX BX  MOV       \ Y-Offset an BX übergeben
               BX INC   \ 2 Byte weitergehen, ab da steht
                        \ der COUNT
               BX INC   \ (siehe ZF-Intern)
               DI POP   \ HERE nach DI übernehmen...
               DI PUSH  \ ...und wieder auf den TOS
                        \ zurücklegen
 ES: 0 [BX]    AL MOV   \ 1. Zeichen ab Y-Seg. + Offset
                        \ vergleichen
      0 [DI]   AL XOR   \ mit 1. Zeichen ab HERE
                        \ ( gleiche Zeichen = 0 )
        63 #   AL AND   \ ASCII 'ausmaskieren'
                        \ ( AL = 0 AND 63 = 0 )
 0= IF ( 1 )            \ Wenn die Zeichen (COUNT !)
                        \ gleichen waren...
 BEGIN          BX INC  \ 'Zeiger' auf die Segmente um
                        \ 1 erhöhen ...
               DI INC
 ES: 0 [BX]    AL MOV   \ ... Buchstaben vergleichen, bis
      0 [DI]   AL XOR   \ das Ergebnis des Vergleichs
 0<> UNTIL              \ NICHT mehr '0' ist.
       127 #    AL AND  \ das höchstwertigste Bit ist im
                        \ letzten Zeichen einer Definition
                        \ gesetzt. Das wird hier maskiert.
 0= IF ( 2 )
               DI POP   \ 0 heißt, daß NUR das höchstw.
                        \ Bit gesetzt war
 ES: 1 [BX]    AX MOV   \ ein WORD ab Offset [BX] aus
                        \ dem Headersegment
               AX PUSH  \ auf den TOS legen
            DX BX MOV   \ ursprünglichen Offset nach BX
                        \ laden
               BX INC
               BX INC   \ Inhalt von BX um 2 erhöhen
 ES: 0 [BX]    AL MOV
      64 #     AL AND   \ Ist das Byte ein Zeichen aus
                        \ [A...Z] ?
  0<> IF ( 3 )
         1 #    AX MOV  \ 'TRUE'  nach AX
 ELSE -1 #      AX MOV  \ 'FALSE' nach AX
 THEN ( 3 )
               DS PUSH
               ES POP   \ ExtraSegment wieder mit
                        \ DatenSegment laden
                  1PUSH
 THEN ( 2 )
 THEN ( 1 )
```

Comment:
Und jetzt kommt es ! Wenn schon der COUNT ungleich war, braucht gar keine weitere Prüfung mehr zu erfolgen. Der Zweig ab dem IF (1) wird überhaupt nicht ausgeführt. Statt dessen wird das 16-Bit Wort, auf das der Faden 'zeigt', aufgenommen. HIER WIRD DIE KETTE WEITERGEREICHT !



;comment

```
            DX  BX  MOV   \ ursprünglichen Thread nach BX
                          \ laden
 ES: 0 [BX] DX  MOV       \ und das WORD ab dieser
                          \ Adresse nach DX laden
            DX  DX  OR    \ Der Zeiger ist weitergesetzt.
 0= UNTIL
               DS  PUSH
               ES  POP    \ ExtraSegment wieder mit
                          \ DatenSegment laden
            AX  AX  SUB   \ AX auf '00' setzen ...
                    1PUSH \ ... und auf den TOS geben
END-CODE
```

Jetzt ist mir klar, warum Tom Zimmer das Ding “Faden” getauft hat :-)

Um ein einfaches, und oft benötigtes Wort wie DUP zu finden, muß ZF in seiner HashTabelle (richtiger: auf dem zugehörigen HashFaden) nicht weniger als 87 Schritte 'gehen', um das Wort überhaupt zu finden. Der eigentliche Vergleich auf die Zeichenkette hat noch gar nicht stattgefunden.

Es wird aber auch deutlich, daß meine Vermutung richtig war: der von HASH gelieferte Thread ist das Ende des Fadens, an dem die Worte 'gleicher Art' aufgereiht sind. Die Adressen steigen ab dem 'Start' des Fadens ständig ab, gewissermaßen in ZF's Tiefe und zu seinen "älteren" Worten.

Wie bei (FIND) zu sehen war, wird der Vergleich der Zeichenketten ab HERE und im Headersegment erst dann eingeleitet, wenn wenigstens der COUNT gleich ist. Logisch - anderenfalls kann es sich gar nicht um identische Strings handeln.

Das relativiert Ray Duncan's Aussage natürlich gewaltig. Auch um DUP zu finden, benötigt ZF nur einen einzigen Stringvergleich. Aber die vorhergehende Arbeit kostet doch einiges an Zeit, wenn auch weit weniger, als wenn jeder Header einem Stringvergleich unterzogen werden müßte.

Wie könnte man das schneller machen ? Ein erster Ansatz wäre vermutlich, die Konstante #THREADS von 16 auf 26 zu erhöhen. Im Thread von DUP hängen zum Beispiel alle Worte, die mit 'D' beginnen, aber auch alle Worte, die mit 'T' beginnen. Ich weiß nicht, ob ich mit DUP zufällig einen Extremfall erwischt habe. Vermutlich gelten ähnliche Bedingungen für alle Worte, die in den “Tiefen” des Wörterbuches definiert sind und obendrein einen “Nachbarn” haben. Nachbarn, bzw. Kombinationen aus Anfangsbuchstaben sind jedenfalls:

A-Q B-R C-S D-T E-U F-V G-W H-X I-Y J-Z

Das wird nicht wenige Worte betreffen.

Noch etwas fällt auf: Wenn nur der erste Buchstabe 'gehasht' wird, und wenn immer dann ein Vergleich der Zeichenketten stattfindet, wenn der Count beider Zeichenketten gleich ist, dann hat diese Definitionsfolge verheerende Folgen für Suchläufe:

```
VARIABLE Ventil1  \  Jedes der Worte liefert den
VARIABLE Ventil2  \ gleichen Thread. Alle haben den
VARIABLE Ventil3  \ gleichen Count.
VARIABLE Ventil4  \  Wenn nach 'Ventil1' gesucht
VARIABLE Ventil5  \  gesucht werden soll, dann müssen
VARIABLE Ventil6  \ alle anderen Ventilnamen über
VARIABLE Ventil7  \ direkte Zeichenkettenvergleiche
VARIABLE Ventil8  \   ausgeschlossen werden ! Ich hab's
VARIABLE Ventil9  \ natürlich wieder mit BXDEBUG
                  \ nachgesehen. Leider, es ist so !
```

ZF hat bei mir 1.220 Definitionen in allen Vokabularen. Dividiert man die Wortanzahl durch die Anzahl der Threads, also durch 16, erhält man, laut meinem Taschenrechener, 76,25. Ergo sollte für die “ältesten” Worte im Dictionary gelten, daß ungefähr 76 “Schritte” auf dem Faden notwendig sind, um das jeweils gesuchte Wort tatsächlich ausfindig zu machen. Ich wollte einfach nachsehen können, wieviele Schritte im Einzelfall wirklich jeweils notwendig sind. Das jedesmal mit dem BXDEBUG schrittweise auszutesten, wäre natürlich ebenso unsinnig wie ermüdend. Deshalb habe ich das Wort (FIND) ein kleines Bißchen modifiziert - und FINDE daraus gemacht. Vielleicht hat der eine oder andere FORTHer das Bedürfnis, selbst einmal nachzusehen. Weil ich (FIND) bereits im Detail beschrieben habe, will ich den gesamten Code von FINDE nicht noch einmal aufschreiben. Statt dessen sei lediglich auf die Änderung, bzw. Einfügung hingewiesen:

Direkt unter die Zeile

```
    DXDXOR            \ Der Zeiger ist weitergesetzt.
```

habe ich einen Zähler eingefügt:

```
1 # #Elemente #)ADD     \ Zähler inkrementieren
```

```
VARIABLE #Elemente   \ Zähler für die Anzahl der Schritte,
   \ die auf dem Thread notwendig sind, um ein beliebiges
   \ Wort im Dictionary zu finden.

\  CODE FINDE .....

: Suche ( -- )                ( Aufruf : Suche empty )
  0 #Elemente !
  BL WORD ?UPPERCASE DUP
  CURRENT @ HASH @
  Finde
  2DROP
  CR
  HERE COUNT TYPE ."  ==> "
                              \ Der Definitionsname liegt noch
  #Elemente @ .               \ bei HERE
  ." ListenElement(e) " ;
```

Selbstverständlich wird im Quelltext vor FINDE die Variable “Elemente” definiert. Experimente mit dem Wort ´Suche´ zeigen, daß die Annahme von ca. 76 “Schritten” bei den ältesten Worten ungefähr stimmt. Plus/Minus 10 Schritte ist das die mittlere Anzahl der Schritte auf einem Thread, die getan werden müssen, um ein Wort wie zum Beispiel DROP aufzufinden.

Jetzt ist es an der Zeit, auf eine der eingangs beschriebenen Voraussetzungen einzugehen, auf den Mut, Fehler zu machen. Ich habe einige Seiten zuvor vermutet, daß Tom Zimmer mit ZF's Hashing “geschummelt” habe. Mittlerweile ist deutlich geworden, daß dem nicht so ist. Ich habe es nur nicht sofort verstanden...

Mit Fragen zum HASHING hat's angefangen. Ich hätte mich, wie früher schon, gar nicht erst darauf einlassen sollen, aber jetzt ist's zu spät. Ich stecke wieder einmal 'knietief' in den Eingeweiden 'meines' ZF - das ich eigentlich nur nutzen will.

;-) fep

Bevor ich ein paar wenige Worte über unser gestriges SVFIG Treffen sagen will, muß ich etwas zu meinem Bericht in der VD 3/99 klarstellen: Unter den von mir erwähnten "Doktoren" haben wir Dr. C.H. Ting, der einen Titel in Chemie hat, und auch Dr. Alfred Tang, mit einem Titel in Mathematik. Ich benutze keine Rechtschreibprüfung, aber die würden hier sowieso nicht helfen. Falls ich nun alles durcheinander gebracht habe, bitte ich um Verzeihung.

Ich sagte, glaube ich, in meinem vorigen Brief, daß Studenten die freien Zeitabschnitte zwischen den Unterrichtsveranstaltungen bevorzugen. Das Treffen am 24. Juli schien diesen Gedanken zu unterstützen. John Rible, kam, obwohl er aufgrund seiner Arbeit wenig Schlaf hatte, früh an, um seine dritte Vorlesung über CPU-Design zu halten. Wir hatten bis zu zehn Leute als Zuhörer, von denen wohl nur 2 oder drei hautpsächlich wegen seiner Vorlesung gekommen waren. So erzählte John mehr über seine gegenwärtigen Aktivitäten mit dem ShBoom-Prozessor als über das CPU-Design mit dem Xilinx-Board. Er gab uns auch eine Reihe auf Web-Adressen über 16-Bit Forth-Prozessoren: John Allen's "trailing edge" 16 und Honkong's MSL16. Ich weiß nicht, ob jemand daran interessiert ist, daher wiederhole ich die Adressen hier nicht.

Einige Leute kamen zum Plaudern beim Mittagessen und einige verliesen uns, so waren wir kaum jemals mehr als ein Dutzend im Raum.

Nach dem Essen präsentierte Dr. Ting einige Ideen, Forth zu propagieren und vielleicht ein wenig Einkommen für die Gruppe zu beschaffen. Der Rest des Nachmittags war eine große freie Diskussionsrunde, die gelegentlich zu Diskussionen über die Einzigartigkeit und die Vorzüge von Forth führte.

Ich wünschte, ich hätte alles aufzeichnen können, was Alan Furman zu sagen hatte, da er es so beredsam vortrug. Aber ich schließe wohl jetzt besser, da ich zu viel geschrieben habe, um jedermann (einschließlich mir selbst) viel Gutes zu tun. Ich möchte gerne von Euch wissen, wie man das Word "Granularity", als eine der Charakteristiken eines guten Programms, die zu einfachem Test und Fehlersuche führt, übersetzt. Nach Alan's Meinung ist "Testability" (Testbarkeit) eines Programms am wichtigsten. Große Module explodieren exponentiell.

Das solls gewesen sein, Leute!

Henry

Lieber Friederich,

jetzt bist Du wohl von Deinem Motoradausflug nach Mittweida und anderen Teilen Deutschlands zurückgekehrt. Ich hoffe, daß es ein guter Urlaub für Dich war und das Du die Chance hattest, dem zukünftigen Professor (Georg Beierlein) mitzuteilen, daß seine Knobelaufgabe im fernen Kalifornien gelesen wurde. (Nein, ich kannte die Lösung nicht. Ist sie dicht an dem dran, was Georg erwartete?)

Nun mein Bericht... Aufgrund des Zeitplans des Cogswell College wurde das SVFIG-Treffen gestern abgehalten, eine Woche eher als sonst. Vielleicht deswegen oder weil John Rible's Vorlesungen zum CPU-Design die Leute zu sehr an die Schule erinnern, war die Teilnahme am Morgen ziemlich knapp. Aber sie wuchs auf über 20 am Nachmittag, dank eines Gastsprechers von Forth Inc., die in Los Angeles ansässig sind; etwa soweit von Silicon Valley entfernt wie Stuttgart von Hamburg.

Obendrein schaffte es John Rible nicht mal selbst zu dem Treffen und so füllte wiedereinmal unser 'immer zuverlässiger' Dr. Ting die Lücke mit ein paar Rückblicken auf seine aktuelle Arbeit, eForth auf die verschiedenen Mikroprozessoren von Chuck Moores P-Serie zu portieren. Die 8-Bit Tools sind immer noch aktuell.

Rick VanNorman, unser Gastsprecher am Nachmittag, bezeichnete sich selbst als "Forth Zigeuner" bevor er vor 3 Jahren bei Forth, Inc. anfing (Da gibt es eine kurze Biografie am Ende seines Artikels über "Fuzzy Forth" in Forth Dimensions, März/April 1995.). Er scheint jetzt die treibende Kraft ("the single mind") hinter SwiftForth zu sein, der aktuellste Weg Windows 95, 98 oder NT Applikationen in voller Übereinstimmung mit ANS Forth zu entwickeln.

Nachdem er SwiftForth Version 2.0 gerade am Tag zuvor fertiggestellt hatte, war Rick sehr informativ bei dessen Beschreibung und gab so bereitwillig Antworten auf alle Fragen des interessierten Publikums, daß er fast vergaß, seine Erfindung 'lokaler Objekte', ein neues Feature in SwiftForth, zu erwähnen.

SwiftForth sollte nicht mit einem anderen Produkt von Forth, Inc. verwechselt werden: Swift-X, ein PC-basierter Target Compiler für verschiedene Prozessoren. SwiftForth wird für 395$ verkauft und SwiftForthPro (welches den Quelltext einschließt, außer den Quellen für den Kernel selbst) für $1995. SwiftForth für Linux kann bald erwartet werden, aber es wird keinen Support für 16-Bit DOS oder den Mac geben. Im Moment gibt es keine Verbindung zwischen SwiftForth und Win32Forth.

Wir müssen Kevin Appert, der der einzige übriggebliebene Forth Anwender bei Lockheed Martin in Sunnyvale ist, dafür danken, daß er seinen früheren Kollegen einlud, bei uns zu sprechen. Kevin schloß das Treffen mit einigen wenigen Tips zur Hardware-Fehlerbeseitigung und mit einer Perle voll Weisheit: "**Traue niemals einem Computer, den Du nicht anheben kannst**." ("Never trust a computer that you cannot lift.") Bis zum nächsten mal verbleibe ich Euer "freundlicher Reporter",

Henry

Ich muß gestehen, daß ich, auch als Direktor der FG, nicht immer alle Namen der Mitglieder parat habe und darum nicht allen Gesichtern auf dem Tagungsphoto einen Namen zuordnen kann. Wer kann hier dabei helfen, dem Henry seine Bitte zu erfüllen?

fep

Eakers CASE für Assembler in ZF und Turbo-Forth

von Fred Behringer <behringe@mathematik.tu-muenchen.de>
Planegger Str. 24, 81241 München

Warum sollte man für **CASE** auch in jenen Situationen High-Level-Forth verwenden, in denen man für die verschiedenmöglichen **OF**-Fälle sowieso "nur" **CODE**-Definitionen einsetzen möchte?

Stichworte: ZF, F83, Turbo-Forth, Assembler, CASE, Jxx-Sprünge über 32 K.

"Das steht doch schon alles irgendwo!" - ? Für Assembler, in **CODE**-Definitionen? In den folgenden Forths ist es nicht enthalten: ZF, Turbo-Forth, WebForth, F-TP, Rick VanNormans OS/2-Forth - soll ich weitersuchen? Ich lasse mich gern belehren.

Getestet habe ich das Wohlverhalten des Programms in ZF und in Turbo-Forth. Es wird viel von "Compiler Security" geredet, also von Narrensicherheit gegen Fehleingaben. Ich glaube nicht, daß es nötig ist, im Assembler-**CASE** eigene Fehlerabfragen einzubauen. Die ganze Konstruktion ist nichts weiter als ein fortlaufend geschachteltes **IF--ELSE--THEN** und die Fehlerabfragen werden von diesen Strukturen erledigt.

Dr. Eakers CASE (für Highlevel-Forthworte) ist eine schöne Sache. Ich kann mir kein Arbeiten mit Forth mehr vorstellen ohne **CASE**. In Forth 83 scheint es noch nicht enthalten gewesen zu sein. Zech bespricht es ausführlich, aber in seinem Forth-83-Glossar wird es nicht aufgeführt. ZF scheint es auch nicht eingebaut zu haben. Turbo-Forth hat es, schon immer, von der ersten Version an. In ANS-Forth wurde es in das Core-Extension-Word-Set aufgenommen und damit in beträchtlichem Maße anerkannt - offiziös gemacht, möchte ich sagen. Ich schwöre auf **CASE**. Es ist so herrlich übersichtlich aufgebaut und so einfach anzuwenden.

CASE bei (jedem) Bedarf (immer wieder) selbst zu konstruieren, wäre natürlich kein größeres Problem: Man beginnt mit einem **OVER** und einem **IF**, baut eine genügende Zahl von **ELSE**s ein, gibt eine noch größere Zahl von **THEN**s dazu, vergißt die passende Anzahl von restlichen **OVER**s und **IF**s nicht und schreibt das Ganze anständig hin, mit Einrückungen und so. Trotzdem: Wie sieht denn das aus? Wer soll sich denn die erforderliche Konstruktion merken? Wie soll man sie ständig bereithalten? Eakers **CASE--OF--ENDOF--ENDCASE** ist mit Leichtigkeit zu erledigen und damit einfacher.

Will ich CASE-Abfragen auf Maschinencode-Ebene machen, so ist das auch kein Problem: Ich nehme in die **CASE**-Konstruktion einfach lauter **CODE**-Definitionen auf. Was mich stört, ist nur, daß die ganze Abfragerei (**CASE** muß ja im allgemeinen viele noch nicht zutreffende Zeilen abfragen, bevor es zu der zutreffenden Zeile gelangt) in High-Level-Forth durchlaufen werden muß. Ist das nicht genau eine Stufe zu hoch, wenn man in der **CASE**-Konstruktion sowieso nur **CODE**-Definitionen unterbringen möchte?

Das untenstehende Programm liefert ein Eaker-**CASE** als Kontrollstruktur für den Forth-Intel-Assembler, das in jede **CODE**-Definition, ähnlich wie **IF--THEN** oder **BEGIN--UNTIL,** sofort eingebaut werden kann.

Etwas Furchtbares passiert aber, wenn man im untenstehenden Testprogramm in die einzelnen **OF--ENDOF**-Teile jeweils etwa 48 (oder mehr) **NOP**s einbaut. Sollte doch überhaupt nichts ausmachen - denkt man. Denkste! Bei 9, 8, 7, 6 geht noch alles gut. Nemma Problemma. Bei **5 TEST [RET]** steigt das System aber aus. Bei noch kleineren Eingaben ebenfalls. Was passiert? Bei 9 und 8 tritt überhaupt kein Sprung auf. Bei 7 tritt im Erfüllungsfall ein Sprung von ein paar Bytes mehr als 48 Bytes auf, bei 6 sind es 96 und noch ein paar, bei 5 dann aber schon mehr als 144. Spätestens bei 5 (von 9 her gesehen) tritt also im Erfüllungsfall ein Sprung (vom betreffenden **ENDOF** zum **ENDCASE**) über mehr als 128 Bytes auf. Die relativen Sprünge der Kontrollstrukturen sind aber in ZF, Turbo-Forth und den anderen 16-Bit-Systemen, die ich kennengelernt habe, mit den relativen Sprungbefehlen programmiert, die mit 7x beginnen und nur ein einziges Byte als (vorzeichenbehafteten) Operanden zulassen. Die genannten Systeme berechnen die erforderlichen Sprungweiten zwar richtig, kappen aber genialerweise den jeweils höherwertigen Teil mit Hilfe von "**C,**" einfach ab und lassen nur ein einziges Byte übrig. Das hat natürlich mit der eigentlich benötigten Sprungweite nicht das Geringste zu tun - und aus ist es. Und das Allerschönste dabei: Eine detaillierte Fehlermeldung oder wenigstens eine Warnung, daß man da etwas falsch gemacht hat, erwartet man vergeblich. Das merkt man erst, wenn das System aussteigt.

Was kann man tun? Ganz einfach. Verwenden Sie die abgeänderten Kontrollstrukturen aus meinem VD-Artikel "BEGIN--UNTIL über 32 K ab 80386 im ZF-Assembler" und alles wird gut - vorausgesetzt, Sie arbeiten nicht etwa immer noch mit einem 286er. Die paar Extrazeilen in jenem Artikel fallen überhaupt nicht ins Gewicht und nach "außen hin" merkt die **CASE**-Konstruktion des vorliegenden Artikels nichts davon, daß es sich bei den eingebauten **IF**s, **ELSE**s und **THEN**s jetzt um mögliche Sprünge von bis zu 32 Kilobyte nach oben oder nach unten handelt. Sie führt sie anstandslos aus.

Für Parameterübergaben in Assembler ist nicht so sehr der Stack das natürliche Mittel als vielmehr die CPU-Register. Ich vergleiche im Teil "**xx OF**" der **CASE**-Konstruktion den Wert **xx** mit dem Wert des Registers **AX**. Die restliche Syntax ist die vom "High-Level"-**CASE** her geläufige. - Soweit der Normalfall, als einfachste Lösung. Nun haben bestimmte Register aber bestimmte Spezialaufgaben, die man vielleicht in die **CASE**-Konstruktion hinein oder darüber hinweg retten möchte. Man denke an **CX** bei Schleifen oder **SI** und **DI** bei Stringverschiebungen. Leicht kann man sich auch Fälle vorstellen, in denen man **AX** nicht von **CASE** "verbraten" wissen möchte. Für solche Zwecke habe ich...

...als Rafinesse noch einen Assembler-Schalter eingebaut: >OF, mit der Syntax **AX >OF, BX >OF** usw. Zulässig sind die üblichen 16-Bit-Register: **AX, CX, BX, DX, BP, SP, SI, DI**. Mit **>OF** läßt sich nachträglich, mitten aus dem Assembliervorgang heraus, mitten im Assemblieren von **CASE**, das Register, mit dem die Werte in **xx OF** verglichen werden sollen, ändern. Diese Assembler-Direktive ist nicht nur "raffiniert", sie ist auch eines jener Elemente, die man sich nicht merken wird oder merken will. Daher habe ich es so eingerichtet, daß sie nur "nachträglich" eingebracht werden kann (siehe auch nächsten Abschnitt).

Falls meine Extras mit >OF nicht zusagen, einfach ignorieren! So, wie ich es programmiert habe, müssen zwar alle Worte, also auch **>OF**, mitgeführt werden, man kann aber so tun, als ob es **>OF** gar nicht gibt. Ich habe es so eingerichtet, daß die **CASE**-Konstruktion dann auf **AX** (als Registervorgabe) wirkt. Sicherheitshalber, falls man das vergessen sollte oder beim Programmieren etwas schiefläuft, habe ich in **CASE** und in **ENDCASE** die Umschaltung **AX >OF** vorgenommen. Das heißt aber, daß eine Umschaltung auf ein anderes Abfrageregister nur innerhalb der **CASE**-Konstruktion vorgenommen werden kann (man vergleiche hierzu **TEST2**). Das heißt aber auch (was man ja sowieso nach dem ersten Fehlversuch merken würde), daß im Quelltext die Worte **OF** und **>OF** bereits vor dem Wort **CASE** definiert werden müssen.

Präemptives Selektieren, also so eine Art lexikographisches Auswahlverfahren, kann man mit dieser Register-Umschalterei beispielsweise ebenfalls bequem einrichten. Im Wort **TEST2** gelangen alle Abfragen mit **AX = TRUE** in einen Topf, alle mit **AX = FALSE** und **BX = TRUE** in einen anderen, dann alle mit **AX = FALSE** und **BX = FALSE** und **CX = TRUE** ... usw.

OF und ENDOF sind Maschinencode-Makros. >OF ist ein reiner Assembler-Schalter. **CASE** ist noch hinreichend klar. Was ist **ENDCASE**?

An diejenigen, denen Compilersicherheit über alles geht: >OF ist kein Code, der sich im laufenden Betrieb selbst verändert (selbstmodifizierender Code). **>OF** ist ein "Metacode", der in einem bedingten Assembliervorgang Code erzeugt. - Oder sehe ich das falsch? (Wie viele andere Autoren auch, verwende ich "Code" als synonym zu "Folge von Maschinensprachbefehlen". Colon-Definitionen sind natürlich in gewisser Weise auch "Code". "Nur-Daten" sind sie nicht.)

Zum Schluß noch eine Frage zum Nachdenken: Führen Sie in ZF oder Turbo-Forth die im vorliegenden Artikel beschriebenen Neuerungen ein und anschließend die aus meinem VD-Artikel "BEGIN--UNTIL über 32 K ab 80386 im ZF-Assembler". Ergebnis: Nichts geht mehr. Warum nicht?

Programm-Listing

```
HEX  ASSEMBLER DEFINITIONS

: OF    # 200 CMP 0= IF ;\ Bezieht sich
                         \ auf AX als Vorgabe
: >OF   ['] OF 6 + !    ;\ BX >OF ersetzt AX
                         \ durch BX in OF usw
: CASE  AX >OF CSP @ !CSP ;
: ENDOF ELSE              ;
: ENDCASE
   [ FORTH ]              \ BEGIN, WHILE,
                          \ REPEAT aus FORTH !
   BEGIN  SP@ CSP @ -
   WHILE  [ ASSEMBLER ] THEN [ FORTH ]
   REPEAT CSP !
   [ ASSEMBLER ]
   AX >OF ;

FORTH DEFINITIONS

CODE TEST
   AX POP
   CASE
     1 OF  8 # BX MOV ENDOF
     2 OF  7 # BX MOV ENDOF
     3 OF  6 # BX MOV ENDOF
     4 OF  5 # BX MOV ENDOF
     5 OF  4 # BX MOV ENDOF
     6 OF  3 # BX MOV ENDOF
     7 OF  2 # BX MOV ENDOF
     8 OF  1 # BX MOV ENDOF
           9 # BX MOV
   ENDCASE
   BX PUSH NEXT END-CODE

CODE TEST2
   AX POP BX POP CX POP DX POP
   CASE
     TRUE OF 1 # DI MOV ENDOF BX >OF
     TRUE OF 2 # DI MOV ENDOF CX >OF
     TRUE OF 3 # DI MOV ENDOF DX >OF
     TRUE OF 4 # DI MOV ENDOF
             5 # DI MOV
   ENDCASE
   DI PUSH NEXT END-CODE
```

Auch die raffiniertesten Compiler kriegen den Assembler nicht tot ! Die virtuelle Maschine Forth auf der Registerebene möglichst nahe 'an die CPU heranzubringen', ist noch immer eine der Lieblingsbeschäftigungen vieler Forther. ***Berichten Sie uns von Ihren Arbeiten, bitte*** *!*

fep

LOGOS

- 1 -


Forth-Gesellschaft
FG
Deutschland e.V.

- 2 -


TH
Gesellschaft e.V.

- 3 -


4th
Gesellschaft e.V.

- 4 -


TH
Gesellschaft e.V.

Ein LOGO für die FG...

...wird gesucht !

Die Forthgesellschaft sollte ein 'Logo' haben, das sich in der VD, in unserer WEB-Site und auf Briefbögen mit hohem Wiedererkennungswert verwenden läßt.

Natürlich sollte das LOGO einen deutlich sichtbaren bezug zu Forth und zur Forthgesellschaft haben.

SIE sind aufgerufen, ein solches LOGO zu entwerfen - ganz gleich wie und womit - und Ihren Entwurf an die Redaktion der VD zu senden. Die Mitgliederversammlung im April 1999 soll, so in Ihre Vorschläge rechtzeitig hier eingehen, DAS LOGO der FG auswählen.

Dem Siegers dieses Wettberwerbs winkt als Preis die kostenlose Teilnahme an der Jahresversammlung 2000 !

Für das Direktorium

Friederich Prinz

Erinnern Sie sich an diesen Aufruf ?

Den haben Sie erstmals in der VD 1/99 lesen können (S. 36).

Die daraufhin bis zur Fertigstellung dieser Ausgabe hier eingegangenen Vorschläge sehen Sie auf der vorherigen Seite. Eine Auslosung / Abstimmung anläßlich der '99er Tagung war mangels Masse bedauerlicher Weise nicht möglich. Während der Tagung 2000 wird diese Auslosung aber sicher stattfinden. Bis dahin soll die 'Abgabefrist' verlängert werden.

Der ausgelobte Preis ist nach wie vor die Kostenerstattung zur Teilnahme an der Tagung 2000 in Hamburg. Gewinnt das LOGO eines Einsenders, der nicht an dieser Tagung teilnimmt, erfolgt eine Einladung zur Tagung 2001 !

Einsendungen sollten 'rechtzeitig' vor der Tagung bei der Redaktion der VD eingehen, damit die Präsentation der LOGOs ansprechend vorbereitet werden kann.

fep

Forthgesellschaft e.V.

Jahrestagung 2000

Hamburg

Bitte versäumen Sie es nicht, sich frühzeitig anzumelden ! Sehen Sie hierzu die Informationen auf der Rückseite dieser Ausgabe. Anmeldeformulare werden Sie als Beilage in der Ausgabe 01/2000 finden.

red

Holländisch ist gar nicht so schwer. Es ähnelt sehr den norddeutschen Sprachgepflogenheiten. Und außerdem ist Forth sowieso international. Neugierig ? Werden Sie Förderer der

HCC-Forth-gebruikersgroep.

Für 20 Gulden pro Jahr schicken wir Ihnen 5 oder 6 Hefte unserer Vereinszeitschrift 'Het Vijgeblaadje' zu. Dort können Sie sich über die Aktivitäten unserer Mitglieder, über neue Hard- und Softwareprojekte, über Produkte zu günstigen bezugspreisen, über Literatur aus unserer Forth-Bibliothek und vieles mehr aus erster Hand unterrichten. Auskünfte erteilt:

Willem Ouwerkerk
Boulevard Heuvelink 126
NL-6828 KW Arnhem
E-Mail: ***w.ouwerkerk@kader.hobby.nl***

Oder überweisen Sie einfach 20 Gulden auf das Konto 525 35 72 der HCC-Forth-gebruikersgroep bei der Postbank Amsterdam. Noch einfacher ist es wahrscheinlich, sich deshalb driekt an unseren Vorsitzenden, Willem Ouwerkerk zu wenden.

Forth in Taiwan

Berichtet von Fred Behringer
<behringe@mathematik.tu-muenchen.de>

Und wieder hören wir, dank der wunderbaren E-Mail-Verbindungen, von der Existenz einer weiteren Forth-Gruppe in der großen weiten Welt: FIG Taiwan, eine lose Vereinigung von Forth-Interessenten, die sich allmonatlich treffen, aber offensichtlich zu noch keiner festen Organisationsform gefunden haben. Gesprächspartner - sehr freundlicher Gesprächspartner - ist Sam Suan Chen <*sschen@iner.gov.tw*> . Gott sei Dank haben er und seine Familie das kürzliche Erdbeben heil überstanden. Nach Taiwan gebracht wurde Forth vor fast 20 Jahren von C.H. Ting, dem inzwischen auch bei uns bekannten Professor aus den USA, Schöpfer von eForth und anderen interessanten Dingen - Tiny BASIC und das Gleitkomma-Paket für den 486 (siehe Bearbeitung des Berichterstatters) gehören dazu. Sam erwähnt **MYSEE**, einen komfortablen **Decompiler für WIN32FOR**, zu beziehen über:

ftp://ftp1.sinica.edu.tw/forth/taiwan/win32for/mysee62.zip.

Ich werde mit Sam in Kontakt bleiben und versuchen, mehr über Forth in China in Erfahrung zu bringen.

beh

Lerserbrief von M. Pilot; Berlin

Beim Betrachten der gängigen Softwareangebote bemerkte ich, daß ein Computerspiel bezüglich der Wirtschaftspolitik fehlt. Vielleicht sollte es Finanzminister heißen. Eine Softwareentwicklergruppe müßte sich finden und neben den entsprechenden Datenrecherchen ein gut durchdachtes modulares Werteflußmodell der Wirtschaft implementieren (Lohnsegmente, Produktionsbranchen, Steuern, Soziales, Banken, Renten usw.), idealerweise gleich passend für Datensätze verschiedener Länder. Man könnte dann durch Variation eines aktuellen Datensatzes (z.B. Erhöhen der Mineralölsteuer, Stauchung der oberen Einkommensbereiche, Einsatz der freien Gelder zum Sponsern von Arbeitsstellen im unteren Lohnbereich, hierbei resultierende Verschiebungen in Konsumsegmenten können gleich mit eingerichtet werden, Steuereinnahmeverschiebungen usw.) verschiedene Fälle durchrechnen.
Das könnte passieren in Tages- oder Wochenschritten. Wie lange bleibt das Modell stabil bzw. ist die Differenz im richtigen Maße ? Mit Sicherheit rechenintensiv und somit etwas für Forth, als auch ein anspruchsvolles Spielzeug mit bildenden Folgen für den Anwender. Da die Softwareleute fixe Kerlchen sind, kriegen die das schon hin.
Neben einer ständigen Verfeinerung des Modells entstünde wahrscheinlich noch ein reger Austausch der Anwender übers Internet oder eine Zeitung bezüglich der gemachten Erfahrungen als auch neue Datensätze.
Außerdem könnte jeder X-Beliebige die Politiker durch nicht vom Tisch zu fegende Meinungen ärgern. Mehr als 500,- DM sollte es aber wohl nicht kosten. Vielleicht entdecke ich in den nächsten Jahren mal so ein Programm in der Softwarelandschaft, falls eine Softwaregruppe so etwas mal in Angriff nimmt. Ist ein solches Programm schon auf dem Markt, sind die Bemerkungen natürlich hinfällig.

M. Pilot

Die Idee, mit Forth Spiele zu programmieren, ist nicht ganz neu. Ich erinnere mich an Jörg Plewes Demo zu seinen Experimenten um ein 'Ballerspiel'. Das durften wir in Höfchen bei Mittweida bewundern. Ulrich Richters 'Schiebespiel' ist hinreichend bekannt und wird 'heftig genutzt'. Eine 'Wirtschaftssimulation' hat meines Wissens noch Niemand mit Forth in Angriff genommen. Warum eigentlich nicht ? Unter den Informatikern der FG werden sich doch Spezialisten aus dem Bereich 'Wirtschaft' finden lassen ? (Ein Schelm, wer Böses dabei denkt !)

Wie wäre es mit einer Diskussion über die VORAUSSETZUNGEN eines solchen Spiels ?

fep

Letzte Meldung:
Die Direktoren der FG haben sich vom 24. - 26. 09.'99 in Markgrafenheide bei Rostock getroffen, um verschiedene Belange des Vereins zu besprechen. Themen waren u.a. die Bereitstellung der VD auf den Web-Seiten der FG, und die Umstellung der E-Mail Adressen auf den neuen Server. *fep*

Einen englischsprachigen Sonderdruck der VD

wollen wir erstellen, um uns (die Mitglieder der FG und die FG selbst) den Forthern in den USA und in England vorzustellen. Der Artikelaustausch zwischen 'Vierte Dimension', 'Forth Dimension' und 'Forthwrite' funktioniert dank Fred Behringers Bemühungen mittlerweile prima (siehe Dixon: Reed-Solomon in diesem Heft). Aber wir können mehr !

Eine komplette Sonderausgabe 'auf die Beine zu stellen' ist gar nicht so schwierig. Was wir dazu brauchen, sind Ihre Vorschläge und Beiträge. Schreiben Sie einen Artikel, den Sie gerne in dieser Sonderausgabe veröffentlichen wollen. Oder nennen Sie uns einen Artikel aus einer der letzten Ausgaben der VD, den Sie gerne im Sonderdruck sehen würden.

Unsere Freunde in Übersee und 'auf der Insel' werden uns bei den Übersetzungen behilflich sein und Korrektur lesen. Sie können nur dann etwas falsch machen, wenn Sie gar nichts tun. Geben Sie es uns einfach...

fep

Kennen Sie eigentlich den SWAP der FG ?

Das Kunstwerk ist unter den Händen von *Rolf Kretschmar* entstanden und wurde der FG von *Michael Kalus* gestiftet.

Forth-Gruppen regional

Moers
Friederich Prinz
Tel.: 02841-58398 (p) (Q)
(Bitte den Anrufbeantworter nutzen !)
(Besucher: Bitte anmelden !)
Treffen: (fast) jeden Samstag,
14:00 Uhr, **MALZ, Donaustraße 1**
47443 Moers

Mannheim
Thomas Prinz
Tel.: 06271-2830 (p)
Ewald Rieger
Tel.: 06239-920 185 (p)
Treffen: jeden 1. Mittwoch im Monat
Vereinslokal Segelverein Mannheim e.V.
Flugplatz Mannheim-Neuostheim

München
Jens Wilke
Tel.: 089-89 76 890
Treffen: jeden 4. Mittwoch im
Monat, **China Restaurant XIANG**
Morungerstraße 8
München-Parsing

µP-Controller Verleih

Thomas Prinz
Tel.: 06271-2830 (p)
micro@forth-ev.de

Gruppengründungen, Kontakte

Fachbezogen 8051 ... (Forth statt Basic, e-Forth)
Thomas Prinz
Tel.: 06271-2830 (p)

Forth-Hilfe für Ratsuchende

Forth allgemein
Jörg Plewe
Tel.: 0208-49 70 68 (p)

Jörg Staben
Tel.: 02103-24 06 09 (p)

Karl Schroer
Tel.: 02845-2 89 51 (p)

Spezielle Fachgebiete

Arbeitsgruppe MARC4	Rafael Deliano Tel./Fax: 089-841 83 17 (p)
FORTHchips (FRP 1600, RTX, Novix)	Klaus Schleisiek-Kern Tel.: 040-375 008 03 (g)
F-PC & TCOM, Asyst (Meßtechnik), embedded Controller (H8/5xx// TDS2020, TDS9092), Fuzzy ds	Arndt Klingelnberg, Consultants akg@forth-ev.de Tel.: ++32 +87 -63 09 89 pgq (Fax -63 09 88)
KI, Object Oriented Forth, Sicherheitskritische Systeme	Ulrich Hoffmann Tel.: 04351 -712 217 (p) Fax: -712 216
Forth-Vertrieb	volksFORTH / ultraFORTH RTX / FG / Super8 / KK-FORTH Ingenieurbüro Klaus Kohl Tel.: 08233-3 05 24 (p) Fax : 08233-99 71 mailorder@forth-ev.de
Forth-Mailbox (KBBS)	0431-533 98 98 (8 N 1) Sysop Holger Petersen hp@kbbs.org Tel.: 0431-533 98 96 (p) bis 22:00 Fax : 0431-533 98 97 Helsinkistraße 52 24109 Kiel

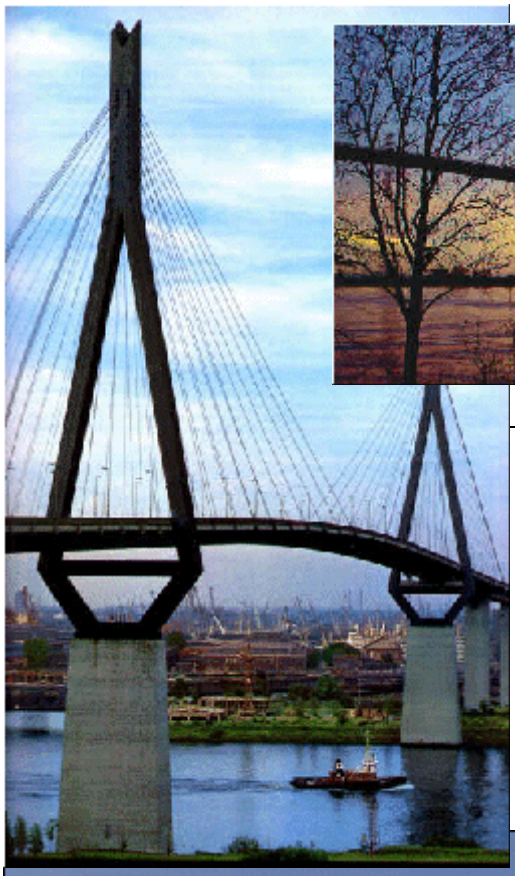
Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren ? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten ? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen ?

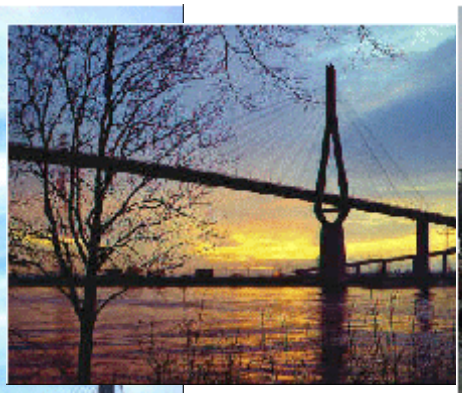
Schreiben Sie einfach der VD - oder rufen Sie an - oder schikken Sie uns eine E-Mail !

Hinweise zu den Angaben nach den Telefonnummern:

Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeistzeiten
g = geschäftlich

Die Adressen des Büros der Forthgesellschaft und der VD finden Sie im Impressum des Heftes.





Forth Jahrestagung 2000

- HAMBURG -

vom 14.-16.4.2000
mit der Option "13.4."

Es beginnt mit Kaffee am frühen Nachmittag des Donnerstag oder Freitag und endet mit dem Mittagessen am Sonntag.

Ich habe für FR-SO 10 Doppel- und 10 Einzelzimmer bestellt, für den DO 8 Doppel- und 5 Einzelzimmer.

Die Tagungskosten für die

	Mitglieder / Nichtmitglieder
Teilnehmer	310 / 340
Ermäßigt	260 / 260
Gäste	260 / 260
Donnerstag	120 / 120
Einzelz.	25 / 25 pro Tag zusätzlich

Diese Preise gelten bei Anmeldung bis 20.2.2000. Danach ist ein "Spätanmelderzuschlag" von einmalig DM 50,- zu zahlen.

Klaus Schleisiek

SEND Signal-Elektronik und Netz-Dienste GmbH
Stubbenhuk 10
D-20459 Hamburg
Tel: +49 40 375008-03
Fax: +49 40 375008-93
http://www.send.de