VIERTE DIMENSION

Volume VI/Nr.1 - März 1990

Themen

- Low-Cost-Automatisierung
- Mensch-ärgere-dich-nicht
- Hercules-Grafikkartentreiber
- Schnelles 68000-FORTH
- FORTH-Implementation auf einem Großcomputer

FORTH MAGAZIN

7,50 DM

FORTH-Tagung '90 in Frankfurt

EDITORIAL

Große Ereignisse werfen ihre Schatten voraus - die diesjährige FORTH-Tagung rückt immer näher. Sie findet - wie aus dem Titel unschwer zu erkennen ist - in Frankfurt statt. Hoffentlich finden die dort gesammelten Erfahrungen und Eindrücke ihren Niederschlag in zahlreichen Artikeln für die "Vierte Dimension".

Bei Erscheinen dieser "Vierten Dimension" schließt die CeBIT in Hannover gerade ihre Pforten. Es wäre interessant zu erfahren, ob der eine oder andere unserer Leser dort etwas Neues in Punkto FORTH entdeckt hat.

Einen Schwerpunkt des vorliegenden Heftes, bildet ein umfangreicher Artikel aus der DDR zum Thema "Low-Cost Automatisierung mit FORTH", der Ihnen einen Einblick in die Einsatzmöglichkeiten eines speziell für die Automatisierung entwickelten FORTH-Systemes bietet.

Desweiteren finden Sie in dieser Ausgabe der "Vierten Dimension" wie immer Interessantes und Anregendes zum Thema FORTH und wir wünschen Ihnen viel Vergnügen bei der Lektüre.

Rainer Aumiller

Denise Luda

(Das Bild Frankfurt/Main Römer stammt von einer Postkarte der *Foto Studio Collection* und wird mit freundlicher Genehmigung der Michel + Co., 6000 Frankfurt/Main, veröffentlicht.)

FORTH-Tagung '90 in Frankfurt

IMPRESSUM

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

Herausgeber:
FORTH-Gesellschaft e.V.

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

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

Quelltext Service

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

Autoren dieser Ausgabe:
Max Strobel, M. Schultheis, K. Kabitzsch, Jörg Staben, Andreas Döring, Jörg Plewe, Frank Stüss, Heinz Schnitter.

Erscheinungsweise:
Vierteljährlich

Redaktionsschluß:
Die zweite Woche im mittleren Quartalsmonat

Auflage:
ca. 1000 Stück

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

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

Vierte Dimension Inhalt

Leserbriefe und Zuschriften

Leserbrief zum Artikel "Der fleißige Biber" (VD V/3) oder: Sisyphus tatsächlich!

Fasziniert vom Biber-Problem, das Herr Prinz in seinem Artikel dargelegt hat, habe auch ich mich mit dieser Sache beschäftigt. Abgeschreckt durch die Zeiten, die in diesem Artikel angegeben werden, habe ich aber einen anderen Weg verfolgt: Mein Biber ist in Code definiert und arbeitet gleich im Bildschirmspeicher (Atari), so daß ich das Arbeiten des Algorithmus gleich ohne Zeitverlust beobachten kann. So benötigt der Rekordbiber aus dem Artikel von Herrn Prinz samt Anzeige knappe drei Sekunden statt über 29 Minuten, jener aus dem Leserbrief von Herrn Andreas Filmann eine gute Minute.

Um die Suche nach effizienten Bibern automatisieren zu können, habe ich den Code dann nochmals geändert: Es wird vorerst kein Stop-Code gesetzt, sondern der Biber bis zum Überlauf oder einem wählbaren Timeout getestet, wobei gespeichert wird, wann der jeweilige Zustand zum ersten Mal und wie oft aufgetreten ist. Soweit alles im Maschinencode. Jetzt wird (in High-Level-FORTH) kontrolliert, ob einer oder mehrere Zustände erst spät aufgetreten sind (variable Grenze), dort das Stop-Zeichen gesetzt und nochmals getestet. Gute Resultate werden gespeichert und die Struktur des Bibers anschliessend systematisch verändert. Obwohl dieses Programm in den letzten Monaten einige Millionen Biber getestet hat, kann ich höchstens bestätigen, daß wirklich fleißige Biber sehr dünn gesät sind: Ergebnisse ab etwa 15 gesetzte Bytes sind bereits sehr selten. Gute Ergebnisse erweisen sich zudem bei genauer Betrachtung oft als Strukturverwandt (die Zustände haben lediglich andere Namen) oder symmetrisch (alle Richtungen genau umgekehrt).

Fazit: Sisyphus, tatsächlich ...

Robert Epprecht
Fällanderstr. 1
CH-8124 Maus

Leserbrief zum Leserbrief von Johannes Teich, letzte VD

Herr Teich äußert in seinem Brief Zweifel an der Stichhaltigkeit des Argumentes der Klammerfreiheit für die UPN. Tatsächlich ist dieses Argument uneingeschränkt gültig. Das von Herrn Teich angeführte Beispiel, in dem er die Klammerung durch eine 'link-vor-rechts'-Regel ersetzt, ist ein Spezialfall und erlaubt keine allgemeinen Rückschlüsse. Wie ich in meinem Artikel 'Über das Wesen der UPN', VD Vol.V, Nr.2, Juni '89, klargemacht habe, erlaubt die Infixnotation *grundsätzlich* keine allgemeine Auswertung von links nach rechts. Hier ein Beispiel, wo ein solcher Versuch scheitern muß:

```
(3 + 4) * (5 + 6)
```

Auch die Anwendung von Rechenregeln hilft nicht weiter:

```
(3 + 4) * 5 + (3 + 4) * 6
```

läßt sich immer noch nicht von links auswerten, enthält dafür aber bereits unerwünschte Redundanzen.

Die Klammerfreiheit von UPN-Ausdrücken bleibt also ein gutes Argument gegen 'Ungläubige'. Puristen sollten ihren Spaß daran haben.

Jörg Plewe,
Mülheim/Ruhr

FORTH-Workshop

Im Herbst diesen Jahres wollen wir einen FORTH-Workshop in der DDR durchführen. Anbieter von FORTH-Systemen und Applikationen sollen die Möglichkeit haben, ihre Produkte vorzustellen und mit potentiellen Anwendern und Interessenten ins Gespräch kommen.

Um uns einen Überblick über den Umfang der angebotenen Leistungen und Produkte zu verschaffen, teilen Sie uns Ihr Angebot bitte umgehend unter folgender Anschrift mit:

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

Bericht der lokalen Gruppe Rhein-Ruhr

Das Ruhrgebiet -
Ein starkes Stück
Deutschland

Die lokale Gruppe Rhein-Ruhr lebt!

In der VD war schon zu lange nichts mehr über die Aktivitäten der lokalen Gruppe Rhein-Ruhr zu lesen. Deshalb dieser Artikel, um den Blick der Welt wieder ein wenig in unsere Richtung zu lenken. Für diejenigen, die noch nie etwas von uns gehört haben, hier ein Einstieg in die Szene:

Geschichte

Obwohl der Name 'Lokale Gruppe Rhein-Ruhr' erst Ende 1987 bestimmt wurde, treffen sich die Mitglieder dieser Gruppe schon wesentlich länger. Die Gruppe Rhein-Ruhr ist nämlich nicht aus dem Nichts, sondern durch Umbenennung der lokalen Gruppe Wuppertal entstanden. Zu dem besagten Zeitpunkt erfolgte die Abwanderung des damaligen lokalen Koordinators Michael Kalus, vielen wahrscheinlich noch als Editor der VD bekannt, in nördlichere Gefilde. Da der neue lokale Koordinator nicht aus Wuppertal, sondern aus Oberhausen stammt, und auch ein Großteil der Mitglieder sich aus dem weiteren Umland rekrutiert, wurde der Name 'Lokale Gruppe Wuppertal' als unpassend erkannt und in 'Lokale Gruppe Rhein-Ruhr' geändert, was auch wesentlich wichtiger klingt. Da die damaligen Treffen im Ottenbrucher Bahnhof in W'tal, einer Kneipe, an Zustrom verloren (Mitglieder beim letzten Treffen '87: 2), wurde beschlossen, die Treffen bei den jeweiligen Mitgliedern selbst abzuhalten. Da einige Mitglieder über geeignete Räumlichkeiten verfügen, die mit Rechnern o.ä. ausgestattet sind, hat sich die Zahl der Teilnehmer inzwischen auf einen Mittelwert von ca. 6 stabilisiert. Die Treffpunkte werden von Monat zu Monat abgesprochen. Durch die relativ geringen Teilnehmerzahlen hat sich dieses Verfahren bewährt und soll auch in Zukunft beibehalten werden.

Wort	Stackrelation	Erläuterung
OPEN	(filename attri/user -- handle/error<0)	Öffnet eine Datei und liefert handle. filename — Counted-string mit Filenamen handle — Filekennung error — Fehlerkennummer <0
CLOSE	(handle -- error)	Schließt ein File error<0, wenn CLOSE fehlgeschlagen.
MAKE	(filename attri/user -- handle/error<0)	Erzeugt ein File mit Länge 0 (und öffnet es).
DELETE	(filename -- error<0)	Löscht ein File.
READ	(addr count handle -- act_count/error<0)	Liest aus dem File mit der Kennung handle.
WRITE	(addr count handle -- act_count/error<0)	Schreibt in ein File mit der Kennung handle.
SEEK	(offset handle -- act_offset/error<0)	Setzt Filezeiger auf offset-Bytes ab Dateianfang. Trick: -1 SEEK -> Dateilänge
ATTRIBUTES	(filename attri/user -- error<0)	Setzt Fileattribute.
GET_PARAMS	(filename puffer -- puffer)	Schreibt die Dateiattribute in den vorgesehenen Puffer.
LOAD	(handle --)	Lädt ein geöffnetes File.
Erweiterungen		
FLOAD	(filename --)	Öffnet, lädt und schließt ein File : FLOAD (read_only) OPEN DUP LOAD CLOSE ;
FLENGTH	(handle -- lenght)	Liefert die Filelänge : FLENGTH -1 SEEK ;

Tabelle 1

Gegenwart

Das Spektrum der Teilnehmer an den Treffen reicht vom 'ewigen Anfänger' (Selbstbetitelung Jörg Staben) über FORTH-Freaks bis zu Leuten, die FORTH beruflich einsetzen. Bei dieser Bandbreite ist es schwierig, einen Konsens für gemeinsame Projekte zu finden. Um so schöner, daß es uns gelungen ist.

Das gemeinsame Projekt 'Streamfile-Interface'.

Dieses Projekt entstand aus einer breit angelegten Unzufriedenheit mit den Bearbeitungsmöglichkeiten solcher Files in den meisten FORTH-Systemen, wenn sie überhaupt vorhanden sind. Versuche, Files zu öffnen, einige Bytes zu lesen oder zu schreiben schlugen fehl oder waren erst nach längeren Forschungen erfolgreich, aber nicht befriedigend.

Deshalb wurde von der Gruppe ein Satz von Worten festgelegt, der die einfache Benutzung von Streamfiles ermöglichen und vereinheitlichen soll. Dieser Satz ist also als Empfehlung für *alle* FORTH-Implementationen gedacht. Die aufgeführten Worte sind unabhängig von Hardware und FORTH-Implementation. Sie gelten also für 16-Bit-Systeme ebenso wie für 32-Bit-Systeme. Fileoperationen sind somit absolut portabel. Die einzelnen Worte findet man in Tabelle 1.

Bei der Diskussion über eine Implementation gehen die Meinungen in der Gruppe auseinander. Die einen denken daran, ein solches Fileinterface auf der Basis der Blockfunktionen zu implementieren. Dies erscheint insbesondere dann sinnvoll, wenn auf dem Zielsystem unter FORTH kein DOS verfügbar ist, was insbesondere auf kleinen Rechnern (Apple II etc.) häufig der Fall ist. Hier können die Schreib- und Leseprimitive (meist R/W) zur Implementation der benötigten DOS-Funktionen dienen. Das Streamfile-Interface soll also auf das Block-Interface aufsetzen.

Das andere Lager, also meist diejenigen mit 'moderneren' Rechnern mit DOS, wollen lieber das vorhandene DOS ins FORTH abbilden. Die Blockfunktionen sollen dann als Übergangskrücke in den Streamfiles emuliert werden. Das Streamfile soll die 'natürliche' Massenspeicherschnittstelle bereits des Kernels sein.

Sicher ist die zweite Möglichkeit wesentlich einfacher auf den passenden Rechnern zu implementieren, da die Hauptarbeit vom Betriebssystem erledigt werden kann. Die erste Möglichkeit ist aufwendiger, aber bis auf die R/W-Primitive möglicherweise portierbar, da das komplette Filesystem im FORTH verwirklicht ist. Andererseits kann ein solchermaßen portiertes Fileinterface unverträglich mit einem evtl. vorhandenen Betriebssystem sein, so daß hier eigene Plattenpartitionen u.a. angelegt werden müssen, die dann vom DOS aus unzugänglich sind.

Die Diskussion ist sicher noch nicht beendet ...

Um jedoch einen Grundstein für die Akzeptierung der vorgeschlagenen Worte durch die Anwender zu legen, ist eine Implementation für das volksFORTH im Gespräch. Die Implementation soll sowohl die R/W-Primitive nutzen, als auch auf dem Filesystem des Rechners aufliegen. Sobald das geschehen ist, werden wir uns wieder melden.

Soweit also der Bericht aus dem Ruhrgebiet. Am Ende möchte ich nun alle FORTHler aus unserem Gebiet einladen, sich mit mir in Verbindung zu setzen, sei es, um Ort und Zeitpunkt des nächsten Treffens zu erfahren, oder auch nur, um etwas über FORTH zu plaudern und Kontakte zu knüpfen:

Jörg Plewe
Großenbaumerstr. 27
4330 Mülheim/Ruhr
0208/423514

FORTH und FORTH-Prozessoren an der Technischen Akademie Esslingen

Vom 11. bis 13. Dezember 1989 fand an der Technischen Akademie Esslingen unter der Leitung von Dr. Konrad Koller (Zentralabteilung Forschung und Entwicklung der SIEMENS AG) ein Lehrgang über "FORTH und FORTH-Prozessoren" statt. 20 Herren aus technischen Bereichen der Privatwirtschaft nutzten die Möglichkeit, sich grundlegend über Eigenschaften und Einsatzvorteile der Programmierumgebung FORTH und der daraus abgeleiteten Hochsprachenprozessoren zu informieren. Jedem Teilnehmer wurde das FORTH-System F83 zur Verfügung gestellt, so daß die Möglichkeit zum praktischen Arbeiten mit FORTH sowohl im ausgezeichnet ausgestatteten EDV-Labor der Akademie als auch nach der Rückkehr aus dem Lehrgang gegeben war. In zwei zusätzlichen Veranstaltungen stellten die Herren Klaus Flesch und Dirk Brühl moderne Entwicklungssysteme bzw. Boards vor, die auf FORTH beruhen.

Die Raumknappheit der Technischen Akademie brachte es mit sich, daß für den Lehrgang einschließlich Übungen nur drei Tage zur Verfügung standen. Bei der Fülle des Stoffes, der sowohl Software-Entwicklung als auch Hardware-Anwendung umfaßte, wäre mehr Zeit - besonders für die Übungen - wünschenswert gewesen. Der Wissensstand der Lehrgangsbesucher war sehr unterschiedlich, einige Teilnehmer hatten bereits weitgehende berufliche Erfahrungen mit FORTH und nahmen hauptsächlich wegen der Nutzung der FORTH-Prozessoren am Lehrgang teil. Aus diesen Gründen wird der nächste Lehrgang zu den Themen FORTH und FORTH-Prozessoren an der Technischen Akademie Esslingen eine Woche dauern und in zwei Tage für Anfänger und drei Tage für Fortgeschrittene aufgeteilt sein. Der Besuch der Veranstaltung ist sowohl für jeden Teil wie auch für den ganzen Lehrgang möglich. Der Termin ist 28.1. bis 1.2.1991. Die Teilnehmeranzahl ist begrenzt.

Technische Akademie Esslingen,
Postfach 1269,
7302 Ostfildern,
Tel. (0711) 340080

Kleinanzeige

NOVIX EB1 Board mit NC4000 Chip. Nicht benutzt, mit org. Unterlagen von Flesch, dem Buch von Ting (Footsteps...) und der Software alles wie neu, kpl. für DM 750,--. Tel. 06151/710983

Erlebnisbericht: FORTH-Implementation auf Großcomputer in 2 Wochen

von Max Strobel,
Arnswalderstr. 25,
2000 Hamburg 73

An einem EDV-Bildungsinstitut wurde im Rahmen von Assembler-Programmiertraining und Systemprogrammierung ein FORTH-Interpreter auf dem IBM-Großsystem erstellt. Schon nach 10 Tagen kam der Prompt 'OK'. IBM-Großcomputer sind von der Maschinensprache her für FORTH nicht ausgesprochen geeignet (z.B. kein Hardware-Stack). In dieser Systemumgebung kann FORTH aber als eine sehr schnelle Sprache gelten. Trotz geringer Vorkenntnisse und Erfahrung der Programmierer stand eine FORTH-Basisversion in 2 Wochen.

Rahmenbedingungen

An einem großen, rein kommerziellen EDV-Bildungsinstitut ergab sich im Rahmen der Ausbildung zum Systemprogrammierer (Umschulungsmaßnahme, Dauer 50 Wochen Vollzeit) ein kleines FORTH-Projekt. Klein war es in mehrfacher Hinsicht. Erstens bestand die Gruppe nur aus 4 Personen, der Rest der Klasse bearbeitete Sortieralgorithmen, einen Basic-Interpreter und ein kommerzielles Batch-Programm. Zweitens standen für das Projekt nur 2 Wochen zur Verfügung; dabei war täglich von 8-10 Uhr theoretischer Unterricht in IBM/370-Makrosprache! Drittens wurde das Kleinprojekt von der Institutsleitung nicht unterstützt, ganz im Gegenteil!

Es war also ein kleines Projekt mit dem Ziel, unsere frisch erworbenen Assembler-Künste an einer vollständigen Aufgabe zu erproben.

Der IBM-Großcomputer

Da vermutlich nur wenige Leser Erfahrung mit Großcomputern haben, möchte ich einige allgemein interessante Dinge erwähnen.

Die Maschinensprache dieser Systeme, bekannt als IBM/370-Assembler, wurde ca. 1964 in ihrem Format festgelegt. Diese Befehlsformate beschreiben den Aufbau der Maschinenkonstruktionen genau nach OpCode und Operanden, die auch genau so gelernt werden. Einige andere Großcomputer (u.a. Siemens) bedienen sich des gleichen Maschinencodes. Die Ausführung zur Laufzeit in der CPU (hier natürlich kein Chip, sondern eine 'Prozessoreinheit'!) wird von Mikrocode gesteuert. Bei einigen komplexen Funktionen sind das wahre Mikro-Programme. Dieser Mikrocode wird beim Kaltstart von einem speziellen Massenspeicher geladen! Diese Idee war seinerzeit revolutionär, da erstmals Objektcode-Kompatibilität innerhalb einer ganzen Computerfamilie möglich war.

Der Assemblerprogrammierer hat in dieser Sprache 15 32-Bit-Register zur Verfügung, davon sind etwa 6 für Systemzwecke ganz oder teilweise belegt. Adressen werden auch in diesen Registern gehalten, dann sind nur 24 Bit signifikant (bei MVS/XA 31 Bit). Diese Adressen stellen nur 'virtuelle' Adressen dar: jedes Programm hat seinen eigenen 16 MB großen virtuellen Adreßraum. Diese Technik hat heute ja zunehmend Einzug in die Mikrocomputerwelt gefunden.

Der Maschinensprache selbst ist ihr Alter anzumerken. Speicheroperanden werden, um den Code verschieblich zu halten, mit einem Basisregister adressiert. Dabei kann aber nur ein Bereich von 4 KB erfaßt werden, da das Displacement-Feld im entsprechenden Instruktionsformat nur 12 Bit zur Verfügung stellt! Das Verringern von Registerwerten ist mit Inline- (immediate-) Operanden nicht möglich, was vor allem bei der Emulation der beiden Stacks gestört hat. Dafür bietet die Maschinensprache sehr komfortable Befehle zum Rechnen mit gepackten Dezimalzahlen, das unterstützt jedenfalls die Hauptanwendung 'kommerzielle Datenverarbeitung' (Cobol!). Natürlich lassen sich die meisten Beschränkungen irgendwie umgehen, optimal ist das aber nicht.

Stichworte

» FORTH-Implementation
» IBM-Großcomputer

Das Arbeiten unter MVS

MVS heißt 'multiple virtual storage' jedes Programm hat seinen eigenen Adreßraum. Das MVS-System ist so hochgradig virtuell, daß auf der gleichen Maschine meist mehrere Betriebssysteme gefahren werden, die ineinander verschachtelt sein können. Das zur Darstellung dieser Zusammenhänge beliebte Schalenmodell erreicht da eine beeindruckende Dicke. Interessant ist, daß jedes System - auch Dialogsysteme - als Batch-Job gestartet wird.

Unsere Arbeitsumgebung war das Einbenutzer-Betriebssystem CMS, die Verbindung zu MVS bestand im Abschicken von Batch-Jobs. Das CMS erlaubt eigentlich ein relativ komfortables Editieren-Kompilieren-Linken, vor allem die Debugging-Unterstützung ist relativ gut (CMS ist auch relativ neu). Leider wird am Großrechner das Kompilieren und Ausführen von Programmen im Vordergrund oft allgemein verboten . es heißt, das belaste die Antwortzeiten für die anderen Benutzer sehr stark. Deshalb war unsere Arbeitsweise folgende: Das Assembler-Quellprogramm wurde im CMS geschrieben. Dann wurde es zusammen mit einer Eingabedatei als Standardeingabe an das MVS geschickt. Nach 1-15 Minuten kam das Assemblerlisting mit einem allgemeinen Protokoll (Job Log) und der Ausgabedatei an uns zurück. War bei der Programmausführung ein Fehler aufgetreten, kam zusätzlich ein Speicherauszug als Hex-Dump. Trotz aller Widrigkeiten (Warten!), stand nach 10 Tagen in der Standardausgabe die begehrte FORTH-Meldung: 'OK'!

Da wir als Programmierer z.zt. sehr schnell lernen, läuft das FORTH jetzt, mehrere Wochen später, interaktiv (!) im CMS. Natürlich ist das System noch nicht fehlerfrei, aber wir wissen jetzt genau, wo und wie Erweiterungen durchgeführt werden können. Wir würden das auch gerne - hobbymäßig - machen, aber wir können das System nicht mitnehmen, wenn wir unsere Ausbildung beendet haben.

Einzelheiten der FORTH-Implementation

Es war naheliegend, die Registerbreite auszunutzen und ein 32-Bit-System zu implementieren. Als Vorlage diente der gute alte, vor allem aber einfache FIG-FORTH-Standard von 1979 als Assemblerlisting für den C 64! Sehr gute Dienste leisteten auch die beiden Bücher von R. Zech. Ich möchte nicht behaupten den 79-Standard hätten wir erreicht, man kann aber leicht die Schlüsselworte erkennen. Die IBM/370-Maschinensprache ist für FORTH nicht ausdrücklich geeignet. So gibt es keinen Hardwarestack, und auch die eigentümliche Adressierung von Programm und Daten in 4K-Blocks über je 1 Register machte anfangs Probleme. Zur Erleichertung wurden bei der Programmerstellung 12 Makros eingesetzt, die viele Vorteile bringen:

- bessere Schreibbarkeit,
- bessere Lesbarkeit,
- leichte Änderbarkeit (Optimierung).

So werden die typischen Funktionen Push-Pop und der Aufbau der Namensheader im Dictionary von Makros erledigt. Diese Technik kann man genausogut auf dem PC einsetzen (verschiedene Speichermodelle).

Der Innere Interpreter benutzt die typische indirekte Adressierung, wie in Listing 1 zu sehen ist.

Diese Anweisungen generieren 14 Bytes Code, manche Mikroprozessoren brauchen wesentlich weniger. Wer die Programmierung in der Groß-EDV kennt, dem wird die Idee, die Bytes zu zählen, etwas abwegig vorkommen, weiß doch nur der Systemprogrammierer, durch wie viele Interpretationsebenen selbst gängigste Standardprozeduren laufen müssen! Unter diesem Gesichtspunkt ist FORTH auf dem Großcomputer auf jeden Fall rasend schnell.

Unser FORTH/370-Assemblerlisting ist nur als ein Vorschlag zu einer Implementation zu verstehen. Wir lernen selbst täglich dazu; inzwischen wissen wir, wie ein vollkommen multitasking-festes System zu erstellen ist (echtes Code-Sharing), man könnte auch einige Systemaufrufe (supervisor calls) implementieren, um das ganze MVS mit seinen antiquierten Dateifunktionen interaktiv richtig in den Griff zu kriegen. Leider bestehen wenig Chancen, dies in der festgefügten Welt der kommerziellen Groß-EDV zu verwirklichen.

Wer an dem Assembler-Listing für unser FORTH/370 interessiert ist, kann es gerne von mir erhalten.

```
NEXT     L     W,0(IP)     lade das W-Register mit dem Wert,
                           auf den IP zeigt,
         ADD4  IP          Makro: zeige mit IP auf das
                           nächste Feld
                           (generiert 2 Byte Code),
NEXT1    L     CA,0(W)     lade den Zeiger auf den Code,
         ADD4              zeige mit W auf den Body des
                           gerufenen Wortes
         BR    CA          und springe zur Adresse im
                           CA-Register, fertig!
```

Listing 1

Die nächste 'Vierte Dimension' erscheint im Juni '90

Testbericht: FFORTH aus heimischen Landen

von M. Schultheis

Stichworte

- schnelles Atari ST FORTH,
- M68000-Code pur,
- kleiner Kern (ca. 13kB),
- gute S/W Grafik,
- Unterstützt PAK-68020/68881

Wer seinen Atari effektiv programmieren will erhält hier ein geeignetes Werkzeug für schnelle Programme, dessen Autor weitere Unterstüzung und einen Update-Service anbietet.

Aus heimischen Landen frisch für den Atari ST stellt sich das FFORTH vor. Eine Programmierumgebung von Jörg Plewe, der als Autor den Lesern der 'Vierten Dimension' schon bekannt ist. Ein kurzer Einblick weckt sicher das Interesse an diesem System.

FFORTH, das zu 90% dem FORTH-83 Standard entspricht, ist zu allererst schnell. Es erzeugt M68000-Code pur, durch Makros und Unterprogrammaufrufe. Applikationen in FFORTH sind um den Faktor 3 bis 5 schneller als im klassisch gefädelten 32-Bit-FORTH.

Der Maschinencode hat aber seinen Preis. Der Decompiler kann im Bereich vom Makros nur als Disassembler arbeiten. Die Analyse von Systemworten verlangt daher ein grundlegendes Verständnis der Maschinensprache und des TOS. Ausgehend von einem 13kB großen FFORTH-Kern, der als Assemblerprodukt optimiert ist, kann jede gewünschte Arbeitsumgebung oder Applikation hergestellt werden. Als Basis dafür stehen 300kB dokumentierter FFORTH-Quelltext zur Verfügung:

- Assembler mit Kontrollstrukturen in gewohnter Postfixnotation.
- Floatpointpaket mit wahlweiser PAK-68020/68881 Unterstüzung.
- Alle wichtigen TOS-System-Aufrufe
- Floatpoint-gestützte S/W Grafik
- Grafische Benutzeroberfläche
- Debug Option mit Trace, Breakpoints und FORTH als Werkzeug
- Freie Anbindung beliebiger Editoren

Die Laufzeiten von FFORTH-Applikationen sind mit C-Programmen durchaus vergleichbar, und der Speicherbedarf neigt sich bei zunehmendem Umfang zu Gunsten von FFORTH.

Zum Kennenlernen gibt es ein Demo-System, das als vollwertiges FORTH einsetzbar ist. Es fehlen nur der kleine Kern, einige Dienstprogramme und der Quellcode für die Erweiterungen, die schon zum Teil in das Demo kompiliert sind. Den Vertrieb von FFORTH hat die Firma GALACTIC in Essen übernommen.

Das Erscheinen Atari-ST-kompatibler VME-Bus-Karten und ähnlicher Profi-Hardware mit Coprozessoren eröffnen einen weiteren Markt. Bei diesem heimischen Produkt besteht vielleicht die Chance, daß es mit PEARL-Realzeitelementen angereichert, an RTOS angebunden, zu einer Bereicherung der Szene beiträgt. Die Verfechter der "Reinen Lehre" mögen solch ketzerische Visionen mit Fassung ertragen oder unterstützen.

Hinweise für Autoren

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

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

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

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

Low-Cost-Automatisierung mit FORTH

FORTH-Anwendungen in der DDR

von K.Kabitzsch, Technische Hochschule Leipzig
FORTSCHRITT-electronic
Automatisierungsbetrieb im Kombinat Landmaschinen

Hardware

Sinkende Hardwarekosten eröffnen auch Chancen zur Automatisierung kleiner und kleinster Objekte. Das nachfolgend beschriebene Konzept wurde als gerätetechnische Basis für Bordcomputer und funktionsintegrierte Automatisierungssysteme in mobilen Landmaschinen und stationären landtechnischen Anlagen entwickelt. Vergleichbare Einsatzfelder gibt es aber auch in mobilen und stationären Maschinen für den Hoch-, Tief- und Straßenbau, Nutzfahrzzeugen, Laborgeräten, Klimaschränken, preiswerten Maschinen für Kleingewerbe bzw. mittelständische Betriebe in Branchen wie Polygrafie, Nahrungsgüterwirtschaft, Bäckereien usw.. Die dort vorherrschende Kosten-Stückzahl-Situation legt ein Hardwarekonzept mit folgenden Eigenschaften nahe:

- kleinste, autark arbeitsfähige Baugruppen (95 x 170 mm^2, Single-Chip-Controller Z8, 24 kByte Speicher, Peripherie zum Prozeß)
- kostengünstige Erweiterungskonzepte, um aus diesen Baugruppen auch größere Systeme konfigurieren zu können (schneller Parallelbus, Dual-Port-RAM, Low-Cost-Lokalnetz)

Besondere Bedeutung hat das Low-Cost-Lokalnetz (in der Automatisierungsbranche auch als Feldbus bezeichnet), welches nach einem Token-Bus-Prinzip mehrere dieser kleinen Rechnerkerne durch eine Zweidrahtleitung zu einem Rechnernetz verbinden kann. Bisher sind folgende Baugruppen realisiert:

- *BM Basismodul*: Einchipmikrorechner, 24 kByte, Feldbus, Prozeßperipherie
- *AES Analogeingabe*: 7 Kanäle, Verstärkung und Abgleich programmierbar, Ströme, Spannungen und Widerstände sind meßbar
- *BR Binär- Ein-/Ausgabe (Relais)*: 8 optoentkoppelte Eingänge, 8 Relais-Ausgänge
- *BT Binär- Ein-/Ausgabe (Transistor)*: 8 optoentkoppelte Eingänge, 8 Transistor-Ausgangsstufen
- *TAB Tastatur für Basismodul*: 15 Tasten, BCD-kodiert
- *DIB Display für Basismodul*: 8-stell. Siebensegmentanzeige, 4 LED
- *FUE Funktionsüberwachung*: überwacht Spannungen, Temperaturen, Watch-Dog
- *SV Stromversorgung*: 5V, 24V, +15V, -15V, Sonderspannung

Derzeit befinden sich weitere Baugruppen in Entwicklung, z.B. Analogeingabe für Dehnmeßstreifen, Dual-Port-RAM, Schnittstelle für seriellen Binär-Zubringer, Speichererweiterung usw.

FORTH-Programmierung

Die Programmierung der Steuerungshardware erfolgt in PROCESS-FORTH, einer um viele automatisierungstechnische Spezialfunktionen erweiterten FORTH-Version. Dazu enthält ein EPROM des Basismoduls das nach Kundenwünschen zusammengestellte Laufzeitsystem. Zur Anwenderprogrammierung existiert ein Entwicklungssystem mit Editor, Compiler, EPROM-Programmierung usw., der erzeugte Fadencode wird auf ein weiteres EPROM des Basismoduls abgelegt. Nutzt man eine zusätzliche Speichererweiterungskarte, so stehen für Programme und Daten insgesamt 60 kByte EPROM und 60 kByte RAM zur Verfügung.

Die besonderen Anforderungen der Automatisierungstechnik haben einige spezielle Eigenschaften von PROCESS-FORTH zur Folge:

Stichworte

- » Single-Chip-Controller,
- » Steuerung,
- » Regelung,
- » analoge Meßwerte,
- » Automatik

- Der Wortschatz ist fast vollständig primarisiert, d.h. hinter allen Worten stehen schnelle Assemblerroutinen.
- Der übliche FORTH-Standard-Wortschatz belegt nur einen geringen Teil des Laufzeitsystems. Ergänzungs-Primaries mit spezifischen Automatisierungsfunktionen sind inzwischen in der Überzahl und haben zu einem Gesamtumfang von maximal etwa 14 kByte geführt. Das Laufzeitsystem wird deshalb nach Kundenwünschen generiert und enthält danach nur noch jene Komponenten, die im konkreten Fall wirklich gebraucht werden.
- PROCESS-FORTH residiert auf einem voll unterbrechbaren Multitask-Betriebssystem mit verdrängender Bedienungsdisziplin und Festprioritäten für jede Task. Dementsprechend umfaßt der Wortschatz auch System- und Zeitdienste sowie Dienste zur Rechner-Rechner-Kommunikation über das Low-Cost-LAN.
- Viele der generierbaren bzw. als Screens nachladbaren Fachsprachkomponenten orientieren sich an etablierten Vorbildern aus dem Automatisierungsbereich und erscheinen daher auf den ersten Blick *FORTH-untypisch*.

Einige dieser Zusatzkomponenten werden im folgenden vorgestellt.

Bezeichnung	Speicherbedarf (in KByte)	Prinzip der Einbindung
Grundsystem	1,2	Grundkern
Systemdienste	0,4	generierbar
Zeitdienste	0,5	generierbar
Gleitkommapaket	1,1	generierbar
16-Bit-Festkommapaket	1,7	generierbar
32-Bit-Festkommapaket	1,5	generierbar
Feldbus-Dienste	1,0	generierbar
Leittechnik-Paket	1,0	generierbar
Datenbank-Grundpaket	0,5	generierbar
Analogperipherie	0,6	generierbar
Binärwertverarbeitung	0,4	generierbar
Kontaktpläne	1,0	generierbar
Funktionsbaustein-Grundelemente	0,4	generierbar
regelungstechnische Funktionsbausteine	3,5	nachladbar
Datenbankfunktionen	4,5	nachladbar
regelbasierte Funktionen	-	-

Tabelle 1: Komponenten des PROCESS-FORTH-Betriebssystem- Fachsprachkonzepts

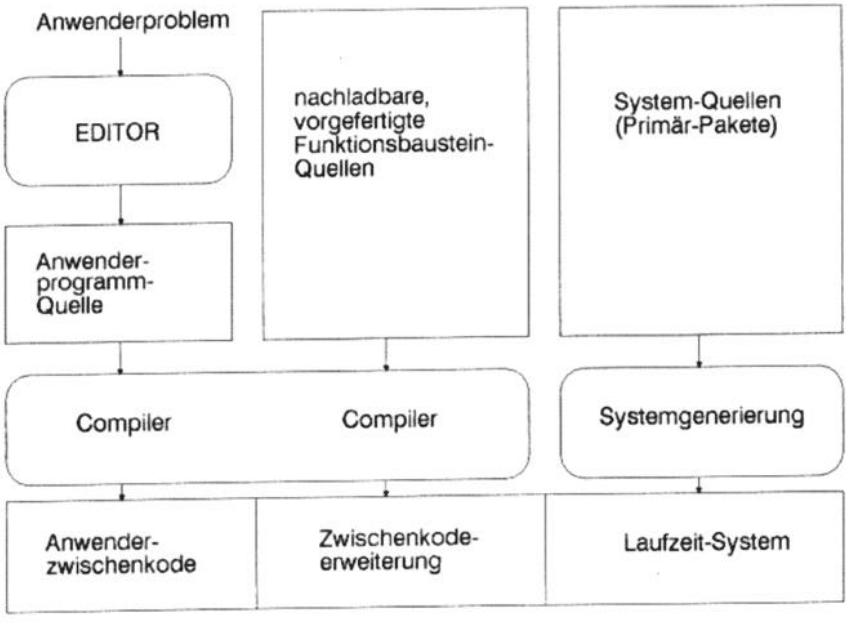


Bild 1: Entstehungsprozeß eines PROCESS-FORTH-Softwareproduktes beim Anwender

Ansprechen der Analogperipherie

Noch relativ FORTH-typisch sehen die Konstrukte zur Erfassung analoger Meßwerte aus.

Beim Ansprechen von Analogperipherie wird der Anwender auf der FORTH-Oberfläche nur mit den Daten und Sachverhalten konfrontiert, die er zur Funktionsfestlegung innerhalb zweckmäßig gewählter Freiheitsgrade benötigt. Alle weiteren Details (Adressen, Steuerworte, Meß- und Korrekturabläufe usw.) werden von Compiler und Laufzeitsystem selbsttätig behandelt (Geheimnisprinzip). Der Programmierer abstrahiert von diesen Einzelheiten, er programmiert eine abstrakte, durch die Semantik der Programmiersprache beschriebene Maschine, die der Compiler auf die reale Maschine abbilden muß. Dazu benötigt er natürlich eine Beschreibung der realen Maschine, welche in Gestalt spezieller Datenstrukturen in die Sprache einzuführen ist. Als zentrale Kategorie für die Formulierung einer Meßaufgabe wird dazu für jede Meßstelle unter einem technologisch sinnvollen Namen eine entsprechende Datenstruktur definiert, z.B.:

```
ANA.MODULE1
  (Baugruppe)
ANA.MUX5
  (Multiplexkanal)
ANA.RANGE32
  (Verstärkung)
MEASURES
BEHAELTERTEMPERATUR
  (Meßstellen-Name)
```

Diese Datenstrukturen und die auf ihr definierten Operationen sind also Software-Strukturelemente, welche die Verbindung zwischen dem physikalischen Prozeßinterface und dem Programm herstellen:

```
MEM =
(Offsetkorrekturwert,
Verstärkungskorrekturwert)

DESCR = (REFER, ANAVER,
ANAMUX, ADANMUX, ADANVER,
ADANSTI, ADANEHI, ADANELO,
ADOFFSET, ADENDW)
```

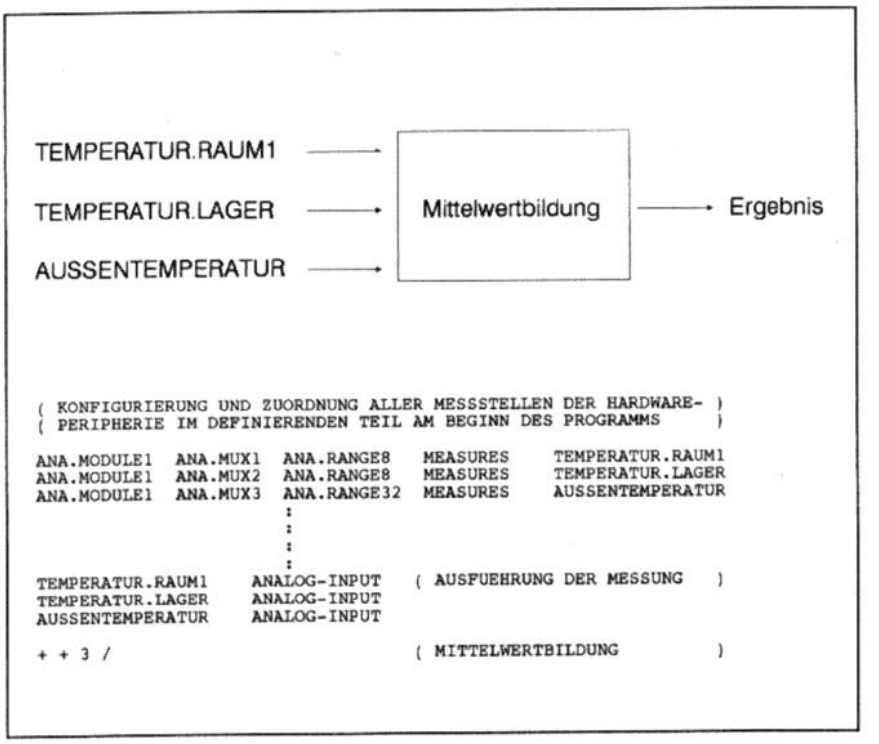
Alle zu dieser Datenstruktur definierten Operationen erhalten durch Voranstellen des Meßstellennamens ihren Bezug zur jeweiligen Meßstelle (z.B. `BEHAELTERTEMPERATUR ANALOG-INPUT`):

- `ANALOG-INPUT` einlesen eines Meßwertes einschließlich Offset-Korrektur
- `OFFSET-VALUE` messen und speichern der Offset-Korrektur-Konstanten
- `RANGE-VALUE` messen und speichern des Verstärkungs-Korrekturfaktors
- `RANGE-CORRECTION` Verstärkungskorrektur

Die gegenwärtige Hardwareausstattung der Baugruppen (Multiplexer) erfordert ihre Behandlung als exklusives Betriebsmittel. Über die Elemente des Gleitkommapakets hinaus sind grundlegende FORTH-Worte zur Meßwert-Vorverarbeitung realisiert (lineare Skalierung, Vorlast, Totzone, Analogschalter, Integrator, T1-Glied usw.).

Binärperipherie und Binärwertverarbeitung

Auch beim Ansprechen binärer Ein- Ausgabeperipherie ist das Geheimnisprinzip verwirklicht. Jede binäre Ein- bzw. Ausgangsklemme wird einzeln durch einen technologisch sinnfälligen Namen referiert, wozu auf FORTH-Niveau über den Compiler folgende Datenstruktur definiert wird:




```
( KONFIGURIERUNG UND ZUORDNUNG ALLER MESSSTELLEN DER HARDWARE- )
( PERIPHERIE IM DEFINIERENDEN TEIL AM BEGINN DES PROGRAMMS     )

ANA.MODULE1  ANA.MUX1  ANA.RANGE8   MEASURES   TEMPERATUR.RAUM1
ANA.MODULE1  ANA.MUX2  ANA.RANGE8   MEASURES   TEMPERATUR.LAGER
ANA.MODULE1  ANA.MUX3  ANA.RANGE32  MEASURES   AUSSENTEMPERATUR
                       :
                       :
                       :
TEMPERATUR.RAUM1    ANALOG-INPUT   ( AUSFUEHRUNG DER MESSUNG   )
TEMPERATUR.LAGER    ANALOG-INPUT
AUSSENTEMPERATUR    ANALOG-INPUT

+ + 3 /                            ( MITTELWERTBILDUNG         )
```

Lsting 1: Beispielprogramm zur Software-Einbindung der Analog Peripherie: Messung von drei Temperaturen, Bildung des Temperaturmittelwertes

```
MEM =   (8 Binärzellen)

DESCR = (REFER, MODAD,
         ANF1, ... ,ANF8)
```

Während das Einlesen aller Binärbaugruppen aus Konsistenzgründen stets gleichzeitig durch einen Strobe-Befehl (`BINARY-STROBE`) erfolgt, ist die Übernahme in den Rechner in 8-Bit-Gruppen organisiert und die dazu vorgesehenen Befehle (`BINARY-INPUT, BINARY-OUTPUT`) nehmen auf einen in der Datenstruktur definierten Gruppennamen (im Beispiel Listing 2: `EINGANGSSIGNALE`) bezug.

Zur Verarbeitung binärer Signale, z.B. im Sinne kombinatorischer Logik, ist das Standard-FORTH nur unzureichend ausgelegt. Dort tritt der Datentyp *Boolean* nur in Form des Flags auf. Auf Grund der Bedeutung derartiger Probleme in der Automatisierungstechnik muß PROCESS-FORTH in diesem Sinne ergänzt werden. Auf Grund der unbeschränkten Funktionenvielfalt in der Binärsignalverarbeitung stellen hier aber Softwarebausteine höherer Komplexität ebenfalls keine akzeptable Lösung dar, so daß auf freie Programmierbarkeit unter Nutzung elementarer Funktionen orientiert wurde. Aus diesen Gründen sind die geschaffenen FORTH-Ergänzungskomponenten konsequent als einfache Primaries realisiert. Sie nutzen die Tatsache aus, daß Stackmaschinen zur Berechnung von Ausdrücken nach Vorrangregeln (z.B. Klammern) von Haus aus prädestiniert sind. Die vorgeschlagene Lösung realisiert die Semantik der in der Fertigungsautomatisierung dominierenden Anweisungslisten /1/2/ auf FORTH-typische Weise, wobei der Bitakkumulator durch den bedeutend tieferen Stack ersetzt wird. Obwohl vom Standpunkt der FORTH-Philosophie Klammerbefehle entbehrlich wären, wurden sie aus Kompatibilitätsgründen zusätzlich realisiert. Der Zugriff des Nutzers auf die Binärvariable erfolgt auf komfortable und selbstdokumentierende Weise durch den Aufruf anwendungsbezogen definierter Variablennamen. Durch das Hinzufügen mehrerer direkt auf Bitvariablen wirkender Zeitwerksbefehle (z.B. Einschalt- und Ausschaltverzögerung) wird der semantische Umfang üblicher Anweisungslisten-Sprachen durch PROCESS-FORTH voll abgedeckt. Die Darstellung von Binärvariablen auf dem Stack wurde bewußt identisch zur Flag-Darstellung gehalten. Damit ist das Zusammenwirken logischer Verknüpfungen mit

Verarbeitungskonstrukten (> , = , usw.) und Steuerkonstrukten (IF...ELSE...ENDIF usw.) sichergestellt sowie die Nutzbarkeit aller Stackmanipulationsbefehle auch für Binärvariable garantiert. Als generierbares Paket wurde hierzu realisiert:

```
BCONSTANT       BINARY.MODULE
EABYTE-END      EABYTE-DEFINITION
BVARIABLE       BITVAREA
L               LN
=N              U
UN              O
ON              XO
XON             SE
RE              U[
UN[             O[
ON[             ].
N.              O.
U.              BINARY-STROBE
BINARY-INPUT    BINARY-OUTPUT
TA              TE
TI              TR
:=
```

Der in Bild 2 dargestellte Kontaktplan /1/2/ zeigt exemplarisch, daß unter FORTH in eingeschränkter Weise auch quasigrafische Problemnotationen möglich sind. Die realisierten Ergänzungen in Compiler und Laufzeitsystem enthalten auch Kontaktplan-spezifische Syntaxprüfungen, Zeitwerke (Ein- bzw. Ausschaltverzögerung), Zählbausteine usw. und sind als generierbares Paket ausgeführt:

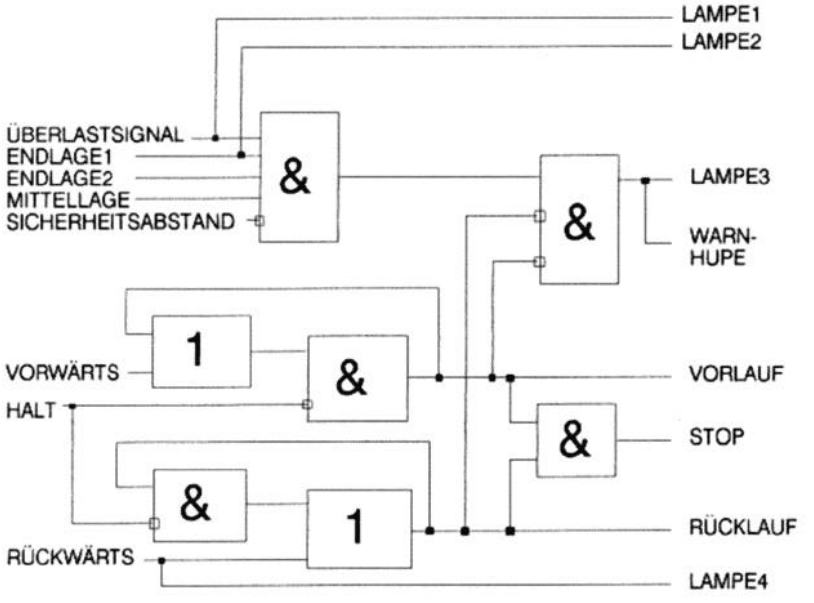
```
---         |         ][-
]/[         -()-|     (()-|
()-|        (R)       (S)
+e-         -e*       e-
--e         $--       %--
ZAEHLER     ZV        ZR
SET         RES       ZERO
```

Die Kontaktplan-Programmierung weist zwar kaum noch Verwandtschaft zur FORTH-Philosophie auf, ermöglicht es jedoch auch absoluten DV-Laien, Steuerungsprobleme zu analysieren, Lösungen zu formulieren und in lauffähige Programme umzusetzen. Durch ihre Verwandtschaft zu den Stromlaufplänen der Starkstromtechnik werden Kontaktpläne weltweit vor allem von Servicehandwerkern, Betriebselektrikern usw. bevorzugt.

Komplexe Funktionsbausteine

Im Gegensatz zu den sogenannten Funktionen sind Funktionsbausteine durch folgende Merkmale charakterisiert:

```
( KONFIGURIERUNG UND ZUORDNUNG ALLER ANSCHLUSSKLEMMEN DER      )
( HARDWARE-PERIPHERIE IM DEFINIERENDEN TEIL AM BEGINN DES      )
( PROGRAMMS                                                    )

BINARY.MODULE1      EABYTE-DEFINITION       EINGANGSSIGNALE
      BITVAREA      UEBERLASTSIGNAL
      BITVAREA      ENDLAGE1
      BITVAREA      ENDLAGE2
      BITVAREA      MITTELLAGE
      BITVAREA      SICHERHEITSABSTAND
      BITVAREA      VORWAERTS
      BITVAREA      HALT
      BITVAREA      RUECKWAERTS               EABYTE-END

BINARY.MODULE1      EABYTE-DEFINITION       AUSGANGSSIGNALE
      BITVAREA      LAMPE1
      BITVAREA      LAMPE2
      BITVAREA      LAMPE3
      BITVAREA      WARNHUPE
      BITVAREA      VORLAUF
      BITVAREA      STOP
      BITVAREA      RUECKLAUF
      BITVAREA      LAMPE4                    EABYTE-END

( EIN-/AUSGABE U. AUSFUEHRUNG DER BINAEREN VERARBEITUNGSAUFGABE)

BINARY-STROBE                               ( EINLESEN VON BINAER- )
EINGANGSSIGNALE     BINARY-INPUT            ( E/A - BAUGRUPPE      )

UEBERLASTSIGNAL     L
LAMPE1              C!
ENDLAGE1            L
LAMPE2              C!
RUECKWAERTS         L
LAMPE4              C!
UEBERLASTSIGNAL     L
ENDLAGE1            U
ENDLAGE2            U
MITTELLAGE          U
SICHERHEITSABSTAND  UN                      ( STACK=ZWISCHENERGEBN.)
VORLAUF             L
VORWAERTS           O
HALT                UN
VORLAUF             :=
                    N.                      ( STACK=ZWISCHENERGEBN.)
RUECKLAUF           L
HALT                UN
RUECKWAERTS         O
RUECKLAUF           :=
                    N.                      ( STACK=ZWISCHENERGEBN.)
                    U.
                    U.
LAMPE3              :=
WARNHUPE            :=
                    DROP
VORLAUF             L
RUECKLAUF           U
STOP                :=
                    DROP
AUSGANGSSIGNALE     BINARY-OUTPUT           ( AUSGABE AN BINAER-   )
                                            ( EA - BAUGRUPPE       )
```



Listing 2: Beispiel-Programm zum PROCESS-FORTH-Paket für Binärsignalverarbeitung (Anweisungslisten-Logik)

- Sie besitzen mehr als einen Ausgangsoperanden (mehrere Eingangsoperanden sind auch bei Funktionen erlaubt).
- Sie verlangen den Austausch einer insgesamt hohen Zahl von Eingangs- und Ausgangsoperanden.
- Ihre Ausgangsoperanden sind nicht nur von den Eingangsoperanden, sondern auch vom inneren Zustand der Bausteine abhängig. Wird also derselbe Baustein mehrmals mit den gleichen Eingangsoperanden aufgerufen, so kann er jeweils verschiedene Ausgangsoperanden liefern. Diese Eigenschaft wird bausteinintern durch die Führung von Datenspeichern entsprechenden Typs (binär, analog, Zeichen usw.) implementiert.

Die interne Implementation komplexer Funktionsbausteine ist unter anderem dadurch gekennzeichnet, daß eine wachsende Anzahl verschiedener Daten über wachsende Befehlsdistanzen lebendig bleibt. Sollen diese auf dem Stack aufbewahrt werden, so wächst der Aufwand für Stackmanipulationen überproportional, weshalb Effizienzverluste nur durch Benutzung adressierbarer RAM-Zellen vermeidbar sind.

Dabei fordert das Geheimnisprinzip, daß diese nur der internen Realisierung dienenden RAM-Zellen dem Nutzer verborgen bleiben. Dieser darf demnach nicht explizit in die Adreßvergabe einbezogen sein, letztere muß vielmehr vom Compiler selbsttätig gelöst werden. Interne RAM-Zellen sollten vielmehr vor dem (unbeabsichtigten) Zugriff des Nutzers zumindest insoweit geschützt werden, daß dieser sie nicht durch Namensaufruf (reference by name) erreichen kann.

Sieht eine Echtzeit-Sprache anwendungsorientierte Funktionsbausteine (z.B. Regler) vor, so kann der Nutzer mit Recht fordern, diese Bausteine in seinem Programm beliebig oft und in beliebigem zeitlichem Zusammenhang aufzurufen. Die Gewährleistung dieser Eigenschaft führt zu folgenden Problemen:

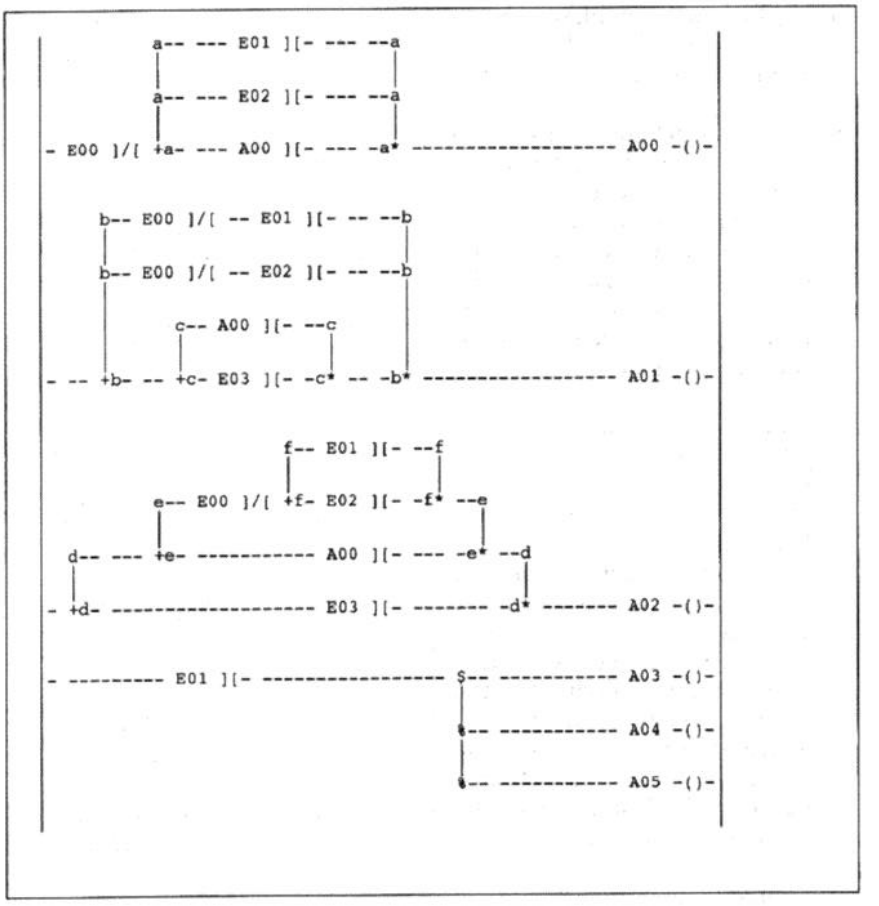


Bild 2: PORCESS-FORTH-Kontaktplan

flüchtige Variable

Befinden sich in einem Multitasksystem mehr als ein Baustein gleichen Typs zwischen Anfang und Ende der Bearbeitung und nutzen beide gleiche RAM-Zellen für flüchtige Variable, so kann nicht ausgeschlossen werden, daß ein Schreibzugriff des Bausteins A ein noch lebendiges Datum des Bausteins B vorzeitig zerstört. Hier sind folgende Lösungen denkbar /3/:

- Vergabe des Bausteins als exklusives Betriebsmittel (Prozedurverwaltung): Ein Task wird blokkiert, wenn es einen bereits arbeitenden Baustein nutzen will und darf erst fortschreiten, wenn dieser Baustein beendet ist.
- Vergabe des Speicherraumes als exklusives Betriebsmittel, d.h. der aufgerufene Baustein darf zwar starten, wird aber blockiert, wenn er auf den kritischen RAM-Bereich zugreift.
- Einbeziehung der RAM-Zellen in den automatischen Kontextwechsel bei Taskumschaltung.
- Vergabe von disjunktem RAM für jeden Baustein gleichen Typs.

Die ersten beiden Lösungen fordern hohen Verwaltungsaufwand zur Laufzeit. Die dritte ist nur für wenige RAM-Zellen (z.B. Registersatz) praktikabel. Die letzte benötigt mehrfachen RAM und verlangt vom Programmierer die verantwortliche Mitwirkung bei der RAM-Vergabe. Beide Nachteile entfallen, wenn der RAM-Bedarf der Bausteine gering ist und die disjunkte RAM-Vergabe dem Compiler übertragen wird.

Zustandsvariable:

Benutzen zwei Bausteine gleichen Typs gleiche Speicherzellen zur Führung von Zustandsvariablen, so sind Konflikte auch bei Vermeidung von Parallelarbeit (Sequentialisierung durch Prozedur- oder Speicherverwaltung zur Laufzeit) unvermeidbar, da Zustandsvariable auch nach Ende der Bausteinabarbeitung lebendig bleiben. Hier bleibt die Vergabe von disjunktem Speicher als einzige Möglichkeit.

Beim Entwurf komplexer Funktionsbausteine werden daher folgende Grundcharakteristika realisiert:

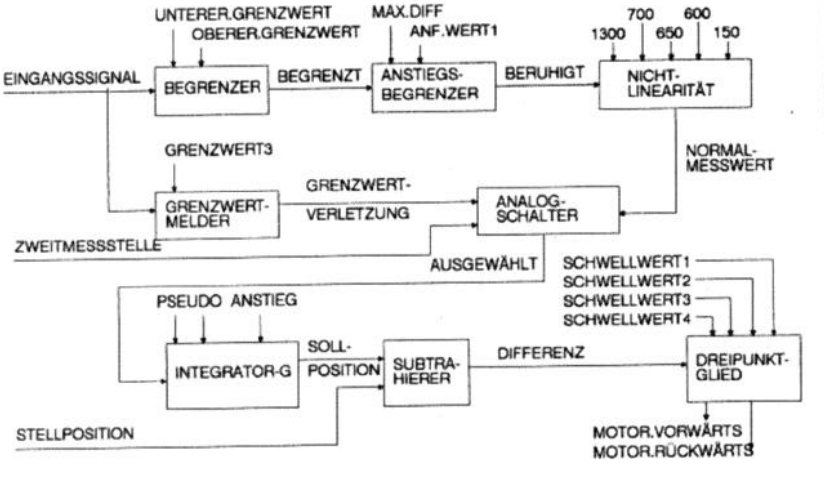
- Jeder Aufruf eines Bausteins führt zur Compilezeit zur Reservierung des notwendigen, objektorientierten RAM-Bereichs.
- Die Adreßreferenz dieses Bereiches wird, vor dem Anwender verborgen, zur Laufzeit der zum Baustein gehörenden Laufzeitroutine vermittelt.
- Die Speicherzugriffe der Laufzeitroutine erfolgen relativ zu dieser Adreßreferenz. Dazu wurde ein Satz schneller Zugriffs-Primaries entwickelt.
- Der Austausch von Eingabe- und Ausgabeoperanden erfolgt über adressierbare RAM-Zellen, die Übergabe der zugehörigen Referenzen geschieht über den Stack in einer ergonomisch günstigen Form (`INPUT:`, `OUTPUT:`, `BUILD-BLOCK:`). Zahl und Reihenfolge der Referenzen werden vom Compiler bausteinspezifisch überprüft.

Das vorgestellte Baustein-Konzept wird vorwiegend zur Schaffung nachladbarer Bausteinbibliotheken genutzt. Beispiele sind Datenbankfunktionen sowie die folgenden Bausteine

Der Autor

Dr.-Ing. Klaus Kabitzsch studierte von 1972 bis 1976 Informationstechnik, Fachrichtung Höchstfrequenztechnik, an der TH Ilmenau. Anschließend bearbeitete er dort als Assistent Probleme des rechnergestützten Entwurfes von Schaltungen und akustischen Oberflächenwellenbauelementen. Ab 1981 war er im VEB FORTSCHRITT-electronic Automatisierungstechnik Leipzig auf dem Gebiet der Geräteentwicklung für die Meßwerterfassung, Steuerung und Regelung tätig und beschäftigte sich insbesondere mit der Softwareentwicklung für Automatisierungssysteme. Seit 1989 arbeitet er als Oberassistent an der TH Leipzig, Sektion Automatisierungsanlagen, mit den Schwerpunkten Rechnerkommunikation, verteilte Algorithmik, Betriebssysteme und Sprachen.

```
OUTPUT01:       GRENZWERT-VERLETZUNG
INPUT01:        EINGANGSSIGNAL
INPUT02:        GRENZWERT3
BUILD-BLOCK:    X22 GRENZWERTMELDER

OUTPUT01:       BEGRENZT
INPUT01:        UNTERER.GRENZWERT
INPUT02:        OBERER.GRENZWERT
INPUT03:        EINGANGSSIGNAL
BUILD-BLOCK:    X11 BEGRENZER

OUTPUT01:       BERUHIGT
INPUT01:        BEGRENZT
INPUT02:        MAX.DIFF
INPUT03:        ANF.WERT1
BUILD-BLOCK:    X01 ANSTIEGSBEGRENZER

OUTPUT01:       NORMALMESSWERT
INPUT01:        BERUHIGT
INPUT02:        1300  S-F
INPUT03:         700  S-F
INPUT04:         650  S-F
INPUT05:         600  S-F
INPUT06:         150  S-F
BUILD-BLOCK:    X09 NICHTLINEARITAET

OUTPUT01:       AUSGEWAEHLT
INPUT01:        NORMALMESSWERT
INPUT02:        ZWEITMESSSTELLE
INPUT03:        GRENZWERT-VERLETZUNG
BUILD-BLOCK:    X21 ANALOGSCHALTER

OUTPUT01:       SOLLPOSITION
INPUT01:        AUSGEWAEHLT
INPUT02:        PSEUDO
INPUT03:        PSEUDO
INPUT04:        ANSTIEG
BUILD-BLOCK:    X13 INTEGRATOR-G

OUTPUT01:       DIFFERENZ
INPUT01:        SOLLPOSITION
INPUT02:        STELLPOSITION
BUILD-BLOCK:    X12 SUBTRAHIERER

OUTPUT01:       MOTOR.VORWAERTS
OUTPUT02:       MOTOR.RUECKWAERTS
INPUT01:        SCHWELLWERT1
INPUT02:        SCHWELLWERT2
INPUT03:        SCHWELLWERT3
INPUT04:        SCHWELLWERT4
INPUT05:        DIFFERENZ
BUILD-BLOCK:    X02 DREIPUNKTGLIED
```



Listing 3: Signalflußbild und seine Umsetzung mit PROCESS-FORTH-Funktionsbausteinen

zur Analogsignalverarbeitung, welche sich vor allem an regelungstechnischen Erfordernissen /3/ orientieren:

```
PID-GLIED              ZEITPLANSTEUERUNG
DREIPUNKTGLIED         ANSTIEGSBEGRENZER
LAUFZEITGLIED          GRENZWERTMELDER
VORLAST                TOTZONE
SKALIERUNG             ANALOGSCHALTER
DYNAMIK                ADDIERER
SUBTRAHIERER           MULTIPLIZIERER
DIVIDIERER             INVERTER
BETRAG                 MISCHSTELLE
NICHTLINEARITAET       BEGRENZER
MITTELWERT             IMPULSVERZOEGERUNG
INTEGRATOR-G
EINGANGSSIGNALVERARBEITUNG
MINIMUMAUSWAHL         MAXIMUMAUSWAHL
AUSGANGSSIGNALVERARBEITUNG
POLYNOM
```

Diese Bausteine entnehmen dem RAM definierte Eingangsoperanden (`INPUT`), verarbeiten sie und legen die Ergebnisoperanden wieder im Speicher ab (`OUTPUT`). Eine Verknüpfung von Bausteinen erfolgt also durch die namentliche Festlegung solcher Übergabezellen in einer zum Bausteinaufruf (`BUILD-BLOCK`) gehörenden `INPUT-OUTPUT`-Liste (Listing 3), deren Bedeutung in der jeweiligen Bausteinspezifikation /4/ definiert ist.

Kontakt: Dr. Klaus Kabitzsch
Gustav-Adolf-Str. 14
DDR-4854 Lützen
Tel. Leipzig, 3943-184

Quellenangaben:

/1/ Berger,H.: Programmieren von Funktionsbausteinen in STEP-5, Siemens-AG Berlin/München, 1985

/2/ SIMATIC S5, Kataloge ST50 ... ST59, Siemens-AG, 1987

/3/ Kabitzsch,K.: Mikrorechner in der Automatisierunspraxis, Akademie-Verlag Berlin, 1987

/4/ Kabitzsch,K.: Sprachbeschreibung PROCESS-FORTH, Version 1.2, TH Leipzig, Sektion Automatisierungsanlagen, 1990

Der Mensch als Jäger und Sammler

Testbericht: POLYMATH und FIFTH

von Jörg Staben / Hilden

Stichworte

» POLYMATH
» FIFTH

Alle Jahre wieder werden die Tage kurz und die Abende lang - die Sommerzeit ist zu Ende. Da wird gerne die Gelegenheit genutzt, endlich einmal Ordnung zu schaffen, denn man wird nur zu deutlich daran erinnert, daß der Mensch im tiefen Inneren ein Jäger und Sammler geblieben ist.

Gesammelt werden nicht nur Unterlagen, die man eh' nie liest, sondern auch Disketten, Backups, Backbakkups und darunter sind dann auch die Sachen, zu denen man doch schon immer mal ...

Das Ergebnis meiner wiedererwachten Ordnungsliebe sind zwei Vorstellungen von FORTH-ähnlichen Programmierumgebungen, beide 1986 fertiggestellt. Vielleicht wird die eine oder andere noch gepflegt oder weiter verbreitet, verdient hätten es beide.

Der Sampler zum POLYMATH könnte in den Vertrieb der FORTH-Gesellschaft aufgenommen werden, das FIFTH befindet sich bereits dort, so daß Interessierte sich dahin wenden können.

POLYMATH V1.0 - ein FORTH-programmierbarer HP-Rechner?

POLYMATH ist eine bewußt einfach gehaltene Programmierumgebung, in der man einfach, schnell und mit Freude seine Probleme interaktiv lösen soll. Der Autor Greg Lobser hat POLYMATH mit einer Polaroid-Kamera verglichen: Sicher gibt es kompliziertere und aufwendigere Systeme, die dann auch bessere Ergebnisse liefern - aber zuerst einmal leistet POLYMATH sofort genau das, was man braucht.

Im Prinzip hat man einen wissenschaftlichen Rechner vor sich, der grundsätzlich mit Fließkomma-Arithmetik arbeitet. An Mathematik ist von SQRT über die Winkelfunktionen bis zu den Logarithmen alles eingebaut, was man so braucht, einschließlich der Umwandlung integer ↔ float.

Darüber hinaus finden sich hier eine Reihe von sympathischen Ideen; so verfügt POLYMATH über zwei Stacks: Zahlenstack und Stringstack.

Beide Stacks werden mit dem Prompt permanent angezeigt, die Zahlen mit frei wählbaren Nachkommastellen und die Strings im Klartext.

Eine Anforderung in POLYMATH könnte wirklich so aussehen:

```
1.0000 Hallo Welt
3.1415 FORTH hält frisch
```

Irgendwo ist natürlich auch das Prompt (ein >) dabei. Die üblichen Stackmanipulationen wie `DUP`, `SWAP`, `OVER` sind mit beiden Stacks möglich.

Zum Arbeiten mit Stacks gehört unweigerlich das *Markenzeichen* von FORTH, die UPN. Nicht so in POLYMATH - hier kann wahlweise präfix oder postfix gerechnet werden! `PRINTING(1)` ist genauso erlaubt wie

```
3 3 * 4 4 * + SQRT
```

wobei dieser Ausdruck auch als

```
SQRT((3*3)+(4*4))
```

akzeptiert wird.

Beim Umgang mit Daten wurde ebenfalls nachgedacht und einiges an Vorarbeit geleistet. So ist einschließlich der FP-Matrix alles vorhanden, was andere Programmierumgebungen auch anbieten. Variable werden, je nach Syntax, initialisiert:

```
Variable X    \ enthält 0
1 Variable Y  \ enthält 1
```

Variablen legen sofort - ohne `@` - ihren Wert auf den Stack; der Preis dafür ist die merkwürdige Zuweisung an Variable:

```
2 --> X
```

Aber manche Programmiersprache verlangt da sogar ein `:=` ...

Für die Quelltexte benutzt POLYMATH ganz normale MSDOS-Textfiles, ein Editor wird im PD-Sampler nicht mitgeliefert. Kompilierte Anwendungen können auch in kompilierter Form als PACKAGES abgespeichert werden, ein HELP-File kann einer Anwendung zugeordnet werden. Damit ist ein `EXPLAIN`

<word> möglich. Dadurch umfaßt eine Anwendung so wie hier immer drei Dateien:

Application: Complex Numbers	
Complex.pm	Text source file for the application
Complex.hlp	Random access help file
Complex.pkg	Dictionary for the application
Application: Polynomial Regression Analysis	
Regress.pm	Text source file for the application
Regress.hlp	Random access help file
Regress.pkg	Dictionary for the application

Wie die Themen der Beispiele zeigen, ist POLYMATH stark mathematisch orientiert. Dennoch zeigt die kleine Einführung GUIDE die Fähigkeiten von POLYMATH zur Textbearbeitung und Textgestaltung. Es ist schön, hier unter den Beispielen die bekannte Anwendung der Rekursion *Türme von Hanoi* (The Towers of Babble) wiederzufinden. An Hand des abgedruckten Codefragmentes kann man einen Eindruck davon bekommen, welche Syntax auch durch die präpostfix-Notation möglich ist. `INVISIBLE` deklariert Daten und Prozeduren lokal, `MYSELF` ist in der FORTH-Welt hinreichend bekannt - manche nennen es `RECURSE` (siehe Listing 1).

Abschließend eine Antwort auf die Frage "Ist das FORTH?" und einen Ausblick auf weitere Projekte: Nach den Maßstäben von Charles Moore (Modularität, implizite Parameterübergabe) ist POLYMATH ein FORTH-System. Allerdings nimmt sich G. Lobser die Freiheit, eine Sprache parallel zu FORTH zu entwikkeln, um mögliche Lösungen zu den bekannten Diskussionspunkten in FORTH anzubieten:

- File-Interface (Text vs. Screens)
- lokale Variablen, die nach bester volksFORTH-Manier mit {} angelegt werden
- Vererbungsorientierte Programmierung (OOPS)
- Komfort gegenüber Geschwindigkeit

Die Vollversion unterstützt den 80(x)87 CoProzessor, so daß eine Zahl durch 8 Byte statt durch 6 Byte dargestellt wird und umfaßt die oben angesprochenen Merkmale.

Vor drei Jahren konnte man die Vollversion von POLYMATH für $40 hier kaufen - in Anbetracht der verstrichenen Zeit dürfte sich zuerst ein Telefonat in die Staaten lohnen:

Greg Lobser
5534 W. Alder Ave.
Littleton, Colorado 80123
phone: (303) 973-1028

FIFTH V2.2 - eine Evolution in FORTH?

FIFTH stellt eine interaktive Programm-Entwicklungsumgebung auf der Basis von FORTH dar und verdankt seinen Namen der Tatsache, daß seine Entwickler zählen können. Die Version 2.2 ist ein shareware Produkt aus dem Jahr 1987 und läuft auf dem IBM PC und dem TI-PC; eine Version für den 68000 mit Multitasking und Virtual Memory (Code, Objekte und Daten größer als der Hauptspeicher) war in Arbeit. Ob diese 68000-Version inzwischen - 1989 - fertiggestellt wurde, ist mir nicht bekannt.

```
Polymath Definitions

\  ----------  Variables  ----------

   3 Matrix   Post       invisible
10 3 Matrix   Disk       invisible
     Variable here       invisible
     Variable there      invisible

 0.5 variable left       invisible
 3.5 variable right      invisible

\  ----------  Plotting Words  ----------

: DrawPosts
  color(2)
  Move(0,0) Draw(right,0)
  Do(1,3)
     If(i=there) Color(3) else Color(2) endif
     Move(i,0) Draw(i,9)
  Loop; invisible

: NextPost
  there mod(3) +1 --> there
  here  mod(3) +1 --> here
  DrawPosts; invisible

\  ---------- Elicit a response from the user ----------

: Query
  Off
  Message('How many disks?')
  Prompt
  If(dup=0) stop(not) endif
  Min(9) --> N
  1      --> here
  2      --> there; invisible

\  ----- Recursive solution of the Towers of Babble
------

: MovePile \ here there n ---
  If(dup=1) drop MoveDisk
  Else
     1-
     Myself(    HereMiddle   )
     MoveDisk(  HereThere    )
     Myself(    MiddleThere  )
  endif; invisible
```

Listing 1

Die Programmierumgebung von FIFTH besteht aus - natürlich - dem Compiler, einem Wordstar-ähnlichen Editor, einem Browser (??) und einer interaktiven Umgebung, die über zwei interessante Kommandos verfügt: `HELP` und `DIR`. `HELP` arbeitet, wie man es inzwischen gewohnt ist: Es erklärt das betreffende Wort und zeigt die Parameter für den Aufruf an.

Der `DIR`-Befehl macht den eigentliche Reiz von FIFTH aus. Er aktiviert einen menü-gesteuerten Editor für das Wörterbuch (Dictionary)! Nun kommt der Browser für das Durchblättern des Wörterbuches zum Einsatz: Mit Hilfe der Cursor-Tasten kann das einer Baum-Struktur ähnliche Wörterbuch durchblättert und geändert(!) werden. Dennoch ist, nach Druck auf die ESC-Taste, der von FORTH gewohnte interaktive Modus verfügbar.

Aus dem DIRectory heraus werden Dateien geladen, abgespeichert, editiert und kompiliert.

FIFTH verfügt über ein Konzept der bedingten inkrementellen Kompilierung; soll ein noch nicht kompiliertes Wort ausgeführt werden, so wird es automatisch kompiliert. Ebenso erzwingt das Verlassen des Editors das Kompilieren, so daß der Programmierer in den meisten Fällen nicht ausdrücklich zu kompilieren braucht. Im Fehlerfall meldet sich FIFTH mit einer invers unterlegten Box; sollte die Programmausführung nicht die gewünschten Ergebnisse zeigen, steht ein kleiner Debugger zur Verfügung, der das gerade ausgeführte Wort und den Stack anzeigt. Das Wort `TRACE`, das interaktiv den Debug-Modus ein- und ausschaltet, wirkt im Programmtext als Breakpoint. Der TRACE-Modus wirkt lokal auf eine Prozedur, damit fehlerfreie Programmteile nicht unnötig durchlaufen werden.

Der Compiler soll schneller als Turbo-PASCAL sein, wobei in den meisten Fällen nur noch der gerade editierte Text kompiliert wird, so daß die Kompilation kaum spürbar ist.

FIFTH benutzt statt der (früher?) FORTH-üblichen Screens & Blocks die MSDOS-Dateistruktur. FIFTH beherrscht Inline-Assembler-Code, 32-Bit-Arithmetik, Fließkommazahlen, automatische 8087-Unterstützung, direct-threaded-Code für eine hohe Ausführunggeschwindigkeit und den Zugriff auf den gesamten Speicherbereich des Rechners (keine 64KB Grenze!).

Ein kleines Code-Beispiel in Form eines Interruptaufrufs ist in Listing 2 enthalten. Wobei es hier wahrscheinlich elegantere Formen gibt, aber es ist wohl eine sehr flexible Form. Dagegen macht das Wort `STACK` all' denen Freude, die nicht so gerne auf dem Stack herumjonglieren.

Die 32-Bit-Arithmetik führt allerdings bei Standarddefinitionen wie `COMMA (,)` zu Schwierigkeiten, da sich alle Berechnungen - so auch in Datenfeldern - auf 4 Byte beziehen. 16-Bit-Operationen müssen entsprechend gekennzeichnet werden (`W,`).

FIFTH macht auch Grafik - was man halt auf dem IBM so Grafik nennt, wenn auf einer Hercules-Karte ein CGA-Emulator läuft. Daher bin ich über einige Quadrate auf dem Bildschirm nie hinausgekommen und konnte die *Towers of Hanoi* nicht genießen.

Aber der Hauptunterschied FIFTH zu FORTH sind deutlich eingeschränktere und damit interessantere Geltungbereiche für Namen als deren globale Gültigkeit im Standard-FORTH. In diesen Sichtbarkeitsregeln (scoping) und in dem neuen Konzept zur Modulverwaltung steckt meiner Ansicht nach gerade der Pfiff dieses FORTH-Abkömmlings.

Was ist ein *Modul* in FIFTH? Es entspricht einem *Wort* in FORTH, einer *Prozedur* in PASCAL oder einem *Unterprogramm* in BASIC. Ein Modul besteht aus vier Teilen:

- dem Namen - das ist normal,
- dem ausführbaren Code und den zugehörigen Daten - für ein direct-threaded-System auch normal,
- dem Quelltext - das ist schon ungewöhnlicher - und
- vor allem - das ist neu - einem Unterwörterbuch, das weitere Unter-Module enthält, die für die Definition des höheren Moduls gebraucht werden.

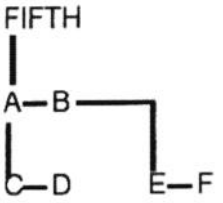
Eingebunden sind diese Module in die Wörterbuchstruktur von FIFTH; hier finden sich sowohl Primitives (die Systemroutinen) als auch die vom Anwender programmierten Module. Jedes Modul kann weitere Module oder Unterwörterbücher (Subdictionaries) besitzen, gehört aber seinerseits ebenfalls zu einem (höheren) Modul. Dies setzt sich fort bis hin zum ROOT-Modul, wobei ein Modul mit einem Subdictionary ein *lokales* Modul für diese Unterwörterbücher ist.

Damit sind auch die Geltungsbereiche der Module festgelegt: Nur das lokale Modul kann auf seine Untermodule und Unterwörterbücher zugreifen. So legt die Position eines Moduls im Wörterbuch auch seinen Geltungsbereich fest: Je höher es sich in der Hierarchie befindet, desto größer ist sein Gültigkeitsbereich.

```
CREATE TIME                                          \ Returns the time in hundreths
                                                     \ of a second.
16 base !
: time
\ es  ds    si  di    dx  cx    bx  ax   int
00000000  00000000  00000000  00002C00 21  int       \ gettime.
stack abcde|c                                        \ get rid of bogusness
[ decimal ]
dup 16 shr swap 16 shl +                             \ Yields time in hour.min.sec.100th
```

Listing 2

Als Beispiel diene folgendes Diagramm:



Ein neues Untermodul in F könnte auf E, B, A und ROOT zugreifen - D und C wären nicht sichtbar, die darin enthaltenen Definitionen nicht erreichbar. C dagegen kann nicht auch auf D zugreifen, sondern nur auf A und ROOT ! Die Suche nach definierten Modulen geht also immer nach links und nach oben. Existieren mehrere Module gleichen Namens, so wird das unterste der Suchreihenfolge benutzt.

Um den Geltungsbereich von Modulen zu ändern, kann deren Position im Wörterbuch verändert werden, indem die Cursortasten benutzt werden!! Es ist wirklich faszinierend zu sehen, wie man ein Modul mitsamt Daten und Quellcode einfach nach vorne verschiebt, um es in einer früheren Definition zu verwenden. Der Moment, den FIFTH zum Neukompilieren der geänderten Module braucht, ist wirklich unmerklich. Es ist auch möglich, Module zu löschen oder an beliebiger Stelle ins Wörterbuch einzufügen. Zusätzlich ist durch den Browser auch gewährleistet, daß man im Editor nur die aktuelle Prozedur bearbeitet, wobei auch jedes Modul einzeln geladen oder abgespeichert werden kann.

Warum hat FIFTH bei all' den Leistungsmerkmalen nicht schon längst FORTH und TurboPASCAL verdrängt?

Zum einen hatte sich die Shareware-Version von FIFTH 1987, als die ganzen Leistungsmerkmale noch spektakulärer waren, kaum verbreitet. Das Konzept des editierbaren Dictionarys zusammen mit dem Grundkonzept von FORTH wird wohl manchen neugierigen Programmiersprachen-Touristen verwirrt und damit verprellt haben.

Zum anderen fehlen trotz des in vielerlei Hinsicht komfortablen Sprachumfangs grundlegende Worte: So kann - ohne FORTH-Erfahrung mit QUERY WORD CONVERT - keine Zahl eingelesen werden. Auch vermittelt die fehlende Möglichkeit, eine alleinstehende Anwendung zu erstellen, das Gefühl, mit einem reduzierten System zu arbeiten.

Zum dritten findet sich nirgendwo ein Hinweis auf eine vollständigere Version 2.5, die wohl zeitgleich zum Shareware-Produkt V.2.2 kommerziell vertrieben wurde.

Zum guten Schluß schreiben wir fast 1990; TurboPASCAL v5.5 sowie MS Quick-C sind auf dem Markt und bestimmen den Standard für die Bedieneroberfläche, die Benutzerführung und den Leistungsumfang einer Programmierumgebung. Wenn jemand dennoch nachhören möchte, ob und wie FIFTH sich weiterentwickelt hat:

CLICK Software
(409) 696-5432
P.O. Box 10162
College Station, Texas. 77840
(P.S. Die Adresse war Frühsommer'88 noch aktuell.)

Mensch-ärgere-dich-nicht in volksFORTH

von Andreas Döring

Quelltext Service

Das Programm wurde in zwei Module aufgeteilt, um es leichter portabel zu gestalten. Das erste Modul (in der Datei MAEDN.SCR) enthält nur das Spiel ohne Ein-Ausgabefunktionen. Diese finden sich (ST-speziphisch mit GEM) in der zweiten Datei MAEDNH.SCR. Da diese aber benötigt werden, um das erste Modul zu übersetzen, wird es zweigeteilt. Das zweite Modul braucht nämlich einige Worte aus dem Spiel (die zentralen Datenstrukturen). Deshalb ist das erste Modul auch zweigeteilt. Die Shadowscreens beziehen sich darauf. Wenn man beide Dateien abgetippt hat, braucht man noch zwei Dateien: die Resource, die man mit einem RCS erstellt und die Mouseform, die man mit dem SME (Simple Mouse Editor), einem weitverbreiteten Public-Domain-Programm malt. Möchte man darauf verzichten, so sind die entspechenden Definitionen in Screen 2 von MAEDNH.SCR wegzulassen. (Man braucht nur finger, die Zeile mit `+thru`). Das nun fehlende Wort `figure` ersetzt man durch eine Umbenennung: `' finger Alias figure`

Die Objektnummern im zweiten und dritten Screen stammen aus dem .H-file, das vom RCS erzeugt wird. Es gibt zwar ein kleines FORTH-Programm, mit welchem dieses automatisch in FORTH-Quelltext umgesetzt wird. Dies nützt aber nicht viel, da fast alle Objektnummern in Feldern benötigt werden. Ich habe deshalb das H-File gedruckt und dann daraus die Nummern direkt entnommen. Da Ihr Resourcefile die Felder mit anderem Index enthalten wird als meins, müssen Sie das genauso machen. (Welche Reihenfolge dabei gilt, steht im Shadowscreen).

Die größte Arbeit dabei ist die Namensgebung der 72 Felder im RCS. Dieses könnte durchaus mal verbessert werden.

Hier nun noch ein paar Worte zur Übertragbarkeit: Will man das Programm auf einem anderen Rechner als dem ST verwenden, so braucht man dafür ein volksFORTH (es gibt nicht viele Systeme, die Deferred-Words erlauben, deshalb eignet sich nicht jedes FORTH) und die Anpassung an die Ein/Ausgabe. Diese muß ja nicht so komplex sein, wie MAEDNH.SCR; wie es ganz einfach geht, zeigt der 8. Screen von MAEDN.SCR. Dort gehört dann auch noch ein einfacher Zufallszahlengenerator dazu, den es aber praktisch auf jedem Rechner gibt.

Stichworte

- volksFORTH
- Mensch ärgere dich nicht

Die GEM-Anpassung hält sich an die Konventionen, die man in der einschlägigen Literatur findet. Vor allem sind an keiner Stelle auflösungsabhängige Koordinaten verwendet. So wird selbst der Durchmesser der

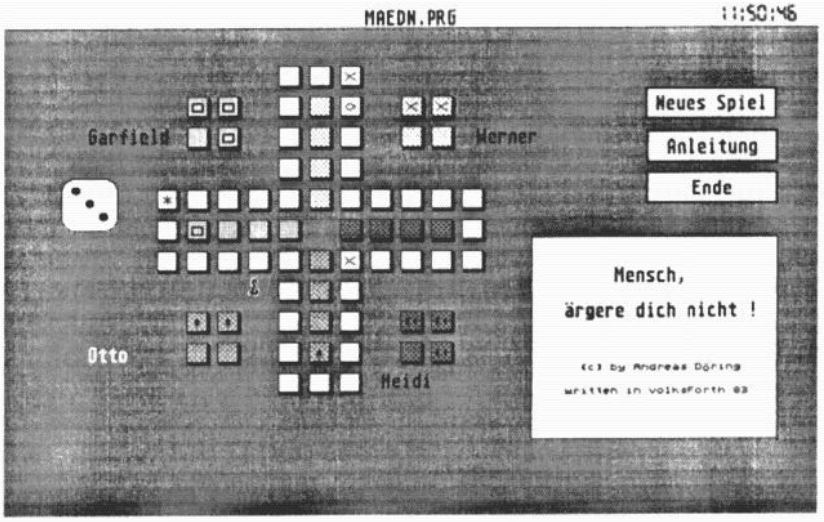


Bild 1: Mensch-ärgere-dich-nicht-Screenshot

Würfelpunkte aus den Koordinaten des Würfels berechnet. Diese aber werden aus der Resource geholt. Da eine Resource aber ziemlich auflösungsunabhängig aufgebaut ist, ist das Programm im Prinzip auflösungsunabhängig. Leider nur im Prinzip, denn zum Beispiel in der Anleitung habe ich den 6*6-Pixelfont verwendet. In den niedrigen Auflösungen laufen nun die Zeilen ineinander. Es ist wohl möglich dies zu vermeiden, aber mir ist das nicht gelungen. Hier ist also noch einiges zu verbessern.

Die Standardroutine `form_do` des GEM habe ich nicht verwendet, da sie immer eine Auswahl eines bestimmten Objektes erfordert, dessen Nummer sie dann liefert. Da beim Würfeln aber nur auf einen Maustastendruck gewartet wird, dieser aber auch zur Auswahl der 3 Knöpfe (Neues Spiel, Anleitung, Ende) verwendet wird, ist es sehr schwer dies mit `form_do` zu erledigen. Ich habe es probiert, und war mit dem Ergebnis nicht zufrieden. Deshalb habe ich dann kurzerhand ein eigenes Wort zur Auswertung der Mausaktionen geschrieben. Dieses verwendet die vom AES dafür vorgesehene Funktion `objc_find`. Diese wird übrigens auch von `form_do` genutzt.

Es hätte noch die Möglichkeit bestanden, den Würfel als eigenes Objekt zu definieren, aber dies ist in Verbindung mit FORTH etwas schwierig, da die Anpassung an die Forderungen des GEM einige Umstände in Assembler erfordert hätten, und das hätte das gesamte Programm etwas unübersichtlich gemacht.

SCR# 0

```
 >>>Mensch ärgere dich nicht<<< (C) by Andreas Döring Jan. 88
 für  volksFORTH 3.80               Melsbacher Str. 25
                                    5450 Neuwied 13

------------------------------------------------------------
Dieses File enthält nur das Spiel selbst ohne Grafik.
Als Schnittstelle zur Grafik und zur Bedienung dienen folgende
 Befehle: show_name  ( -- )   Anzeige des Spielers
          gmove ( from to--)  Grafische Anzeige des Zuges
          ginit ( -- )        Initialisierung der Grafik
          click ( -- )        Warten auf Tastendruck o.Ä.
          get_throw ( --n)    Eingabe eines Zuges;n ist das Feld
                              von dem der Stein weggezogen wird
          dice      ( --w)        Würfel
w=Würfel, n=Feld, p=Spielernummer, i=Index, dn=Differenz
                  zwischen zwei Feldern, f=flag
```

SCR# 1

```
\ Loadscreen npl, game                                  ad27sep87
Create player_natures 8 allot \ Vektoren für 4 Spieler
Defer 'comp      Defer 'human
: set_npl ( f n--) swap IF ['] 'human ELSE
                           ['] 'comp  THEN
                   swap 2* player_natures + ! ;
: get_npl ( n--f) 2* player_natures + @ ['] 'human = ;
1 2 +thru          \ Schnittstelle für GEM
include maednh.scr \ GEM-Einbindung, kann ausmaskiert werden
\needs gmove 8 load \ falls keine GEM-Einbindung vorhanden

 3 6 +thru             \ Spiel
: game  BEGIN player @ 1+ 3 and dup player ! show_name
        player @ 2* player_natures + perform
        ?stack  vic? ( stop? or)  UNTIL ;
\ ?stack kann weggelassen werde, nur am Anfang zum Testen
```

SCR# 2

```
\ Datenstruktur mit Zugriff                             ad16sep87
Variable player

Create men 4 4 * allot \ Spielfiguren
: man@ ( n --m) player @ 2* 2* + men + c@ ;
: man! ( m n--) player @ 2* 2* + men + c! ;
: splr_cd   ( x--x') player @ 2* 2* + ; \ Bei e oder h
: e? ( n--f) $c0 and $80 = ;    \ Elysium?
: >e ( dn--) $80 or splr_cd ;   \ Eingang in die schönen Gefilde
: h? ( n--f) $c0 and $c0 = ;    \ Home?
: >h ( i--)  $c0 or splr_cd ;   \ Leider zurück
: 1h? ( --f) false 4 0 DO I man@ h? or LOOP ; \ Jemand zuhaus?
: vic?   ( --f) true 4 0 DO I man@ e? and LOOP ;
: init 4 0 DO I player !   4 0 DO I dup >h swap man! LOOP
           LOOP 3 player !  ;
```

SCR# 3

```
\ allgemeine Definitionen für Züge
: field ( n--p i true/false) player push
 4 0 DO I player !  4 0 DO dup I man@ =
    IF drop J I true endloop endloop exit THEN
                        LOOP LOOP drop false ;
: start     ( -- n) player @ &10 * ;
: add_d ( w n--n' f) dup e? IF dup 3 and + dup 4 < -rot
                               swap -3 and or swap
     ELSE dup h? IF swap 6 - IF false ELSE drop start true THEN
        ELSE  start - &40 mod +
              dup &39 > IF &40 - dup 3 > IF false exit ELSE
                        >e true THEN ELSE
              start + &40 mod true THEN  THEN THEN  ;
: free?     ( n--f) field dup IF 2drop player @ = THEN not ;
\\ : field men 16 rot scan dup IF men - dup 2/ 2/ swap 3
                                  and rot THEN ;
```

SCR# 9

```
                                                        ad13jan88
```

SCR# 10

```
\ Loadscreen                                              ad8jan
je 1 Wort pro Spieler, entweder 'comp oder 'human, je nach Natur
 Forwärtsreferenzen, um die Besetzungen der Spieler aus dem GEM
  steuern zu können
  Es gilt f=true  ==> player n = 'human
          =false ==> player n = 'comp
    set_nsp setzt, und get_nsp holt den entspechenden Wert
Hier stehen die Definitionen, die das GEM zum Zeigen der Steine,
  Codieren der Felder etc. braucht.
Zum Testen erweist sich das GEM als sehr unpraktisch, deshalb
   liegen in screen 8  TOS-Definitionen, als Ersatz des GEM
   dann wird hier das GEM geladen
GAME bedarf eigentlich keiner Erklärung. Ich mache darauf
     aufmerksam, daß die Berechnung des nächsten Spielers vor
     und nicht nach dem Aufruf des Spielers durchgeführt wird.
     ( stop? or) erlaubt einen Abbruch mit der esc-Taste
```

SCR# 11

```
 Das Prinzip des 'Information hiding' ist gut zu sehen:
aktueller Spieler wird so oft gebraucht, daß es zu kompliziert
  wäre ihn auf dem Stack zu verwahren
Feld mit den Spielfiguren, enthält die Nummer des Feldes
Diese zwei Worte reichen zum Zugriff auf das Feld, damit ist die
  Kenntnis der tatsächlichen Implementation nicht nötig
Fehlte anfangs, was Probleme bei ?throw ergibt
Die folgenden 4 Worte dienen der Kodierung der Felder, so daß
 danach nur darüber zugegriffen werden kann. Dafür ist aber noch
 eine Operation zu definieren, um damit arbeiten zu können.

wird benötigt für die richtige Verarbeitung der 6
Sieg, wenn alle Steine im Elysium angekommen sind
init setzt alle Figuren auf ihre Anfangsplätze, damit als erstes
 Spieler 0 am Zug ist, muß player mit 3 initialisiert werden
  ( siehe 'game' )
```

SCR# 12

```
                                                          ad8jan
field ist der Zugriff auf die Spieler in anderer Richtung.
 Diese Definition ist zwar langsamer, hält sich dafür aber an
 die Konvention ( Verwendet nur man@ ). Sie setzt allerdings
 voraus, daß auf jedem Feld höchstens eine Figur steht.
n ist das Feld, auf das ein Stein aus dem home gesetzt wird
add_d ist die oben erwähnte Operation. n' ist das Feld, auf das
 ein Stein von Platz n kommen würde, wenn er w Schritte vorwärts
 zieht. Dabei wird nicht überprüft, ob n oder n' überhaupt be-
 legt sind.
 Ein Fehler ( false als Ergebnis) liegt vor, wenn das Elysium
 verlassen würde, oder n ein Home-Feld ist, und w keine 6 ist.

prüft, ob der aktuelle Spieler das Feld n betreten kann.
Diese Definition von Field ist zwar viel schneller und kürzer,
 aber unsauber, da sie die Kenntnis über men vorraussetzt.
```

Mensch-ärgere-dich-nicht in volksFORTH

SCR# 4

```
\ n_six?, possible?, ?throw, ?move                          ad16sep87
: n_six? ( --f) true 4 0 DO I man@ dup e? swap h? or and LOOP ;

: possible? ( w--w f) false 4 0 DO over I man@ add_d swap
                      free? and or LOOP ;

: ?throw ( n--f) player push dup free? IF dup field IF swap
                  player ! dup >h  under swap man! gmove
                ELSE drop  THEN true ELSE drop false THEN ;

: ?move  ( w n--w false/true) dup field 0=   IF drop false
    exit  ELSE drop player @ - IF drop false exit THEN THEN
     over 6 = over h? 0= 1h? and and start free? and
              IF drop false ELSE
   2dup add_d over free?  and IF dup ?throw drop 2dup swap field
      drop nip  man! gmove  true ELSE 2drop false THEN THEN ;
```

SCR# 5

```
\  human, out?, move                                        ad16sep87
: human n_six? IF 0  3 0 DO drop click dice dup
      6 = IF leave THEN  LOOP          ELSE click dice THEN
     BEGIN  possible?                                      WHILE
       BEGIN get_throw ?move dup 0= IF bell THEN UNTIL
           6 = dup IF click dice swap THEN not ?dup
     UNTIL drop   ;

' human is 'human

: out? ( --f) 4 0 DO I man@ h? IF I leave THEN LOOP
            start  dup ?throw IF over man@ over gmove
        swap man! true   ELSE 2drop false THEN ;

: move ( w--w) 4 0 DO I man@ ?move IF drop  leave THEN LOOP   ;
```

SCR# 6

```
\ throw?                                                    ad23dec87
: throw? ( w--w f) -1 BEGIN 1+ dup 4 < ?dup 0=
                    IF drop false dup THEN
           WHILE 2dup man@ add_d   \ solange noch nicht alle
            IF dup field           \ Zug möglich?
              IF drop player @ -   \ eigene Figur?
                IF dup ?throw drop swap 2dup man@ swap
                    gmove man! drop true dup \ für UNTIL
                ELSE drop false THEN
              ELSE drop false THEN
             ELSE drop false THEN
           UNTIL ;
```

SCR# 7

```
\ comp,                                                     ad13jan88

: throw_or_move dup  throw? 0= IF move THEN ;



: comp n_six? IF 0 BEGIN 1+ dice dup 6 - WHILE drop
   dup 3 = dup IF nip  THEN ?exit REPEAT nip ELSE dice THEN
    BEGIN possible? dup 0= IF nip THEN          WHILE
    dup 6 = 1h? and  IF out? 0=
       IF throw_or_move THEN   ELSE          throw_or_move  THEN
                  6 = dup IF dice swap THEN not UNTIL ;
' comp Is 'comp
```

SCR# 8

```
\ Ersatz des GEM zum Testen                                 ad13jan88
\needs code include assemble.scr
Code random ( --d) &17 # A7 -) move $e trap 2 A7 addq .l
                   D0   SP -) move Next end-code
: dice random drop 6 mod drop 1+ dup ." wurf: " .  ;
: ginit ;
: show_name cr     ." Spieler: " . ;
: get_throw   ." Ihr Zug bitte: "
    pad dup 1+ 5 expect span @ over c! number drop ;
: click key drop ;
: gmove swap ." von" . ." nach" . ;

false 0 set_nap
true  1 set_nap
false 2 set_nap
false 3 set_nap
```

SCR# 13

```
                                                            ad8jan
flag ist true, wenn kein Stein im Feld ist. ( dann braucht der
  Spieler eine 6 )
flag ist true, wenn der aktuelle Spieler mit dem Wurf w einen
  Zug ausführen könnte.

wenn das Feld n von einem fremden Spieler besetzt ist, wird er
 hinausgeworfen. steht dort allerdings ein eigener Stein, ist
 flag=false

wenn die Figur auf Feld n den Zug w ausführen kann, so geschieht
 das und das Flag ist true, anderenfalls false
 es wird auch überprüft, ob auf dem Feld n überhaupt eine eigene
 Figur steht und ob eventuell der Zug verboten ist, weil der
 Spieler mit einer 6 eine neue Figur auf das Feld setzen kann.
 Damit enthält dieses Wort ( fast) alle Spielregeln
```

SCR# 14

```
HUMAN OUT? MOVE                                             ad13jan88
das ist der Zug eines 'menschlichen' Spielers. Wenn er/sie eine
 6 braucht, wird dreimal gewürfelt.
 Ist ein Zug möglich, wird solange auf die Eingabe eines gülti-
 gen Zuges gewartet.
 wurde eine 6 gewürfelt, so wird noch mal gewürfelt und der
 Spieler ist nochmals dran.

Patchen des deferred word
ab hier sind alle Worte Hilfsdefinitionen für comp
out? setzt falls dies möglich ist einen neuen Stein ins Feld
      f=true; es wird vorausgesetzt, daß vorher 1h? getestet
      wurde und true zurückgab

beliebiger Zug ( hier immer die Figur mit dem niedrigsten Index)
  possible? muß vorher true gewesen sein !
```

SCR# 15

```
THROW?                                                      ad13jan88
schlägt, wenn es irgendwie möglich ist, eine gegnerische Figur:
 Alle vier eigenen Figuren werden nacheinander genommen.
 Wenn sie den Zug ausführen können, und auf dem Zielplatz eine
 gegnerische Figur steht, wird diese mit ?throw geschlagen.

Ein verbesserter Algorithmus für den Computerzug müßte hier
 und bei move eingebaut werden.

Beispielsweise könnte man dafür sorgen, daß man möglichst nicht
 auf ein feindliches Startfeld kommt, keine anderen Figuren
 überholt und z.B. alles tut, um andere am Eingang in das Ely-
 sium zu hindern. (Wie wäre es mit einer kleinen Verfolgungs-
 jagd?)
```

SCR# 16

```
THROW_OR_MOVE COMP                                          ad13jan88

der Name sagt eigentlich alles. Für einen wirklich optimalen
 Spielalgorithmus müßte diese Definiton herhalten. Man könnte
 eine Art gewichteten Wertungsbaum, der die Abstände zum Ziel,
 und zu anderen Figuren und schließlich deren Wert ( also Ab-
 stand vom Elysium) auswertet.

Computerzug: Fast genauso wie human. Nur das der Zug nicht vom
 Spieler erwartet wird, sondern berechnet wird. Dazu wird ge-
 prüft, ob eine neue Figur rausgesetzt werden muß.
 Natürlich kann auch der Computer nochmal ziehen, wenn er eine
 6 hatte
Schließlich wird auch dieses Wort gepatcht. Man könnte ja
 eventuell auch ein anderes Wort nehmen, für Spiele mit der
 dritten Art...
```

SCR# 17

```
\ Ersatz des GEM zum Testen, Hinweise für eigene Graphikeinbind.
XBIOS(17) liefert eine 24-Bit-Zufallszahl, im Standartsystem
 noch nicht enthalten

Umrechnung der Zufallszahl auf 1..6, und Anzeige des Wurfes
Anstelle der Ausgabe mit . kann der Würfel auch gezeichnet wer-
den. Beim C64 könnte man zum Beispiel daran denken, den Zeichen-
satz neu zu definieren, und dabei Symbole für den Würfel vorzu-
sehen.
GINIT hat normalerweise das Spielfeld zu zeichnen. (Abkürzung
 von graphics-initialize
Die Bedeutung von SHOW_NAME und GET_THROW geht aus dem Namen
hervor. Wie viele Möglichkeiten es dafür gibt, läßt maednh.scr
 nur ahnen. Für Computer ohne Maus bietet sich eine Bedienung
 mit Cursortasten oder Joystick an. Die Darstellung der Figuren
 kann zum Beispiel mit Herz, Pik etc. erfolgen.
```

Mensch-ärgere-dich-nicht in volksFORTH

SCR# 0

```
Dieses File enthält das in FORTH umgesetzte H-File zur Resource
 und die gesamte GEM-Einbindung, es wird mit Rasterkoordinaten
 gearbeitet, da die Normalkoordinaten leider ( noch?) nicht ein-
 setzbar sind.
 Viele Teile dieses Files können genauso, oder in leicht abge-
 wandelter Form für eigene Programme verwendet werden. Zum Bei-
 spiel wurde der nächste Screen aus einem anderen Programm über-
 nommen. Es wurde darauf geachtet, daß nur dokumenierte Funk-
 tionen verwendet werden.
 Zum Teil sind die verwendeten Funktionen oder Strukturen in der
 Literatur nachzulesen.
 Besonders interessant dürfte der Würfel sein, da er völlig ohne
 Absolutkoordinaten arbeitet.
 Dies gilt übrigens für das gesamte Programm, so daß es auf auch
 auf späteren ST-kompatiblen Computern auch bei höheren Auflö-
 sungen arbeiten müßte, falls auch dort volksFORTH verfügbar ist
```

SCR# 1

```
\ loadscreen mouse-forms                                  ad17jan88

: finger  3 graf_mouse show_c ;
use maedn.img  dos

Create mouse_form &75 allot
mouse_form &75 handle (fileread drop  maednh.scr


: figure mouse_form >absaddr mofaddr 2! &255 graf_mouse
                                                    show_c ;



1 10 +thru
```

SCR# 2

```
\   zwei Screens enthalten die RSC-Infos                  ad17jan88
: ct Constant does> @ state @ IF [compile] Literal     THEN ;
0   ct screen          2 ct    stat_sp1    3 ct sp_name1
1   ct inputnam        4 ct    stat_sp2    5 ct sp_name2
2   ct instruct        6 ct    stat_sp3    7 ct sp_name3
74  ct Ende            8 ct    stat_sp4    9 ct sp_name4
73  ct Anleitung      10 ct    OK         11 ct cancel
0   ct quit   81  ct sp_n2 82 ct sp_n3 83 ct sp_n1 84 ct sp_n4
80  ct Neues_sp   85 ct dice_f
\ und nun die Spielfelder
| Create fld 28 c, 27 c, 26 c, 25 c, 24 c, 23 c, 22 c,
     21 c,  20 c, 29 c, 19 c, 18 c, 17 c, 16 c, 13 c, 12 c, 7 c,
    6 c, 5 c, 1 c, 3 c, 4 c, 9 c, 10 c, 15 c, 52 c, 51 c, 50 c,
    49 c, 48 c, 43 c,  42 c, 41 c, 40 c,  39 c, 38 c,
    37 c, 36 c, 35 c, 34 c,
```

SCR# 3

```
\ Forts.                                                  ad2jan87
| Create elys 44 c, 45 c, 46 c, 47 c, 30 c, 31 c, 32 c, 33 c,
              2 c,  8 c, 11 c, 14 c, 53 c, 54 c, 55 c, 56 c,

| Create home  65 c, 68 c, 67 c, 66 c, 61 c, 64 c, 63 c, 62 c,
              59 c, 60 c, 58 c, 57 c, 71 c, 70 c, 69 c, 72 c,


| : gfobj ( n--objc) dup h? IF $F and home + ELSE
            dup e? IF $F and elys + ELSE fld + THEN THEN c@ ;

| : gobjf ( objc--n f) >r   fld  dup 40 r@ scan IF swap -
               ELSE 2drop  elys dup 16 r@ scan IF swap - >e
                ELSE 2drop  home dup 16 r@ scan IF swap -
                 3 and    >h ELSE 2drop rdrop false exit
                 THEN THEN THEN rdrop true ;
```

SCR# 4

```
\  GEM's                                                  rd 3
gem also definitions
| Create screen_c 8 allot   \ Bildschirmauflösung
| 2Variable fld_sz  \ Größe eines Einzelfeldes
: init_res 0 4 wind_get intout 2+ 4@ screen_c 4! ;
: init_fds 1 gfobj objc_getwh fld_sz 2! ;
: rsc_load          rsrc_load" maedn.rsc" ;
: init_screen 3 0 0 0 0 screen_c 4@ form_dial ;
: objc_ret ( objc#--)  dup objc_gaddr $A. d+ 1@ -2 and >r
                       screen_c 4@ r>  1 objc_change ;
: show_object2  ( -- )   init_object
 0 little 4@ big 4@ form_dial 1 little 4@  big 4@  form_dial
   0 ( install) 3 ( depth)  big 4@ objc_draw ;

: hide_object2  ( -- )   hide_c  2 little 4@  big 4@
    form_dial 3 little 4@  big 4@ form_dial ;
```

SCR# 13

```
\                                                         ad17jan88
Leider wird für dieses Programm für jede Auflösung eine eigene
 Resource benötigt, obwohl RSCs ja auflösungsunabhängig ge-
 schrieben sind.
 Dies liegt daran, daß die Zeichenausgabe nicht proportional zur
 Auflösung ist. Deshalb liegen zwei Strings, die in der hohen
 Auflösung genau hintereinander liegen, dann plötzlich übereinander. Diese Probleme können erst dann beseitigt werden, wenn
 man entweder auf die höhere Auflösung des sw-Bildes verzichtet
 oder die Auflösungen insgesamt so hoch sind, daß bei Umrechnung kaum störende Ungenauigkeiten entstehen würden, also auch
 nur noch ein Font für alle Auflösungen nötig sind.
Eine Automatik mit zwei RSCs und Prüfung der Auflösung ist nicht
 schwierig und müßte RSC_LOAD (Screen 4) geändert, werden.
 ( Natürlich könnte man irgendwie auch eine universelle RSC
   malen, aber dies würde ich als unsauber empfinden.)
```

SCR# 14

```
\ loadscreen mouse-form                                   ad17jan88

ändert den Maus-zeiger in den Zeigefinger
 Diese Datei wurde mit dem SME.PRG ( Public-Domain)     erzeugt.
 Sie enthält eine neue mouse-form.
Hier soll sie hin.
und hier wird sie geladen.
dann wird auf das alte File zurückgeschaltet.
So wird sie nachher eingeschaltet. show_c ist nur sicherheits-
 halber.
Wieso das 75 Byte und nicht 35 Byte, wie in der Literatur ange-
 geben sind, weiß ich auch nicht, aber das Imagefile hat diese
 Größe, und ich nehme an, daß dort nicht mehr als nötig drin ist
Laden der gesamten GEM-Einbindung
```

SCR# 15

```
Wenn man im RCS die eigene Resource erzeugt hat, dann kann man
 das .H-File umsetzen. Dafür druckt man es, wenn möglich.
Bedeutungen (siehe Text):
3 Bäume: sceen, instruct, inputnam  1 Alert: quit
srceen: 3 Buttons: Ende, Anleitung, Neues_sp
        die Spielfelder (keine Namen, nur object-Nummern)
          insgesamt 72
        4 Strings: sp_n1,...sp_n4
        1 i-box (nicht sichtbar) siehe bei Würfel
input_nam: 4 radio-buttons:    stat_sp1,...stat_sp4
           4 editable strings: sp_name1,...sp_name4
           2 buttons OK, cancel
Anleitung: 1 button, kein Name nötig

Nach fld stehen die Objektnummern so, wie sie der erste Spieler
 durchlaufen würde, wenn er nur Einsen würfelte.
```

SCR# 16

```
Felder des Elysiums: Zuerst die 4 von Spieler 1 (bzw. 0), dann
 von 1 usw., dabei kommen die Äußersten zuerst.

Bei home gilt das Gleiche, nur spielt hier die Reihenfolge der
 Felder eines Spielers untereinander keine Rolle.


Umwandlung von Object- in Feldnummer und umgekehrt.


Durch Verwendung des Stringbefehls 'scan' besonders schnell.
 f ist false, wenn das Object kein Spielfeld war.
```

SCR# 17

```
 Ich danke hier meinem Bruder, der diesen Teil geschrieben hat.
Ab sofort kommen alle Definitionen in das GEM-Vocabulary.
Um das Programm in allen Auflösungen lauffähig zu gestalten.
 Prinzipiell könnte es auch auf ein Fenster ( fester Größe) begr
 enzt werden.
 Größe von Feld 1. Alle Felder sollten gleich groß sein.
Läd die Resource. Falls man den Namen ändern will, dann hier.
Wiederherstellen des grauen Bildschirms.
Invertierte Objecte werden wieder Normalgezeichnet.

grow_box und Zeichnen des Objekts. Sollte bei eigenen Malereien
 etwas fehlen, dann kann das an der geringen Tiefe liegen. Sie
 sollte aber nicht größer als nötib sein, dann geht's schneller.

shrinkbox und Grauzeichnen
```

SCR# 5

```
\ Spielfiguren                                              ad17jan88
| Create  pl_forms 3 c, 4 c, 5 c, 6 c,


| Create  pl_colors 1 c, 2 c, 3 c, 4 c,

| : show_fig ( field--)
    player @ dup pl_forms + c@ sm_type pl_colors + c@ sm_color
                        gfobj objc_offset swap fld_sz @ 2/ +
                        swap fld_sz 2+ @ 2/ +
                        hide_c   1 pmarker figure ;
| : redraw_f ( n--) gfobj 1 screen_c 4@ objc_draw ;

: set_fig's player push 4 0 DO I player !
           4 0 DO I man@ show_fig LOOP  LOOP  ;
```

SCR# 6

```
\ Würfel ... : Anpassung an alle Auflösungen              ad27sep87
\needs code 2 loadfrom assemble.scr
Code random ( --d) &17 # A7 -) move $e trap 2 A7 addq .l
                   D0   SP -) move Next end-code  \ XBIOS
Create dc_size 8 allot
2Variable dot_fact 2Variable dc_middle
: init_dc dice_f dup objc_offset swap 2dup dc_size 2!
             rot objc_getwh swap
              2dup 4 / swap 4 / swap dot_fact 2!
             >r  rot + swap r> + dc_size 4+ 2!
             dc_size    dup @ swap 4+ @ + 2/ dc_middle !
             dc_size 2+ dup @ swap 4+ @ + 2/ dc_middle 2+ ! ;
: dice_form     0 sf_interior 1 sf_perimeter dc_size 4@ rfbox ;
: dot ( x y--)  1 sf_interior 0 sf_perimeter
   dot_fact @ * swap dot_fact 2+ @ * dc_middle 2+ @ + swap
       dc_middle @  + swap dot_fact @ 2 5 */ circle ;
```

SCR# 7

```
\ Forts. Würfel                                             ad17jan88

: one 0 0 dot ;                 : two -1 -1 dot 1 1 dot ;
: three one two ;               : four two -1 1 dot 1 -1 dot ;
: five four one ;               : six four -1 0 dot 1 0 dot ;

Create dices ] one two three four five six [

: show_d ( w--) dice_form 1- 2* dices + perform ;

' evnt_timer Alias wait


: (dice ( --w) random drop 6 u/mod drop 1+ dup show_d
               100. wait ;
```

SCR# 8

```
\ Namen-Verwaltung,                                         ad2jan87
Create sp_stati stat_sp1 c, stat_sp2 c, stat_sp3 c, stat_sp4 c,
: Atari?    4 0 DO sp_stati I + c@
                state_gaddr 1@ 1 and 0= I set_npl LOOP ;
: rt_Atari? 4 0 DO  sp_stati I + c@
                    0 0 0 0 I get_npl 1 and 0 objc_change LOOP ;
Create sp_namen sp_name1 , sp_name2 , sp_name3 , sp_name4 ,
Create sp_nn       sp_n1 ,    sp_n2 ,    sp_n3 ,    sp_n4 ,
| : sp_namei ( i--laddr) inputnam     tree! 2* sp_namen + @
             text_gaddr ;
| : sp_ni    ( i--laddr) screen tree! 2* sp_nn + @ text_gaddr ;

| &10 Constant nam_len  \ muß auch in RSC stimmen !
: set_sp_n    4 0 DO I sp_namei I sp_ni nam_len lcmove LOOP ;
: reset_sp_n  4 0 DO I sp_ni I sp_namei nam_len lcmove LOOP
                  screen tree!    ;
```

SCR# 9

```
\ Forts.,          Objektverwaltung                         ad17jan88
: show_screen  screen tree! show_object2 set_fig's figure  ;

: input_names  hide_object2  inputnam tree! show_object2
               finger sp_name1 form_do dup objc_ret
              OK = IF Atari? set_sp_n ELSE rt_Atari? reset_sp_n
              THEN hide_object2  show_screen ;

: instruction   hide_object2 instruct tree! show_object2 finger
                0 form_do  objc_ret hide_object2 show_screen ;

: proceed ( obj_nr--obj_nr false/true)
     Ende       case? ?dup IF quit alert 2 =  IF   rsrc_free
                      grexit bye ELSE show_c exit true THEN THEN
     Neues_sp   case? ?dup IF init input_names  abort  THEN
     Anleitung case?  dup IF instruction           THEN     ;
```

SCR# 18

```
Das sind die Formen für Polymarker: Stern, Quadrat, Kreuz,Rombus
 man kann hier etwas probieren, um mit dem Hintergrundmuster
 beste Erkennbarkeit zu erreichen.
Die Farben der Spieler. Bei sw natürlich nur Schwarz. Auch hier
 wären Versuche angebracht.
Zeichnet das Symbol des Spielers in das Feld n. Es hat die halbe
 Feldgröße. Das kann man eventuell weglassen.


Löscht die Figur auf Feld n. Es wird einfach das Feld neu ge-
 zeichnet.
Jedesmal wenn das Spielfeld wiederhergestellt wird (auch am An-
 fang) müssen die Figuren neu gezeichnet werden.
 Hier wird zum Beispiel auf Befehle von maedn.scr zugegriffen.
```

SCR# 19

```
Hier der berüchtigte Würfel. random wurde bereits oben beschrie-
 ben.

Größe und Lage des Würfels. Wird bei jedem Starten neu ermittelt
Abstände der Punkte horizontal und vertikal, Mittelpunkt des W.
Initialisierung des Würfels.  DICE_F  ist eine i-box,
            die die Größe und Lage des Würfels hat.
            die Punkte haben untereinander den gleichen Abstand
            wie zum Rand. Der Mittelpunkt ergibt sich  aus dem
            arithmetischen Mittel der Koordinaten der Eckpunkte

Zeichnet den Würfel (gefülltes Viereck mit runden Ecken)
Punkt mit Koordinaten x und y. Mittelpunkt: 0 0
 -2<x<2   -2<y<2
    Der Radius der Punkte beträgt 2/5 des Abstandes
```

SCR# 20

```
Man braucht sich nur einen Würfel vorzustellen, um diese 6 Wör-
 ter zu verstehen.


Tabelle der sechs Routinen. ( Das gleich wie ' one , ... )

Zeichnet Würfel für Wurf w

Wenn der Computer gegen sich spielt, geht das so schnell, daß
 man den Würfel nicht erkennen kann. Deshalb etwas Geduld 68k!

Hier der berüchtigte Würfel. Das von MAEDN.SCR benutzte Wort
 liegt allerdings etwas weiter hinten. Damit ist das Spiel
 unterbrechbar!
```

SCR# 21

```
Nicht erschrecken! Es sieht schlimmer aus, als es ist.
Tabelle der buttons aus inputnam
Wertet bei Verlassen die B's aus. Wenn sie selektiert sind, wird
  der entsprechende Spieler zur Maschine.
Wird das Formular mit Cancel verlassen, muß der Status der But-
 tons wiederhergestellt werden (ohne Neuzeichnen).
aus inputnam     die Namen der Mitspieler als Tabelle (Objct#)
aus screen        "   "   "   "   "   "   "
liefert die adresse (Langwort) des Namen-Strings in der Resource
 für screen und inputnam


Die Maximallänge des Namens (9 Zeichen ! + '0' C-Format)
Die Namen werden einfach kopiert.
Die Richtung ist die gleiche wie bei Atari?
```

SCR# 22

```
zeichnet den Bildschirm mit allem was dazugehört ( außer dem
 Würfel, denn dazu müßte der letzte Wurf bekannt sein.
damit die Namen richtig eingegeben werden können, darf nur OK
 und Cancel exit- bzw. touchexit-status haben! Der Rest der
 Definition dürfte klar sein, gell?


zeigt das Anleitungsobject. Dabei ist die Maus ein Zeigefinger,
            genau wie bei der Spielernameneingabe.

verarbeitet die objc# ( von screen). Nur die drei Buttons werden
            ausgewertet (true).
            Dabei wird bereits der Mechanismus (screen 12) vor-
            ausgesetzt( game wird über abort gestartet.) man
            könnte es auch durch init ersetzen.
```

SCR# 10

```
\ Funktionen, die von MAEDN.SCR aufgerufen werden          ad2jan87
: show_name ( i--) 8 0 DO I sp_nn + @ screen_c 4@ 0 1
                                        objc_change  2 +LOOP
          2* sp_nn + @ screen_c 4@ 1 1 objc_change ;

: gmove ( from to--)  swap redraw_f show_fig ;

: (click ( --obj false/true) 1 1 1 evnt_button drop 0 8
             graf_mkstate   intout 2+ 2@  swap
             objc_find proceed     1 1 0 evnt_button drop ;

: click (click 0= IF drop THEN ;
: get_throw ( --n) BEGIN (click 0= dup IF drop  gobjf THEN
                   UNTIL ;
```

SCR# 11

```
\   ginit infinit dice                                     ad2jan87
: ginit    grinit hide_c  rsc_load  init_res  init_screen
           screen tree! set_sp_n init_fds init_dc
           init
           show_screen  1 sf_color Atari? ;


: infinit    BEGIN click false UNTIL ;


: dice     (dice  graf_mkstate intout 6 + @ IF click THEN  ;
```

SCR# 12

```
\ loadscreen und Autorun-Mechanismus                       ad16jan88
\needs code           2 loadfrom assemble.scr
use strings.scr 5 load 6 load
include gem\basics.scr
use gem\vdi.scr  3 load  5 7 thru 9 load 12 load 15 load
use gem\aes.scr  3 load  5 load 14 17 thru 22 load 25 load
                 29 load 31 load 35 36 thru
use gem\supergem.scr 2 load 4  load          .( GEMs loaded) cr
: objc_getwh objc_gaddr $0.14 d+ 12@ swap ;
: bell 7 conl ; \ in diverses.scr
B: include maedn.scr

   : set_auto ['] infinit Is 'quit ['] game Is 'abort ;
   : main ginit set_auto  infinit  ;
 ' main Is 'cold                    \ für Autostart
savesystem maedn.prg .( Danke ) bye
```

SCR# 23

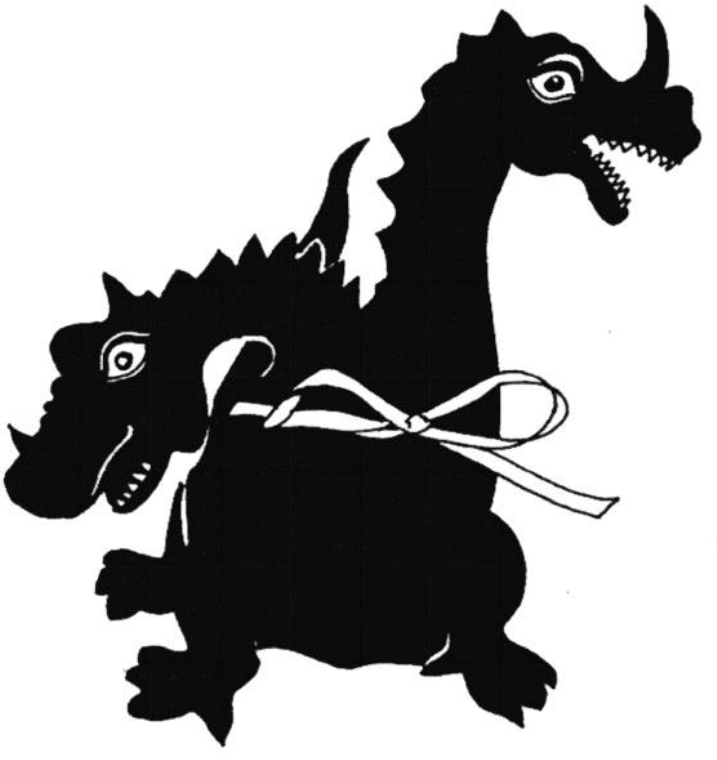
```
zeigt den Spieler n, indem dessen  Name  invertiert
      gezeichnet wird.
     Vorher müssen alle Namen deselektiert werden.

erst alte Figur löschen, dann neu zeichnen.

erwartet einen Mausknopfdruck und verarbeitet ihn, wenn möglich.
       kein form_do, da dann immer ein Object angeklickt werden
       müßte. Dafür müßte dann der screen und alle Unterobjekte
       exit oder touchexit sein.
wie oben, nur werden nicht verarbeitete Objecte ignoriert.
Zugeingabe, diesmal mit allem Komfort: Wartet bis ein Spielfeld
            angeclickt wurde.
```

SCR# 24

```
die gesammelten Initialisierungen: Würfel, Spielfeld
          Spielernamen, Spielerstatus,  Füllfarbe
    grinit ist eine Definition des FORTH-GEM's, es erledigt
    die Anmeldung an VDI und AES


Endlosschleife, wenn kein Spiel läuft. (nach Sieg oder anfangs)
   dies gehört nicht zu den Definitionen, die von maedn.scr
   erwartet werden.

Würfel- diesmal mit Interaktivität: dice ist eine der wenigen
  GEM-Routinen die mit Garantie aufgerufen werden.
```

SCR# 25

```
loadscreen aller für das GEM notwendigen Dinge.          ad17jan88
würde man das gesamte GEM einbinden, wäre das Programm hoff-
 nungslos zu groß. Deshalb werden hier nur die Screens geladen,
 die tatsächlich gebraucht werden.
 Noch kleiner würde das Programm, wenn man einen Targetcompiler
 verwendet. Doch wer hat den schon!
 Man kann aber noch ein übriges tun, indem man das Fileinterface
 rauswirft und den Assembler und das Fileinterface nochmals,
 aber auf den Heap lädt. ( Siehe Handbuch des volksFORTH )
 Außerdem kann man daran denken, alle Definitionen headerless
 zu laden ( siehe Editorquelltext des volksFORTH).
 Zum Autorun-Mechanismus siehe Text.
 Achtung! Es könnte sein, daß die Nummern der Screens sich ge-
 ändert haben. Deshalb sollte man sich nicht wundern, wenn das
 System irgendein Wort nicht kennt, dann hat es etwas Falsches
 geladen.
```



FORTH-Tagung '90

Wir möchten Sie heute noch einmal an unsere FORTH-Tagung '90 vom 20. bis 22. April 1990 an der Johann-Wolfgang-von-Goethe-Universität in Frankfurt am Main erinnern.

Die politischen Veränderungen in der DDR zeigen ihre positive Wirkung auch auf die FORTH-Gesellschaft e.V. und auf unsere Jahrestagung. Schon jetzt zeichnet sich ab, daß wir mit einer starken Beteiligung aus der DDR rechnen können. Wir erwarten Gäste aus den Forschungsinstituten von Berlin, Dresden, Leipzig, Ilmenau und der Kammer der Technik in Suhl. Es wurden sehr interessante Vorträge angemeldet und vor allem besteht großes Interesse an Geschäftsverbindungen zu westlichen Firmen.

Vorgespräche mit den Vertretern der Gruppen in der DDR, die sich mit FORTH befassen, haben gezeigt, daß Interesse an einer weitergehenden Zusammenarbeit besteht. Die Mitgliederversammlung am Sonntag, den 22. April ist sicher ein geeignetes Forum, dafür Wege zu finden.

Schicken Sie bitte Ihre Anmeldung möglichst bald an das Tagungsbüro und vergessen Sie nicht Ihre Zimmerreservierung beim Verkehrsamt Frankfurt am Main vorzunehmen.

Heinz Schnitter

Einladung

zur ordentlichen Mitgliederversammlung

der FORTH-Gesellschaft e.V.

am Sonntag, den 22. April 1990 ab 10:00 im Institut für Angewandte Physik,
Robert-Mayer-Str.2-4 an der Johann-Wolfgang-von-Goethe-Universität
6000 Frankfurt/Main.

Ab 20. April findet das jährliche FORTH-Treffen statt. Auch in diesem Jahr gibt es wieder interessante Vorträge über FORTH-Anwendungen in Wissenschaft und Industrie.

Tagesordnung der Mitgliederversammlung

1. Rechenschaftsbericht des Direktoriums
2. Kassenbericht und Jahresbilanz 1989
3. Aussprache und Entlastung des Direktoriums
4. Wahl des Direktoriums
5. Mitgliederversammlung 1991
6. Berichte aus den Lokalen Gruppen
7. Verschiedenes

Anschrift: *FORTH-Tagungsbüro '90*
c/o Frank Stüss
An der Turnhalle 6
6369 Schöneck

Schnelles FORTH für den MC68000

von Jörg Plewe,
Großenbaumer Str. 27,
4330 Mülheim

Stichworte

» Subroutine-threaded-Code,
» Forthchips,
» MC68000

Interaktive Programmiersprachen wie FORTH oder Lisp müssen ihre hervorstechenden Eigenschaften meist durch einen großen Laufzeitverlust gegenüber den Compilersprachen wie C, Pascal oder Fortran bezahlen. So ist ein Programm in volksFORTH auf dem Atari ST mindestens 5 mal langsamer als ein vergleichbares Programm in C. Dieser Laufzeitnachteil wird heute meist durch Spezialprozessoren wie den NC4000 oder RTX2000 oder den Lisp-Chip von HP ausgeglichen. Dieser Artikel soll nun zeigen, daß auch der weit verbreitete MC68000 ein Prozessor ist, der alle Voraussetzungen erfüllt, um 'FORTH-Prozessor' heißen zu dürfen. Dabei wird gleichzeitig gezeigt, wie ein schnelles FORTH auf diesem Chip zu implementieren ist.

FORTH-Systeme sind prinzipiell Implementationen einer nur gedachten, virtuellen FORTH-Maschine. Ein Chip kann dann ein FORTH-Chip sein, wenn er die Eigenschaften der virtuellen Maschine durch Hardware nachempfindet.

Das hervorstechendste Merkmal der virtuellen FORTH-Maschine ist die Tatsache, daß sie über zwei Stacks verfügt. Der eine dient zur Aufnahme von Daten als allgemeine Schnittstelle zwischen den FORTH-Unterprogrammen, den Worten. Der zweite, der Returnstack, wird benutzt, um beim Aufruf eines Wortes die Rücksprungadresse für dieses Wort bereitzuhalten. Dieser Stack unterliegt, obwohl er in der Regel vom inneren Interpreter automatisch bedient wird, anders als in C einem uneingeschränkten, expliziten Zugriff durch das Programm (`>R` oder `R>`). Worte wie `'."'` können diesen Umstand nutzen. Der MC68000 bietet nun 8 Allzweckregister, die durch Adressierungsarten mit postinkrement oder prädekrement als Stack eingesetzt werden können. Es sind also mehr als die benötigten zwei vorhanden, was später noch ausgenutzt werden kann. Was den MC68000 als FORTH-Prozessor aber nun so interessant macht, ist die Tatsache, daß eines dieser Register - A7 - vom Prozessor genauso bedient wird, wie es ein innerer Interpreter in *klassischen* FORTH-Implementationen mit dem Returnstack tun muß.

Der innere Interpreter hat die Aufgabe, (virtuellen) FORTH-Code abzuarbeiten. Hier ein kurzes Beispiel:

```
: nip ( n1 n2 -- n2 )
      swap drop ;
```

Beim Kompilieren wird hierfür folgender Code erzeugt:

```
Header für NIP
      Sprungcode zu SWAP
      Sprungcode zu DROP
      Sprungcode zu NEXT.
```

Der innere Interpreter hat nun die Aufgabe, diese Sprungcodes auszuwerten. In den klassischen Implementationen besteht der Sprungcode einfach aus einer 16-Bit Adresse. Eine kleine Routine des inneren Interpreters verzweigt nun dorthin. Bei der Verzweigung zu `SWAP` wird dabei die Adresse des 'Sprungcode zu `DROP`' auf dem Returnstack abgelegt. `SWAP` ist nun selbst dafür verantwortlich, den Rücksprung durchzuführen. Das gleiche gilt natürlich für unser `NIP`, das ja ebenfalls irgendwo aufgerufen wurde. Dazu dient das Wort `NEXT`. Es holt die Adresse vom Returnstack und verzweigt dorthin. Es ist einleuchtend, daß dies zu extrem kurzen Programmcodes führt, was in vielen Fällen durch begrenzten Speicher sehr wichtig ist. Man erkennt aber auch, daß der innere Interpreter für seine Arbeit Zeit braucht. Diese kann bis zu 2/3 der Rechenleistung in Anspruch nehmen. Auf dem MC68000 ist dieser innere Interpreter nun schon eingebaut, wenn man eine einfache Übersetzungsregel benutzt:

```
'Sprungcode zu <Wort>' --> JSR <Wort>
'Sprungcode zu NEXT'   --> RTS
```

D.h. statt beim Kompilieren die 16-Bit Adresse eines Fortwortes ins Vokabular einzutragen, trägt man den Code für JSR ($6100) und die 32-Bit Adresse ein. Das ';' erzeugt dann noch den Eintrag für RTS ($4E75). Man sieht, daß man nun statt 16 Bit stets 48 Bit (Code + Adresse) benötigt. Der gesteigerte Platzbedarf ist aber auf modernen Rechnern mit in der Regel mindestens 1 MB Speicher kein Problem mehr. Arbeitet der Prozessor diesen Code ab, geht er genauso vor wie der oben beschriebene innere Interpreter. Wenn JSR <Adresse> ausgeführt wird, dann schiebt der MC68000 dabei die Adresse des folgenden Befehls auf den Stack, der durch das Register A7 bezeichnet wird. Der Befehl RTS wiederum verzweigt zu der obersten Adresse auf dem Returnstack und entfernt sie vom Stack.

Der Code hat nun explizit folgende Form:

```
Header von NIP
      JSR SWAP
      JSR DROP
      RTS
```

Der MC68000 kann diesen Code nun direkt ausführen. Man bemerke, daß hier zugleich virtueller FORTH-Code und Maschinencode steht. Ein Prozessor, bei dem diese beiden identisch sind und der dazu noch einen separaten Datenstack bedienen kann, darf sich wohl FORTH-Prozessor nennen.

Ein weiteres wichtiges Feature des MC68000 besteht darin, daß er einige FORTH-Primitive direkt als Opcodes kennt. Hier einige Beispiele:

(Das Register A6 sei als Datenstack, A5 als Zeiger auf die USER-Variablen benutzt)

```
DROP ↔ addq.l #4,a6
DUP  ↔ move.l (a6),-(a6)
NIP  ↔ move.l (a6)+,(a6)
OVER ↔ move.l 4(a6),-(a6)
LIT  ↔ move.l #xxxxxxxx,-(a6)
```

Viele andere elementare FORTH-Worte bilden sehr kurze Routinen:

```
@    ↔ move.l (a6),a0
       move.l (a0),(a6)
!    ↔ move.l (a6)+,a0
       move.l (a6)+,(a0)
BASE ↔ lea offset_base(a5),a0
       move.l a0,-(a6)
```

Eine FORTH-Implementation, die nun solchen 'Subroutine-threaded'-Code erzeugt, ist gegenüber der klassischen Implementation sehr viel schneller und kommt z.T. an C-Programme schon heran. Damit ist aber der volle Leistungsumfang des MC68000 noch lange nicht erschöpft. Wenn man soweit gediehen ist, fällt es nicht schwer, das System weiter zu beschleunigen. So kann man statt der absoluten JSR-Sprünge falls möglich auch die schnelleren und kürzeren relativen Sprünge BSR.W und BSR.S erzeugen. Die Sprungweiten betragen hier maximal 32 kByte bzw. 128 Byte. Es ist nicht schwierig, den Compiler diesbezüglich zu modifizieren. Auch dann sind immer noch Maschinencode und FORTH-Pseudocode identisch. Der Code kann jederzeit dekompiliert werden!

Der nächste Optimierungsschritt ist komplexer und zerstört die Identität von Maschinen- und Pseudocode, führt aber zu merklichen Geschwindigkeitssteigerungen und sollte in einem guten System optional zuschaltbar sein. Gemeint ist die Benutzung von Makros. Im obigen `NIP`-Beispiel braucht der MC68000 für das `DROP` 44 Taktzyklen (`JSR+ addq.l #4,a6 +RTS`). Davon werden allein 36 für Hin- und Rücksprung gebraucht. Das gleiche gilt für das `SWAP`.

Nun ist es möglich, beim Kompilieren nicht den Sprung zu `SWAP` oder `DROP` ins Vokabular einzutragen, sondern direkt den gesamten Code zu kopieren. `NIP` sähe dann so aus:

```
Header von NIP
  move.l    (a6)+,d0
  move.l (a6),-(a6)    (SWAP)
  move.l d0,4(a6)
  addq.l #4,a6         (DROP)
  rts                  (NEXT)
```

Dieses `NIP` dürfte etwa vier mal schneller sein als das obige! Es kann aber nicht mehr ohne weiteres dekompiliert werden und der Code hat an Transparenz verloren. Bei der Implementation könnte man z.B. jedem Wort, das als Makro dienen soll, die Länge seines Codes in den Header eintragen. An dieser Stelle ist es aber wichtig, daß diese Möglichkeit immer abschaltbar sein muß, da sonst eines der wichtigsten Features von FORTH, das einfache Debugging, abhanden kommt.

Es sind noch weitere Optimierungen denkbar. Z.B. kann an der *Nahtstelle* zweier Makros oft ein unnötiges Verschieben von Daten auf den und vom Stack wegoptimiert werden. Dies sind aber bereits Spezialitäten und führen hier zu weit. Mit einem System, das auf die oben beschriebene Weise implementiert ist, braucht man sich vor manchem C-Compiler nicht mehr zu fürchten. Außerdem hat sich gezeigt, daß ein FORTH-83 Kern immer noch recht klein sein kann. Das System des Autors (FORTH-83 Kern, vorgeschriebener Wortschatz) belegt nur etwa 13kB auf der Diskette.

Benchmark

Das obige `NIP`-Beispiel soll nun als Benchmark dienen. Getestet wurde das folgende Programm auf verschiedenen FORTH-Systemen für Atari-ST (MC68000, 8 MHz).

```
: nip ( n1 n2 -- n2 )
     swap drop ;
: bench ( -- )
     1
     100000 0 DO 1 nip LOOP
     drop ;
```

bzw. für das 16-Bit volksFORTH:

```
: bench ( -- )
     1
     10 0 DO
       10000 0 DO 1 nip LOOP
     LOOP drop ;
```

Getestet wurden folgende Systeme:

- FFORTH: System des Autors, funktioniert wie oben beschrieben. Testbericht (hoffentlich) in diesem Heft. In der Makroversion des Tests wurde die Variable `MACRO` eingeschaltet und statt des ':' ein 'M:' geschrieben.
- 32FORTH: 32-Bit System mit gefädeltem Code von M&T (D.LUDA Software). Wird durch einen komfortablen Multitasker etwas langsam.
- volksFORTH: Das einzige 16-Bit System. Beschreibung überflüssig.
- 4xFORTH: 'Schnelles' 32-Bit FORTH der Dragon Group. Funktioniert ähnlich wie oben beschrieben.
- FORTHmacs: 32-Bit System mit gefädeltem Code. Testbericht siehe VD Volume V, Nr.4, Dezember '89.

Die Ergebnisse

System	Zeit
FFORTH ohne Makrobenutzung	2.7 s
FFORTH mit Makrobenutzung	0.88 s
32FORTH	6.0 s
volksFORTH	4.7 s
4xFORTH	1.2 s
FORTHmacs	4.4 s

Resümee

- Der MC68000 eignet sich hervorragend für die Implementation von FORTH. Die Implementation ist einfach und straigth forward.
- Subroutine-threaded-Code führt zu Laufzeitverbesserungen gegenüber klassischen Systemen um einen Faktor 4-5.
- Die Ausführungsgeschwindigkeit reicht an die von kompilierten Programmen heran.
- Die Stackbreite ergibt sich natürlich zu 32 Bit.
- Der erzeugte Code ist ca. 100% länger als klassischer 16-Bit Code.
- Durch leichte Kompromisse ist die Erzeugung von reloktablen Programmfiles möglich.
- Weitere Stacks, z.B. für Floatingpoint, sind leicht einzurichten.

Ein Treiber für die Hercules-Grafik-Karte im volksFORTH

von Frank Stüss

Das volksFORTH mausert sich; zumindest sollte es das. Da volksFORTH einer der besten FORTH83-Dialekte ist, welche auf dem IBM laufen, sollte es doch auch eine - zumindest rudimentäre - Grafik-Anbindung besitzen. Um dies zu ermöglichen, muß man aber auch daran denken, daß es gerade für den PC verschiedene Grafikkarten mit ebenso verschiedenen Auflösungen und Farben gibt. Zusätzlich ist ja das volksFORTH nicht nur ein FORTH für PC's sondern auch für andere Rechner. Es ist also erforderlich eine möglichst allgemeine Grundlage zu schaffen, eine Grafik zu implementieren. Dies ist mir hoffentlich einigermaßen gut gelungen. Die größte Schwierigkeit bestand darin, zu entscheiden, was den nun ein Grafiktreiber alles können muß. Erstens sollte er verschiedene Grundroutinen beinhalten, die natürlich möglichst schnell sind, jedoch sollte er deswegen nicht gleich größer als 20 kB werden. Wenn keine Grafik-Anwendung 'obendrauf' gesetzt wird, ist ja der ganze Grafik-Teil nur Platzverschwendung. Andererseits müssen auf jeden Fall alle Möglichkeiten einer Grafikkarte ausgenutzt werden und das auf FORTH-Ebene. Nach einigen Rückfragen mit Anwendern und Programmierern (unter anderem auch in Aachen während der FORTH-Tagung bei nächtelangen Diskussionen und 'Streitereien' - ich möchte an dieser Stelle allen Beteiligten danken), kristallisierte sich mehr oder weniger eine Anzahl von Mindestanforderungen an den Treiber heraus. Er muß alle Möglichkeiten der jeweiligen Karte ausnutzen, als da wären:

- Seitenwahl: Es kann sowohl die auf dem Bildschirm sichtbare, als auch die gerade zu beschreibende Bildschirmseite gewählt werden.
- Farben und Farbpalettenauswahl soweit möglich
- Setzen anderer Attribute
- Scrolling (hoch und runter)
- Windows
- volle Kompatibilität zum älteren Video-Treiber 'MULTI.VID' (Dies gilt für das IBM-volksFORTH, ich weiß nicht, ob es diesen Treiber z.B. auch für den Atari gibt). D.h. alle Befehle und Möglichkeiten die dieser Treiber hat (Multitasking, Windows, etc.), müssen komplett emuliert werden. Wenn man statt MULTI.VID eben z.B. HERCULES.VID einlädt und seine Anwendungen damit laufen läßt, so fällt höchstens die schnellere Bildschirmausgabe beim Hercules-Treiber auf.
- Umschalten zwischen verschiedenen Grafikmodi. Dies schließt auf jeden Fall auch Textdarstellung in einem hochauflösenden Modus ein. D.h. man schaltet auf Grafik und schreibt normal weiter.
- Zusätzlich ist zur Ansteuerung von Punktgrafiken eine Primitiv-Pixelroutine, sowie eine Routine zum Ausrechnen des jeweiligen Bytes im Video-RAM implementiert.

Besonders die letzten zwei Punkte ermöglichen es, eine halbwegs effiziente Grafik zu implementieren, die weitgehend unabhängig von der verwendeten Karte ist. Die Idee von Detlef Heid war, noch zwei Routinen zum schnellen Ziehen von vertikalen und horizontalen Linien einzubauen, jedoch konnte ich mich nicht dazu entschließen, da ich glaube, daß der Treiber auch so effizient genug ist und man Linien nicht immer braucht (sprich: Die Arbeit war mir zuviel). Meine Absicht ist, mit diesem Treiber eine Art Standard fürs volksFORTH zu setzen, der einerseits endlich ermöglicht, Grafikprogrammierung zu betreiben, andererseits Programmierer dazu auffordert, diesen Standard zu übernehmen und ihn auf andere Karten und Rechner auszudehnen.

Quelltext Service

Die FORTH-Worte, welche implementiert sind, zeigt die Liste am Ende des Textes.

Nun zum Hercules-Treiber. Im Gegensatz zu anderen Karten, wird die Hercules-Karte sehr wenig vom BIOS des PC unterstützt. Dies zwang mich dazu, spezielle Routinen für Scrolling-, Cursor-, und Textdarstellungen im hochauflösenden Modus zu schreiben. Dabei bin ich einige Kompromisse eingegangen. MS-DOS unterstützt grundsätzlich Textdarstellung in Grafikmodi durch eine Zeichentabelle im ROM für die ASCII-Codes 0-127 und

Stichworte

» volksFORTH,
» Grafik-Treiber,
» Hercules-Karte

durch eine nachladbare Tabelle (DOS-Programm 'GRAFTABL') im RAM für die Codes 128-255. Diese Tabelle hat einen Nachteil: Sie stellt die Zeichen im 8x8-Raster dar. Es ist zwar nicht so schön, wie das Hercules 14x9-Raster, jedoch kann man nun 90X43 Zeichen auf dem Bildschirm darstellen. Um die einzelnen Funktionen des Treibers zu erfüllen, habe ich separate Sätze von Routinen, jeweils für Text- und Grafikmodus geschrieben. Um eine gewisse Geschwindigkeit zu wahren und eine gleiche Menge an Nerven zu verlieren, geschah die Programmierung durchweg in Assembler. Die Möglichkeit Farben anzusprechen entfällt hier natürlich; um aber kompatibel zu anderen (späteren) Treibern zu bleiben, sind die beiden Worte PALETTE und FARBE trotzdem eingebaut. Sie tun nichts außer »DROP«. Eine weitere kartenspezifische Eigenschaft ist die Möglichkeit, die Karte so zu konfigurieren, daß sie auch andere Karten im Speicher zuläßt. Dies bewirken die Befehle FULL und HALF. Standardmäßig ist die Karte im Fullmodus, benutzt also den gesamten Videorambereich. Wer tatsächlich eine zweite Karte benutzt, kann den Befehl HALF benutzen, die Stelle ist im Quelltext angegeben. Im File PIXEL.SCR sind die Implementation einer Pixel- sowie einer Line-Routine enthalten.

Quellen:

[1] M. Tischer: "PC-Intern, Systemprogrammierung" Data Becker

[2] Hercules-Graphics-Display User's Manual.

Da die Quelltexte für den Abdruck zu umfangreich waren, sind sie alle über den Quelltextservice beziehbar (siehe Impressum) - die Red.

Wort	Stackrelation	Erklärung
`writepage`	(-- addr)	übergibt die Adresse der Variablen, welche die Nummer der aktuell zu beschreibenden Seite enthält.
`writepage@`	(-- n)	übergibt die Nummer der zu beschreibenden Seite.
`writepage!`	(n --)	setzt die zu beschreibende Seite.
`seepage`	(-- addr)	übergibt die Adresse der Variablen, welche die auf dem Bildschirm sichtbare Seite angibt.
`seepage!`	(n --)	setzt die sichtbare Seite.
`setpage`	(n --)	setzt seepage und writepage auf Nummer n.
`grafik`	(--)	schaltet in einen Grafikmodus auf eine Seite. Welcher Modus und welche Seite benutzt werden liegt im Geschmack des Treiber-Programmierers. Da beim Hercules-Treiber jeweils zwei Seiten vorhanden sind, bleibt die gewählte Textseite auch im Grafik-Modus erhalten.
`-grafik`	(--)	schaltet in die Normal-Textdarstellung um. Die Seite nach gusto... Im Hercules-Treiber bleibt die Seite aktiv, welche im Grafikmodus beschrieben wurde.
`grmode`	(-- addr)	Variable. Enthält die Nummer des gerade aktiven Grafikmodus, welche Treiberspezifisch ist.
`grafmode`	(n --)	schaltet in den Modus n. Beim Hercules-Treiber bedeutet dies: 0 : Text Seite 0, 1 : Text Seite 1 2 : Grafik Seite 0 3 : Grafik Seite 1.
`p/cc`	(-- n)	pixel per character-column, Punkte vertikal im Buchstabenraster.
`p/cr`	(-- n)	Punkte horizontal.
`p/b`	(-- n)	Pixel per Byte. Punkte pro Byte im Videoram (minimal einer).
`p/row`	(-- n)	Punkte pro Bildschirmzeile.
`p/col`	(-- n)	Punkte pro Bildschirmspalte. Diese Quasi-Konstanten werden ebenso wie c/row, c/col, c/dis je nach Grafikmodus geändert.
Attribute:		
`blink` `underlined` `invers` `normal` `bright`	(--)	setzen das Attribut der nächsten Textausgabe. Wenn ein Treiber eins der aufgeführten Attribute nicht unterstützt, so hat der Aufruf keinen Effekt.
`palette`	(n --)	setzt die aktuelle Farbpalette
`border`	(n --)	setzt die Rahmenfarbe.
`color`	(n --)	setzt die aktive Farbe; diese drei Wörter haben bei der Hercules-Karte keinen Effekt.
`catt`	(-- n)	übergibt das aktive Attribut.
`curshape`	(n1 n2 --)	Definiert das Erscheinungsbild des Textcursors
`area`	(-- addr)	Diese User-Variable enthält die Adresse der Tabelle des aktiven Fensters: Schreibposition, Attribute, Seite und Rahmenkoordinaten.

Wort	Stackrelation	Erklärung
Area:	<name>	Erzeugt eine neue Tabelle, deren Adresse bei Aufruf von NAME in area abgespeichert wird. Alle Ausgabe- Befehle des Treibers orientieren sich an area.
terminal	(--)	Ist der Standardausgabebereich (von Area: erzeugt).
window	(n1 n2 --)	Definiert ein Fenster von Zeile n1 bis einschließlich Zeile n2. Dieses Fenster umschließt ganze Zeilen (kompatibel zu MULTI.VID).
hwindow	(n1 n2 n3 n4 --)	Definiert ein Fenster von Zeile n1 bis n2 und von Spalte n3 bis n4.
full	(--)	löscht die Fensterbereiche und weitet den Ausgabebereich auf den gesamten Bildschirm aus.
(type	(addr n --)	Ist das Type für die aktuelle Seite (writepage). Es wird nicht auf die Fensterkoordinaten geachtet.
(emit (del (at (at?		ähnlich zu oben.
(cr	(--)	Springt in den Anfang der nächsten Zeile. Wenn die aktuelle Schreibposition die letzte Zeile des Fensters ist, wird der Inhalt des Fensters um 1 nach oben gescrollt und die unterste Zeile gelöscht.
(page	(--)	löscht das aktive Fenster.
scroll	(--)	rollt das aktive Fenster nach oben.
scrolldn	(--)	rollt das aktive Fenster nach unten.
blankline	(--)	löscht die aktive Zeile im aktiven Fenster.
cur!	(--)	setzt den Cursor in das Fenster des aktiven Tasks.
curat?	(-- x y)	übergibt die Koordinaten des Bildschirmcursors.
status	(-- addr)	Dieses Flag gibt an, ob die Statuszeile ausgegeben werden soll, oder nicht. (status on, status off).
.base .sp (.drv .scr .space		sind Hilfswörter für die Statuszeile, welche auch anderweitig benutzt werden können.
(.status	(--)	gibt den Status aus. Wird bei deffered .STATUS eingehängt.
gclear	(--)	löscht den gesamten Bildschirm, unabhängig vom Modus. Hinter- und Vordergrundfarbe bleiben erhalten.
Folgende Wörter wurden redefiniert:		
restorevideo	(-- n/ff)	Wenn ungleich null, dann wurde der gesamte Bildschirminhalt gesichert. N übergibt die Kennung des Speicherbereichs (beim PC : das Segment), in welchen das Bild geschrieben wurde.
savevideo	(n --)	holt das Bild mit der Kennung n wieder zurück.
video@	(-- addr)	übergibt das Segment (beim PC) der aktuell zu beschreibenden Seite im Video-RAM.
Für die Grafik:		
(pixel	(-- addr)	übergibt die Adresse der primitiv-Pixel Routine. Diese ist mit einem Assembler-Call aufzurufen, also Rechnerabhängig. Beim PC sind die Register folgendermaßen zu setzen: A+ : 0<>Punkt setzen, 0=Punkt holen A- : Wenn A+ <> 0 dann 0=Punkt löschen, 1=Punkt setzen >1 Punkt invertieren Allgemein enthält A- das Attribut des gesetzten Punkts. R+ : Nummer der zu beschreibenden Seite C : X-Koordinate D : Y-Koordinate Ausgaben: Beim Punkt-Holen enthält A- das Attribut des Punktes. Die Register A C D werden verändert. (kompatibel zu den BIOS-Interrupts)
(getpos	(-- addr)	übergibt die Adresse der Routine (getpos. Sie berechnet die Adresse der Koordinaten im Video-RAM. In dieser Adresse relativ zum Videosegment ist das Byte, welches den Punkt (x, y) enthält. Die Register: C : X-Koordinate D : Y-Koordinate Es wird die aktuell zu beschreibende Seite benutzt (writepage). Ausgabe: W : Adresse des Byte.
(video@	(-- addr)	Wie oben, jedoch für das Videosegment. Keine Eingabe. Ausgabe: W : Segment des Video-RAM abhängig von writepage.

Lokale FORTH-Gruppen, die sich regelmäßig treffen:

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

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

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

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

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

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

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

FORTH-Fachgruppen:

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

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

Es möchten in ihrer Region eine Gruppe gründen:

3300 Braunschweig Martin Holzapfel, Bassestr.17.

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

5000 Köln 60 Michael Heycke, Boltensternstr.

4830 Gütersloh 1 Ludwig Röver, Holzheide 145A

Eine Fachgruppe will gründen:

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

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

Hier kann man um Rat fragen:

02103/556 09 Jörg Staben, Dienstag und Freitag, 20.00 - 22.00 Uhr

06187/91503 Frank Stüss

02845/28951 Karl Schroer

05221/23504 Andreas Findewirth, Im Großen Vorwerk 48, 4900 Herford

Ansprechpartner zu bestimmten Interessengebieten:

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

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

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

FORTH-SYSTEME

ANGELIKA FLESCH

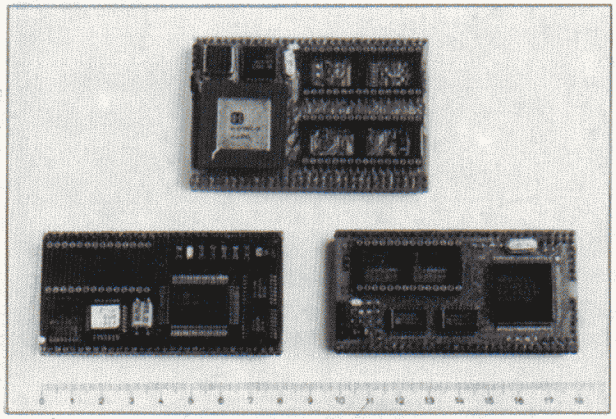
Postfach 1103
D-7814 Breisach

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

UR/FORTH

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

ModuNORM



CPU-Steck-Module im Scheckkartenformat:

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

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

LMI FORTH-83 Metacompiler

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

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

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

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

Serieller ROM/RAM Simulator

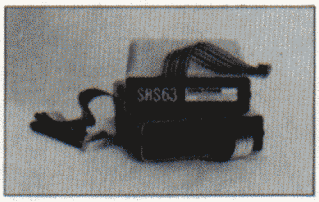
Entwickeln Sie romfähige Programme ?

Müssen Sie neu entwickelte Einplatinencomputer testen ?

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

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

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



Dann ist unser SRS63 die optimale Ergänzung Ihres Arbeitsplatzes.

Sie werden vom Preis-Leistungsverhältnis überrascht sein.

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

0041

mikrap

Mikrap
AG für Mikroelektronik-Applikation

CH-8840 Einsiedeln
Langrütistrasse 44
Postfach

Telefon 055 53 65 55
Telefax 055 53 61 23