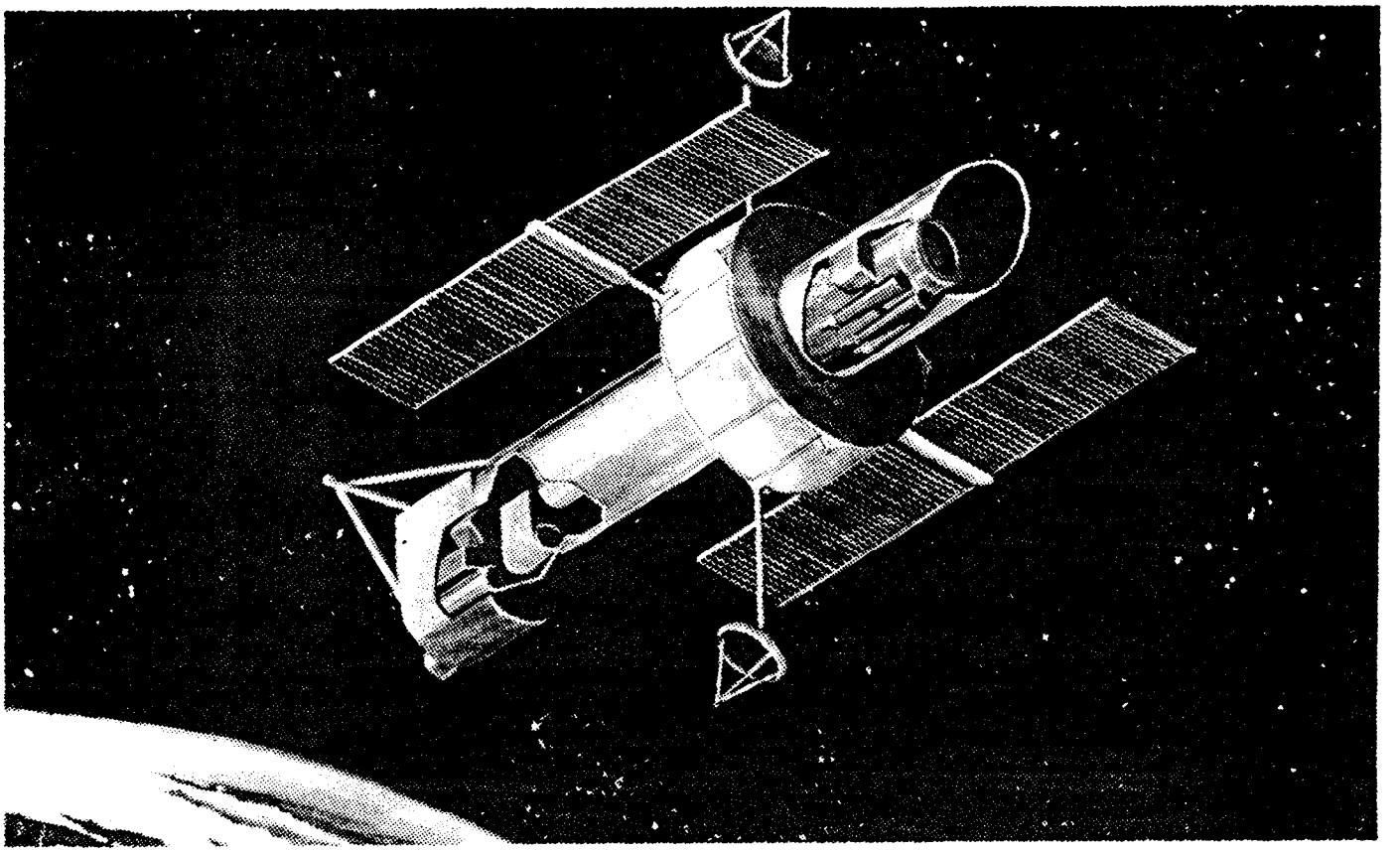
5.-DM

VIERTE DIMENSION

Vol-I/No.1 Oktober 1984



EDITORIAL

Hallo liebe Freunde des Forth & der Forth - Gesellschaft !

Zuerst einmal muß ich mich wohl entschuldigen für die Verspätung mit der diese VIERTE DIMENSION bei euch angekommen ist. Natürlich gibt es auch eine Ausrede dafür, nämlich: Dies ist unsere erste VIERTE DIMENSION die halbwegs professionell aussieht, weil wir sie im FOTOSATZ produziert haben. Um dabei Kosten zu sparen, haben wir dem Fotosetzer versprochen alles mögliche selber zu machen, und das hat uns dann letztlich viel mehr Zeit gekostet als ursprünglich erwartet. Wie dem auch sei: Hier ist sie unsrere No.1 der VIERTEn DIMENSION und ich hoffe, daß sie euch besser gefällt als die Nullnummer. Die ja teilweise sehr ungünstige Kritiken provoziert hat. Das geht soweit, daß einige Leute glauben wir würden "Weißmalerei" betreiben und nur die positiven Kritiken zitieren. Ich hoffe, daß euch die LESERSEITE (S.18) vom Gegenteil überzeugen kann. Andere Wirkungen Eurer Briefe sind: Der C64 - (resp. 6502 -) DISASSEMBLER (S. 5) und der FIG-FORTH - DECOMPILER (S. 11), die der Georg Rehfeldt in die PUBLIC DOMAIN übergeben hat, sowie die ersten Tips und Tricks AUS DER TRICKKISTE (S.11), die DR. med. dent. Greiner ausgegraben hat. Ich hoffe, daß ihr nun angeregt seid auch mal in eure "Werkzeugkisten" zu schauen und uns mit TIPS & TRICKS aushelft. Ach ja, KLEINANZEIGEN gibt es jetzt auch (S. 22).

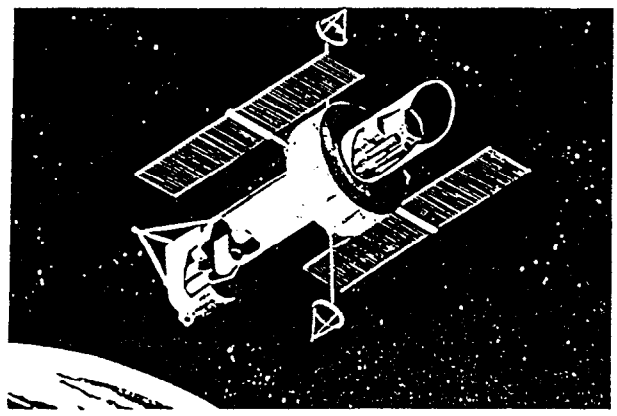
Die vielen Anfragen nach guten FORTH - Implementationen für den Computer XXX machen uns ganz verlegen, hauptsächlich deshalb, weil wir sie nicht beantworten können, was wiederum daran liegt, daß unsere für diesen Zweck entworfenen Fragebögen nur spärlich an uns zurückgesendet wurden. Aber wir geben die Hoffnung nicht auf und haben auch dieser Ausgabe einen solchen Fragebogen beigeheftet. Bei ausreichender Beteiligung, dürfte eine Auswertung in der No.2 erscheinen. Solange müßt ihr euch mit PRODUKTNEUIGKEITEN (S.16) behelfen, wobei wir weder garantieren, daß es sich dabei wirklich um Neuigkeiten handelt, noch ob die Aussagen der Hersteller über deren Produkte zutreffen. Für nicht eingehaltene Versprechungen jedenfalls fühlen wir uns nicht zuständig. Aber eines ist schon getan, um die Anfragen an uns besser handhaben zu können: Jochen Dahmke von der AG-DOKU hat nämlich alle über FORTH erhältlichen Informationen, soweit er an diese herankommen konnte, in eine Datenbank eingetippt. Auf diese können wir jetzt zurückgreifen wenn jemand speziellere Fragen zum Prozessor XYZ in Bezug auf Forth hat. Oder alle Quellenhinweise zu FORTH - Publikationen benötigt. Der Anfang ist getan.
Auch wir hier, also das TEAM VOM C.I.A, waren nicht untätig und haben alle uns bekannte PUBLIC DOMAIN SOFTWARE in unserer (natürlich in FORTH geschriebenen) Datenbank erfasst. Dabei handelt es sich im um die in den FORTH DIMENSIONS und einigen in FIG-CHAPTER-Newslettern veröffentlichten Sources sowie den ersten Screens aus unserer eigenen Programmierküche. Ein Verzeichnis dieser Screens könnt ihr für DM 3,- bei uns bestellen (Bestellnummer FIG-KOPIE 001 und den Rückumschlag bitte nicht vergessen !) Im Verzeichnis findet ihr einige Stichworte über den jeweiligen Sinn & Zweck der Screens, falls es dabei stand auch Autor sowie für welchen Standard das Programm verfasst wurde, die Anzahl der Screens und die Zahl der Kopien die dabei herauskommen. Leider können wir diese Screens auf keinem anderen Datenträger als PAPIER und in keinem anderen Format als alphanumerischem ASCII-Code liefern, so daß ihr euch die Mühe machen müßt, diese selber einzutippen. Eins sei noch vorweg gesagt: Ihr habt zwar den Vorteil auch mal den Programmier- und Dokumentationsstil anderer FORTHler kennenzulernen, ob euch dieser zusagt oder ob ihr auch versteht was dort steht, ist eine andere Frage.

Doch zurück zu diesem Heft, ihr werdet unter dem Titel " Einige Todsünden wider das effektive Programmieren " (S.20), einen etwas lockereren Versuch der Betrachtung gewisser EDV-Praktiken zu lesen bekommen. In der Hoffnung solche Texte nicht als reine Überheblichkeit ausgelegt zu bekommen, bitte ich euch doch selber mal etwas "lustiger" über eure Erfahrungen in der EDV resp. Programmiererei zu berichten. Computern kann auch durchaus menschlich sein ! Nur wird dieser Aspekt meistens unter den Tisch geschwiegen.

Ernster ist da schon der Artikel "Trends in der Entwicklung der Programmiersysteme" von A.Goppold gemeint. Aus Platzgründen stellen wir euch in dieser Ausgabe nur Teil I & Teil II vor. Teil III ist natürlich der eigentliche Knüller, aber der erscheint erst in der nächsten Ausgabe. Soviel sei hier aber verraten: Neu sind die beschriebenen Trends nur für BASIC, PASCAL & Co KG!

Als Anregung zu weiteren ernsthaften , weil folgenschweren Gedanken könnnt ihr den Artikel " Entwicklung einer orthogonalen Syntax... " auf Seite 3 verstehen. Aber was rede ich hier soviel, am besten ihr lest selber nach was wir so alles zu bieten haben...

Euer Horst-Günter Lynsche



Zum Titelbild:

Ob es nun gerade dieser Satellit ist oder ob es ein anderer sein wird, fest steht jedenfalls: FORTH wurde von der NASA auserwählt als Betriebssystem für diverse Satelliten zur Erforschung des Weltraumes, namentlich zur Erforschung einiger Planeten unseres Sonnensystems, zu fungieren. Wenn FORTH auf der Erde keinen ganz so guten Start hatte: Zwischen den Sternen hatte es schon immer besssere Karten. (Man erinnere sich: Das erste FORTH - System wurde von Charles Moore Anfang der 70er Jahre im Radioastronomieinstitut Kitt Peak entwickelt und eingesetzt.)
Bildquelle: L-5 NEWS, VOL.9/No.8, August 83, Photo courtesy NASA

Impressum:
Herausgeber: Forth - Gesellschaft Deutschland
Verantwortlich: Horst - Günter Lynsche
Redaktion: Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6
040 - 43 50 70 (11-18 Uhr)
Mitarbeiter: Thomas Braun (Layout)
Mathias Heitmann (Satz und Konvertierung)
Eckard Schmidt (Druck)
Klaus Schleisiek (Forth)
Andreas Goppold (Meta)
Ronald Zech (Fig-Bugs)
sowie alle weiteren Mitglieder nach Lust und Laune

Entwicklung einer orthogonalen Syntax für die Sprache FORTH

von Andreas Goppold
Die Arbeitsgruppe LEIBNIZ der FORTH Gesellschaft Deutschland erarbeitet neue Konzepte für die Gestaltung eines Programmiersystems auf der Basis von FORTH. Der hier ausgearbeitete Vorschlag beschäftigt sich mit ideographischen Zeichen für die primitiven FORTH-Operationen.

Software-Ergonomie und Orthogonalität

Die heutigen Anstrengungen der Programmiersprachen-Entwicklung gehen in Richtung auf die Verbesserung der Produktivität des Programmierers. Im Fokus der Bemühungen ist dabei die Unterstützung des Programmierers durch den Computer. Die Benutzer-Unterstützung von Maschinen wie LISA, XEROX Star, Altos und Macintosh zeigt die Richtung, in die man gehen kann. Weitere Beispiele der Entwicklungsrichtung programmiererunterstützender Systeme werden in ACM84 aufgeführt. (Siehe auch: Goppold: Das Paradigma der Interaktiven Programmierung). Ein Trend für Software-Entwicklungs-tools ist in Richtung auf orthogonale Programmier-Umgebungen, die logisch konsistent sind und mit einem einzigen Satz von Regeln beherrscht werden können. (Siehe auch: Goppold: Trends der Programmiersysteme)

Orthogonalität: Hier, eine logische Kategorisierung, mit möglichst geringen Überlappungen der logischen Kategorien. Jede Kategorie stellt sozusagen eine separate Dimension in einem multidimensionalen logischen Begriffsraum dar. Mengentheoretisch eine Unterteilung in disjunkte Teilmengen.

Diese Bedingung war bisher nicht gewährleistet, da Programmiersprachen, Betriebssysteme und Hilfsprogramme jeweils unterschiedliche Modi der Bedienung aufwiesen, so daß ein Programmierer bis zu 7 verschiedene Sprachen benutzen mußte, wenn er mit seinem System arbeiten wollte.

Die Programmierumgebung FORTH

FORTH ist von seiner ganzen Konzeption her als interaktives System entwickelt worden, und zwar schon um 1969 herum, 10 Jahre, bevor das Konzept des interaktiven Programmierens durch die Entwicklung der Microcomputer überhaupt erst seinen Eingang in den allgemeinen Sprachgebrauch fand. FORTH stellt dem Benutzer eine kohärente Programmierumgebung, die er bei der Erstellung und Testen seiner Software nicht verlassen braucht. Es stellt durch interpretatives Arbeiten ein Höchstmaß von Interaktivität bereit, das von konventionell compilierten Sprachen nur schwer erreicht werden kann. Einer der gewichtigsten Vorteile von FORTH ist, daß es auch auf den allerkleinsten Rechnern (in 8 K) eingesetzt werden kann und deswegen auch auf allen bekannten Computern implementiert worden ist. Damit ist es unbegrenzt portabel. Dazu kommt noch die Tatsache, daß das System schneller als PASCAL p-code ist, und graduell auf maximale (assembler-) Geschwindigkeit optimiert werden kann.

Trotzdem hat FORTH, das nach dieser Beschreibung wie das non-plus-ultra System erscheinen müßte, keine besonders weite Verbreitung erfahren. Die Gründe hierfür sind natürlich vielfältig, zum Teil politischer Natur, zum Teil sprachlicher. Der politische Grund ist, daß FORTH nicht durch eine große Herstellerfirma oder andere Organisation gestützt wird. Dafür lassen sich ebenfalls Gründe finden, warum es für eine große Organisation nicht aussichtsreich erscheint, FORTH anzuwenden. Die Sprache ist das Werk eines Einzelgängers, Charles Moore, und hat viele Züge, die man als eigenbrötlerisch bezeichnen kann. Es ist, um es positiv zu sagen, eine "persönliche" Sprache, mit persönlicher Note, die sich schon allein aus diesem Anstrich her nicht gut in eine Konzernumwelt einfügen läßt. Bisher war das allgemeine Paradigma der Computerei ja mit "institutionell" zu beschreiben (siehe auch GOPINT), und eine Sprache, in der man "write-only" programmieren kann, ist eben für die institutionelle Umwelt verdächtig.

Nun hat sich allerdings der Charakter der Programmierumgebung tiefgreifend gewandelt, von dem institutionellen Paradigma hin zum persönlichen. Damit sind die Akzeptanzhürden für FORTH niedriger geworden. Eine immer noch bestehende Problematik für die Akzeptanz von FORTH sind die Namen der FORTH-Worte, die die kalifornischen Erfinder von FORTH "erfunden" haben. Kürzel wie

§ ! ' C, .

etc. deren Sinngehalt nur einem FORTH-Kenner zugänglich ist. Zudem sind sie inkonsistent.

Nun ergibt sich hier in Deutschland eine andere Situation. FORTH ist hier noch recht unbekannt, und es gibt noch kaum Programmierer, deren Gewohnheiten man berücksichtigen muß. Es gibt nur die Möglichkeit, ein elegantes Progammiersystem etwas anders zu verpacken und es damit weiten Kreisen zugänglich zu machen. Es spielt hierbei keine Rolle, wenn die Schreibweise der Sprache anders aussieht als in Amerika, da man einen einfachen CODE-Übersetzer schreiben kann, der ein Programm von einer Syntax in die andere überführt.

Das ist bei FORTH deshalb so trivial, weil es fast keine Syntax hat. Ein FORTH Programm ist eine Liste (wie in LISP) und wird rein sequentiell im Onepass-Verfahren abgearbeitet. Keine look-aheads und andere Problematiken sind nötig. Dadurch wird CODE-Übersetzung sehr billig. Solange man sich innerhalb der stack-Eigenschaften für die einzelnen Worte hält, kann man ihnen Namen geben, wie man will.

Die Ansätze für eine benutzerfreundliche FORTH-Syntax

Die FORTH Gesellschaft verfolgte anfänglich den Weg, die englischen Kommandos von FORTH "eins-zu-eins" ins Deutsche zu übersetzen. Dies erwies sich bald als quälend. Erstens gibt es im Deutschen längst nicht so wie im (amerikanisch-) Englischen, die Gelegenheit und die Mentalität zu kurzen "snappy" Formulierungen. Deutsch ist eine Sprache der Konkatenation. Deutsch ist genau, aber lang. Ein Programmierer verwendet, wenn er kann, nur 3-Buchstaben-Kommandos, weil er, als die Speerspitze der Rationalisierung, es nicht ertragen kann, wenn er übermäßig Zeit mit dem Tippen von Namen verbringen muß. Gewöhnliche Programmiersprachen wie FORTRAN und die meisten Assembler kommen dieser Vorliebe entgegen, indem sie nicht mehr als 5 oder 7-Charakter Namen erlauben.

Daß die Rationalisierung ihre Kehrseite hat, weil niemand außer dem Programmierer, und der nach drei Monaten auch nicht mehr, verstehen kann, was mit dem Code gemeint ist, hat man durch Kommentare zu beheben versucht, aber im Ganzen ist die Frage der Dokumentation von Programmen eher das größte Rätsel der Computerei. Wie in GOP-PROG gezeigt, ist Selbst-Dokumentation in FORTH, wenn auch nicht erzwungen, so doch sehr einfach möglich.

Das Problem einer Ergonomisierung der FORTH-Syntax stößt also auf mehrere Hürden. Einerseits mag man wegen der Tipp-Effizienz nicht gerne lange Namen, andererseits sind prägnante und kurze deutsche Worte rar, und dritterseits scheint eine Übersetzung der in den englischen Worten enthaltenen Konzepte nicht so recht in das deutsche Begriffsumfeld zu passen.

Die logische Maschine von FORTH

Daher wurde der Ansatz der Überarbeitung der FORTH-Syntax etwas weiter gefaßt. Zu Hilfe kommen wieder einige Eigenschaften von FORTH, die man anderswo nicht findet, Kategorien, in denen man mit dem normalen Informatik-Begriffs-Instrumentarium deshalb auch meist nicht zu denken gewohnt ist. Diese Besonderheiten sind: FORTH arbeitet auf einer logischen Maschine, die durch stack-Programmierung einfacher als normale Registermaschinen konventioneller Bauart ist, aber trotzdem auf einer solchen konventionellen Maschine relativ effizient implementiert werden kann. Man hat die Möglichkeit, maschinennah zu programmieren, (wie in C kann man mit pointern direkt im Speicher arbeiten, ohne Variablen oder Felder zu benutzen).

Die Hierarchisierung

Eine weitere Möglichkeit bei FORTH ist die Hierarchisierung. Moderne Programmiersprachen sind strukturiert. Obwohl auch unter Experten keine wirkliche Einigkeit über die Natur dieser Strukturierung herrscht, läßt sich doch soviel sagen, daß man nach logischen Kategorien zusammengehörende Bereiche auch örtlich zusammenlegt. Diese logische Kategorisierung ist aber unvollständig. Die exakte Natur dieser Kategorisierungen zu klären, ist hier nicht möglich, aber man kann oberflächlich sagen, sie ist eindimensional. Alte Programmiersprachen machen Subroutinen-Aufrufe nicht einfach, und man vermeidet sie daher nach Möglichkeit. Wenn man Subroutinenaufrufe macht, dann geschieht das meist nur in ein oder zwei Ebenen. Das heißt, es gibt eine Hauptroutine, die dann einen ganzen Satz von Unter-routinen aufruft, aber jeweils nur auf der nächst unteren Ebene.

In FORTH ist es eher umgekehrt. Hier hat man gewöhnlich 5 oder 6 Ebenen der Subroutinen-Hierarchie, oft auch mehr als 10. Dies wird möglich durch den explizit vorhandenen return-stack, der separat von dem Datenstack für die reine Kontroll-Information freibleibt und daher eine größerer Freiheit erlaubt als Sprachen, wie PASCAL oder C, die nur einen Stack sowohl für Parameterübergab

als auch Kontrollinformation benutzen. Die meisten Micro-Computer haben demnach auch nur ein stack-Register (rühmliche Ausnahme: die 6800 und 68000er Serie von Motorola).

Die Hierarchisierung führt zu einer natürlichen Ordnung oder auch vertikalen Strukturierung von Problemen nach Ordnungsprinzipien, die man sonst eher im Management-Bereich findet (daher Hierarchie). Es gibt eine klare Entscheidung von "Kompetenzen" der einzelnen Routinen und ihren Kommunikationsmöglichkeiten. Der stack kann ja nur zwischen Routinen derselben Ebene und der beiden nächsten Ebenen dienen. Darunter und darüber aber nicht mehr. Wenn man auf Variable verzichtet, dann ist der Kontrollfluß eines Programmes sehr elegant und interferenzfrei zu lösen, ein ungeheurer Fortschritt zu den bekannten Programmen, in denen Daten und Ablaufinformation in Variablen und damit logisch nicht unterscheidbar sind.

Diese Hierarchisierung unterstützt eine logische Trennung von Kompetenzebenen. Wie in bekannten Beispielen aus der Wirtschaft gibt es eine Ebene, in der die Entscheidungen gefällt werden, und eine Ebene, die das "dirty work" tut. Im Programm heißen diese beiden Ebenen Kontrollstruktur und Produktionsstruktur und sind als Beispiel in GOP-PROG in der Funktion
DELETE-RIGHT-WORD
gezeigt.

Ein Computer ist dazu da, dem Menschen "dirty work" abzunehmen. Programmieren war bisher selber "dirty work", weil der Programmierer alle Ungereimtheiten der Maschine auffangen mußte. Der aussichtsreichste Ansatz zu einer grundlegenden Verbesserung der Programmiersituation liegt in der Konstruktion einer besseren logischen Maschine. In seinem Artikel "Can Programming be liberated..." zeigt Backus die Richtung, in die man gehen muß.

The FORTH Way

Solange man aber noch nicht den total radikalen Weg gehen kann, kann man auf Konzepte zurückgreifen, die mit wenig Aufwand eine Maschine in eine andere verwandeln können und sie damit dem Benutzer zugänglicher machen. FORTH ist einer dieser Wege.

Die neue FORTH-Schreibweise

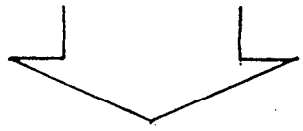
Die jetzige Schreibweise von FORTH weist in diesem Bezug noch einige Verbesserungsmöglichkeiten auf, die mit einer Überarbeitung auch genutzt werden können. Normale Programmiersprachen haben einen relativ kleine Satz von Operatoren, man kann sie auch Primitivoperatoren nennen. Es gibt davon nicht mehr als hundert. In FORTH gibt es die PRIMITIVES oder den FORTH-Kernel, der ebenfalls etwa diese Größe hat. Diese hundert Operatoren (oder FORTH Worte) machen sozusagen die Kleinarbeit des Programms. In FORTH sind alle anderen möglichen Worte, und von denen gibt es unendlich viele, auf diese Worte aufgebaut (durch den Mechanismus der FORTH Definition).

Mnemonik der Kommandos

Es stellt sich nun heraus, daß es wünschenswert ist, daß man nicht nur die Worte mit den Primitiv-Operationen in "die untere Ebene" gruppiert und möglichst keine Primitiv-Operationen in den oberen Ebenen vorkommen läßt, sondern daß man zudem noch das Aussehen dieser Operatoren so gestaltet, daß sie "mnemonisch" an das erinnern, was sie tun. Da hilft eben die oben erwähnte angloamerikanische Denkweise wenig, und es gibt andere, wesentlich bessere Wege, eine maximale logische Konsistenz zu erreichen.

Heutige Maschinen wie APPLEs LISA und der Macintosh geben hier natürlich die idealen Möglichkeiten, in Form von Pictogrammen die Wirkungsweise zu verdeutlichen, aber auch in einer alphabetgebundenen Darstellung ist mit Hilfe einer ideographischen Grammatik viel möglich.

Wenn wir die Konzepte hinter den FORTH-Worten zurückverfolgen, (die ja sehr maschinennah sind) dann gibt es einige Primitiv-Kategorien, die sozusagen die Primitives der Primitiv-Operationen sind. Diese Kategorien sind:

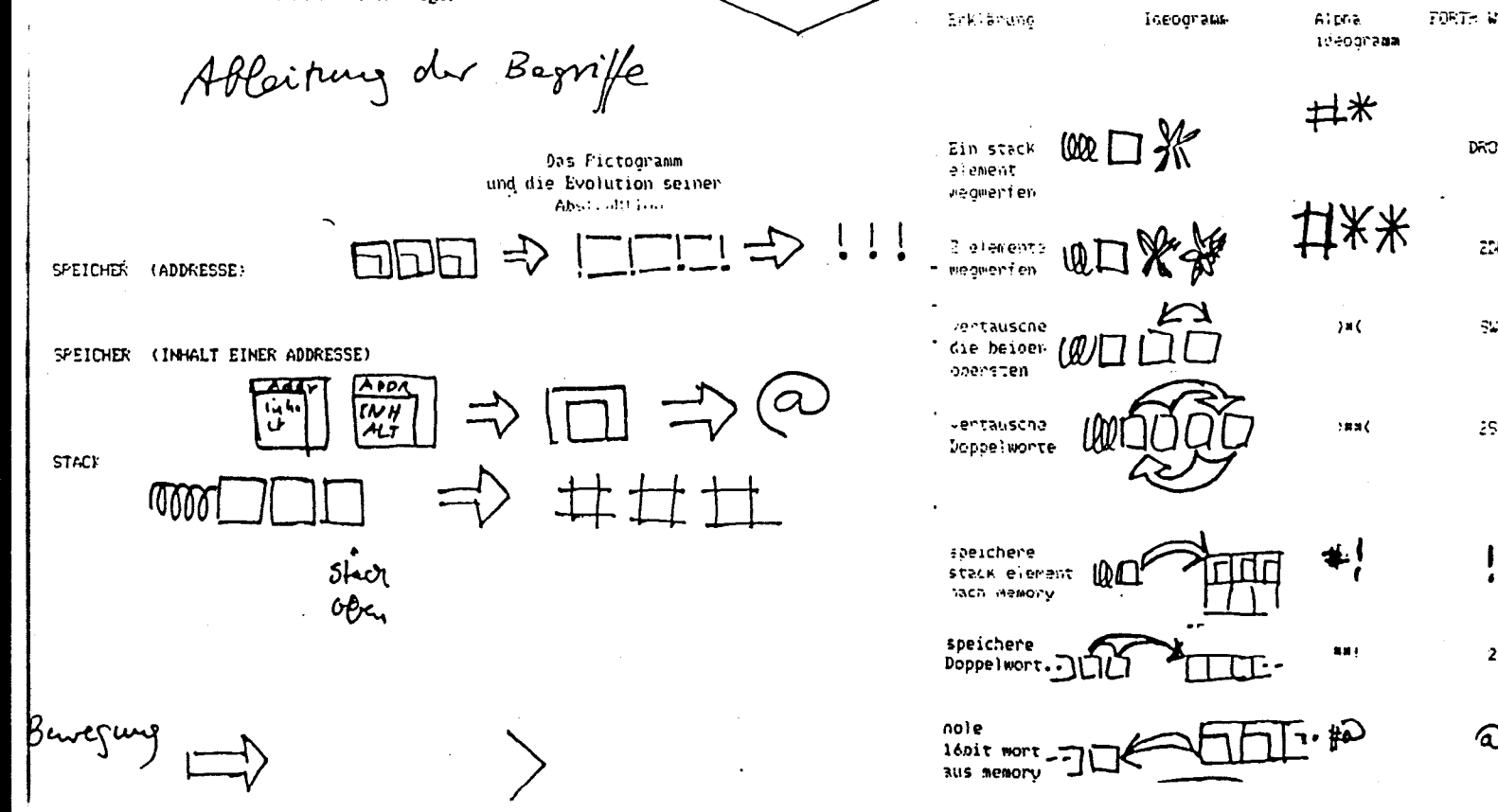


Komplexere Kommandos werden dann wieder, wie gehabt, mit normalen Worten bezeichnet. Hier haben sie wieder ihre Berechtigung, da ein Ideogramm den komplexen Zusammenhang nicht mehr deutlich machen würde.

Der Vorteil von Ideogrammen für die Primitivoperationen liegt auf der Hand: Erstens kann man auf der Basis der Grammatik ableiten, WAS die Operation tut. Zweitens ist das Kommando kurz. Es ist sogar leichter zu tippen, als die englischen Worte, weil es nur maximal 3 verschiedene Zeichen aus einem Zeichensatz von 10 Zeichen sind. Eine Ableitung aus dem vollen Diagramm ist auf einer Maschine wie LISA möglich, so daß eine solche Darstellung maximal geeignet für Schulung ist. Die intuitive Vorstellung von der tatsächlich im Computer stattfindenden Operation wird gestarkt, und der Programmierer arbeitet mit einem isomorphen Gedankenmodell des Computers. Weitere Abstraktion in höheren Ebenen der Hierarchie verhindert nicht, bei jeder auftauchenden Frage nach der Definition auf die Bestandteile zurückzugreifen.

Diese Syntax ist isomorph mit der bekannten FORTH - Syntax, und kann durch einfache Wort - Ersetzung in diese umgewandelt werden. In FORTH ist nicht das äußere Erscheinungsbild der Sprache genormt, sondern die innere Arbeitsweise, also die Funktion des Compilers.

Dank dieser Definition muß sich der Programmierer also nicht auf irgendwelche mehr oder weniger genauen oder zu dicken oder schlecht indizierten Handbücher verlassen, sondern er kann das System vor Ort analysieren.
(Literatur siehe Trends in der .....)



Ein Disassembler für den 6502

In Ergänzung zu einem Decompiler (siehe "Ein Decompiler...." in dieser Vierten Dimension) benötigt man oft eine Möglichkeit, Maschinencode listen zu können. Damit kann man sich ein Bild von den Kernal-Routinen und anderen Code-Definitionen machen. Häufig verfügt der Computer über einen eingebauten oder zuladbaren Maschinensprache-Monitor, der einen Disassembler enthält. Für alle FORTH-Freunde, die mit einem 6502-System arbeiten und keinen Disassembler besitzen, der von FORTH aus ansprechbar ist, folgt hier ein Disassembler, der vollständig in FORTH formuliert ist. Er läßt sich relativ einfach in den vorgenannten Decompiler einbinden.

Untersucht man die Opcodes des 6502 anhand einer hexadezimal sortierten Befehlsliste, so fällt sofort auf, daß alle Befehle, deren niederwertiges Bit Nr. 0 gesetzt ist, eine große Reglmäßigkeit aufweisen. Diese Hälfte der Opcoes kann man einfach weiter unterteilen: alle Opcodes, deren Bit Nr. 1 ebenfalls gesetzt ist, ($03, $07, $0B, $0F, $13,...) sind keine gültigen Befehle. Das andere Viertel der Opcodes mit gesetztem Bit Nr. 0 und nicht gesetztem Bit Nr. 1 ($01, $05, $09, $0D,...) sind die 8 Befehle: ORA, AND, EOR, ADC, STA, LDA, CMP und SBC mit je 8 Adressierungsarten (ind,X), Zero-Page, immediate, absolut, (ind),Y , Zero-Page,X , absolut,Y und absolut,X . Es gibt nur die eine logische Ausnahme: einen Befehl STA ZZ gibt es nicht.

Bei der anderen Hälfte der Opcodes mit nicht gesetztem Bit Nr. 0 läßt sich eine Regelmäßigkeit nicht so einfach feststellen. Der folgende Disassembler faßt deshalb diese Hälfte der Opcodes in der Tabelle SHORTCODE mit ihren wesentlichen Daten zusammen. Die Daten der sytematischen Hälfte der Opcodes sind in die beiden kurzen Tabellen SCODE und ADRMODE gefaßt, die von dem Wort SHORTCODE1 ausgewertet werden. Das Wort SHORTCODE0 schließlich liefert die für alle Opcodes benötigten Informationen.

Das Wort DIS schließlich disassembliert ab der Adresse, die auf dem Stack liegt, Zeile für Zeile, bis das Disassembling mit RETURN abgebrochen wird. Es benutzt dazu außer SHORTCODE einige Tabellen mit ASCII-Zeichen. Noch eine Bemerkung zur Art und Weise, wie das Wort TABELLE die Zahlen einliest: Die Übergabe der Zahlen auf dem Stack ist in FIG-6502-Systemen wegen der bedauerlich kleinen Stacktiefe nicht möglich, deshalb der mühsame Weg über BL WORD HERE NUMBER DROP.

von Georg Rehfeldt

Screen nr.:26

```
 0( Disassembler für 6502     -          30.03.84re )
 1
 2 HEX   TOOLS   DEFINITIONS
 3
 4 : TABELLE      ( +n -- )
 5               <BUILDS   0 DO
 6                         BL WORDS HERE NUMBER DROP , DROP LOOP
 7               DOES>     SWAP + COUNT SWAP C$ ;
 8
 9
10 80 TABELLE   SHORTCODE0        ( 8b1 -- 8b2 +n )
11( Tabelle folgt aus Platzgründen im nächsten Screen )
12( Vorsicht beim abtippen . )
13
14 -->
15
```

Screen nr.:27

```
 0( Disassembler für 6502     -          30.03.84re )
 1
 2Alles folgenden Zahlen sind HEXADEZIMAL.
 3Die neuen Worte werden in das ( bereits definierte !!)
 4Vocabulary TOOLS geschrieben.
 5
 6
 7 TABELLE        ( +n -- )
 8 Ein Definitionswort, das einehn Suchbegriff ins Wörterbuch
 9 schreibt, dann +n Worte durch Leerzeichen von einander getrennt
10 aus dem Eingangstext (INPUTSTREAM), in Zahlen umwandelt und
11 diese als 16b Werte ins Wörterbuch (Kapitel: TOOLS ) schreibt.
12 Ein mit TABELLE definiertes Wort braucht vor dem Aufruf die mit
13 2 multiplizierte Nummer des gewünschten 16b - Wertes und lie-
14 fert diesen getrennt in 2 8b-Werten. Oben auf dem Stapel liegt
15 das höherwertige Byte . ( 8b1 - 8b2 +n )
```

Screen nr.:28

```
 0 0810 0000 0000 0341 2510 0320 0000 0332
 1 0AC1 0000 0000 03A1 0E10 0000 0000 0362
 2 1502 0000 0741 2341 2710 2820 0732 2832
 3 08C1 0000 0000 23A1 2B10 0000 0000 2862
 4 2A10 0000 0500 2141 2410 2120 1C32 3132
 5 0CC1 0000 0000 21A1 1010 0000 0000 2162
 6 2B10 0000 0000 2941 2610 2920 1CD2 2932
 7 0CC1 0000 0000 29A1 2F10 0000 0000 2962
 8 0000 0000 3241 3141 1710 3610 3232 3132
 9 0000 04C1 32A1 31B1 3810 3710 0000 0000
10 2051 1F51 2041 1F41 3410 3310 2032 1F32
11 05C1 0000 20A1 1FB1 1110 3510 2062 1F72
12 1451 0000 1441 1541 1B10 1610 1432 1532
13 09C1 0000 0000 15A1 0F10 0009 0000 1562
14 1C51 0006 1341 1941 1A10 2210 1332 1932
15 08C1 0000 0000 19A1 2E10 0000 0000 1952     --->
```

Screen nr.:29

```
 0( Disassembler für 6502     -          30.03.84re )
 1
 2 SHORTCODE0     ( 8b1  - 8b2 +n )
 3 Ein Wort, definiert durch TABELLE, das $80 16b - Werte zur Ver-
 4 fügung stellt. Es dientzur Ermittlung von Informationen für die
 5 wenig systematischen OPcodes des 6502. Nach Aufruf liefert +n
 6 einen Index auf das Mnemonic. 8b2 trägt 2 Informationen. Das
 7 obere Nibble ( Halbbyte ) ist die NUmmer der Adressierungsart,
 8 das untere Nibbel ist die Befehlslänge in Byte - 1. 8b1 muß ei-
 9 ne gerade Zahl sein und ist der mit 2 multiplizierte Index in
10 die Tabelle. Siehe auch TABELLE
11
12
13
14
15
```

Screen nr.:30

```
 0( Disassembler für 6502     -          30.03.84re )
 1
 2 0 VARIABLE SCODE -2 ALLOT
 3 03 C,  2 C,  12 C,  1 C,  30 C,  1E C,  12 C,  2C C,
 4
 5 0 VARIABLE  ADRMOD -2  ALLOT
 6 81 C,  41 C,  51 C,  32 C,  91 C,  A1 C,  72 C,  62 C,
 7
 8 : SHORTCODE1  ( 8b1  --  8b2 +n )
 9               2/ DUP 1 AND
10               IF DROP 0 0
11               ELSE 2/ DUP 7 AND ADRMODE + C$ SWAP
12                    2/ 2/ 2/ 7 AND SCODE + C$
13               THEN ;
14( VARIABLE MODE     0 VARIABLE  LENGTH
15-->)
```

Screen nr.:31

```
 0( Disassembler für 6502     -          30.03.84re )
 1 SCODE -  ( --  addr )
 2 Eine Tabelle von acht 8b Werten, die einen Index auf ein Mnemo-
 3 nic liefern. Nur von SHORTCODE1 benutzt. Siehe dort.
 4
 5 ADRMODE   (  -- adr )
 6 eine Tabelle von acht 8b-Werten, die 2 Informationen tragen:
 7 das obere Nibbel ist ein Index für eine Adressierungsart, das
 8 untere Nibbel ist die Befehlslänge in Byte - 1. Siehe SHORTCODE
 9
10 SHORTCODE1  ( 8b1  -- 8b2 +n )
11 Ein Wort zur Ermittlung des Index auf die Mnemonictabelle,
12 genauso wie SHORTCODE0 aber nur für ungerade Zahlen.
13 Die Bytes bedeuten dasselbe wie in SHORTCODE0.
14 MODE  ( -- addr ) Variable für die Adressierungsart
15 LENGHT ( -- addr ) Variable für die Befehlslänge in Bytes - 1
```

```
Screen nr.:32
 0( Disassembler für 6502     -          30.03.84re )
 1
 2: SHORTCODE     ( 8bl -- +n )
 3                DUP 1 AND IF DUP 89 =   ( ungerade Codes ? )
 4                             IF DROP 2 THEN SHORTCODE1
 5                          ELSE SHORTCODE0 ( sonst gerade Codes )
 6                          THEN SWAP DUP 3 AND LENGHT !
 7                2/ 2/ 2/ 2/ MODE !   ;
 8
 9: TEXTAB        ( char +n 8b -- )
10          <BUILDS C, 0 DO DUP WORDS HERE COUNT
11                 >R HERE R CMOVE R> ALLOT LOOP DROP
12          DOES> COUNT >R SWAP R * + R> TYPE ;
13
14-->
15
```

```
Screen nr.:33
 0( Disassembler für 6502     -          30.03.84re )
 1
 2 SHORTCODE      ( 8b  -- +n )
 3 Ein Wort zur Ermittlung der nötigen Informationen für alle 6502
 4 OPcodes. 8b ist der 6502-OPcode. +n ist der Index auf die Mne-
 5 monictabelle. Seiteneffekte sind das setzten der Variablen
 6 LENGHT auf die zugehörige Befehlslänge -1 und der Variablen
 7 MODE auf die zutreffende Adressierungsart. s.a. SHORTCODE0 (!)
 8
 9 TEXTTAB        ( char +n 8b -- )
10 Ein Definitionswort, das einen Suchbegriff ins Wörterbuch
11 schreibt, mit 8b als Count-Byteund anschließend +n Worte aus
12 dem Eingangstext, die durch char voneinander getrennt sind, als
13 ASCII-Zeichen ins Wörterbuch schreibt. char und Count werden
14 auf alle +n Worte angewandt. Ein mit TEXTTAB definiertes Wort
15 erwartet den Tabellenindex und druckt das entspr. Wort aus.
```

```
Screen nr.:34
 0( Disassembler für 6502     -          30.03.84re )
 1
 2 BL 39 3 TEXTTAB  .MNEMONIC   ( +n -- )
 3 "BY ADC AND ASL BCC BCS BEQ BIT BMI BNE BPL BRK BVC BVS
 4 CLC CLD CLI CLV CMP CPX CPY DEC DEX DEY EOR INC INX INY
 5 JMP JSR LDA LDX LDY LSR NOP ORA PHA PHP PLA PLP ROL ROR
 6 RTI RTS SBC SEC SED SEI STA STX STY TAX TAY TSX TXA TXS TYA
 7
 8 2F 0E 1 TEXTTAB .VOR    ( +n  -- )
 9/ / /A/ /Z/#/ / /(/ (/Z/Z/ /(/
10
11 2F 0E  3  TEXTTAB  .NACH   ( +n -- )  /    /     /     /     /
12/    /,X /,Y /,X)/),Y/,X /,Y /     /)     /
13
14: 4U.R  ( u -- )   0 <# # # # #> TYPE ;
15: 2U.R  ( u -- )   0 <# # #> TYPE ;     -->
```

```
Screen nr.:35
 0( Disassembler für 6502     -          30.03.84re )
 1.MNEMONIC       ( +n -- )
 2 Die Mnemonictabelle von $39 Worten zu 3 Byte Länge, durch TEXT-
 3 TAB definiert. Druckt aus dem Index +n das entsprechende Wort
 4 aus. Die Tabelle der 6502-Mnemonics ist alphanumerisch sortiert
 5
 6.VOR    ( +n -- )
 7 Diese ebenfalls durch TEXTTAB definierte Tabelle von $0E Worten
 8 von 1 Byte Länge, enthält die VOR dem Operanden in Abhängigkeit
 9 von der Adressierungsart anzuzeigenden ASCII-Zeichen.
10
11 .NACH ( +n --.)
12 selbiges wie .VOR für die Zeichen NACH dem Operanden.
13
14 4U.R  ( u -- ) druckt u als vierziffrige Zahl mit führender 0
15 2U.R  ( u -- ) druckt u als zweiziffrige Zahl mit führender 0.
```

```
Screen nr.:36
 0( Disassembler für 6502     -          30.03.84re )
 1 FORTH DEFINITIONS
 2 : DIS  ( addr -- )  BASE § >R  HEX
 3   BEGIN TOOLS CR DUP 4U.R SPACE DUP C§ DUP 2U.R SPACE
 4         SHORTCODE >R
 5         LENGTH § DUP IF OVER 1+ C§ 2U.R SPACE THEN
 6         DUP 2 =       IF OVER 2+ C§ 2U.R SPACE THEN
 7         2 SWAP - 3 * SPACES R> MNEMONIC SPACE1+ MODE § DUP .VOR
 8         0C = IF DUP C§ DUP 80 AND IF 100 - THEN OVER + 2+ 4U.R
 9              ELSE LENGTH § DUP 2 SWAP - 2* SPACES
10                   ?DUP IF 2= IF DUP § 4U.R
11                             ELSE DUP C§ 2U.R THEN
12                        THEN  THEN MODE § .NACH LENGTH § +
13       KEY 0D =
14    UNTIL DROP R> BASE !  FORTH ;
15
```

```
Screen nr.:37
 0( Disassembler für 6502     -          30.03.84re )
 1
 2 DIS ( addr -- )
 3 Das Wort, das ab  addr disassembliert. Ein Maschinenbefehl wird
 4 in der Form :
 5       addr   OPcode   (byte) (byte) Mnemonic  Operand
 6
 7 Beispiele: 0FA5  6C 70 09 JMP  (0970)
 8            0123  EA       NOP
 9Nach dem listen eines Befehls, wartet DIS auf einen Tastendruck,
10CR bricht das Disassembling ab, jede andere Taste listet den
11nächsten Befehl in der nächsten Zeile.
12
13
14
15
```

Und nun einige Beispiele für diesen 6502 - Dissasembler :

```
" dup " :
dup compiling cfa f9e § fa0 code

0fa0 dis
0fa0 b5 00 lda z 00,x
0fa2 48  pha
0fa3 b5 01 lda z 01,x
0fa5 4c 70 09  jmp  0970
```

```
" r> " :
r> compiling cfa e7f § e81 code

0e81 dis
0e81 ca  dex
0e82 ca  dex
0e83 68  pla
0e84 95 00 sta z 00,x
0e86 68  pla
0e87 95 01 sta z  01,x
0e89 4c 77 09  jmp 0977
```

(P.S.: Alle Zahlen sind Hexadezimalzahlen. Die kleinen Buchstaben stehen als deutliches Zeichen für den C64 - Alltag. d.RED.)

Trends in der Entwicklung der Programmiersysteme

(alternativer Titel: Die stille Software-Revolution)

Teil I : Der sogenannte Anwendungsstau

Während die Fortschritte auf dem Hardwaresektor jährlich eine Verdoppelung irgendeines Systemfaktors aufweisen können, sei es Speicherdichte, oder Preis/Leistungsverhältnis, oder Geschwindigkeit, stagniert die Leistungsfähigkeit der Programmierung auf einem Niveau, das etwa in der Mitte der 60er Jahre erreicht wurde. Folglich stöhnen alle Computerhersteller und Anwender unter dem "Anwendungsstau", und alle klagen, daß sich die neuen Computersysteme viel besser verkaufen würden, wenn nur genügend qualitativ hochwertige Software vorhanden wäre.

Auf der Suche nach der Lösung des Softwareproblems werden immer mächtigere Programmiersprachen entwickelt, so zum Beispiel ADA, ein Konstrukt, das alle Eigenschaften der sprichwörtlichen "eierlegenden Wollmilchsau" aufweist. Doch trotz oder vielleicht gerade wegen solch ungeheuer komplizierter und leistungsfähiger Hilfsmittel scheint die Programmiertechnologie nicht viel schneller von der Stelle zu kommen. Was bei der Suche nach Lösungen des Programmierproblems leicht vergessen wird, ist, daß man das Problem als System ansehen muß, mit vielen ineinander versch,lungenen Faktoren, die sich nicht durch "overkill" mit noch leistungsfähigeren Werkzeugen lösen lassen.

Die Faktoren, die die Programmierproduktivität erhöhen, sind nämlich andere, als die Syntaxeigenschaften einer Sprache oder ihre Mächtigkeit. Programmierproduktivität läßt sich auf zweierlei Weise erhöhen, einerseits dadurch, daß man den Experten immer stärkere Hilfsmittel a la ADA in die Hand gibt, was zwar deren Produktivität erhöht, aber durch die immer weitere Steigerung dieser Komplexität andererseits auch den Kreis der Experten sehr klein hält, und damit die Gesamtproduktion ebenfalls sehr gering. Dies ist aber bisher der am meisten beschrittene Weg gewesen. Heutige Microcomputer-Ökonomien weisen aber in eine ganz andere Richtung. Es entstehen laufend neue Programmiersysteme, die es auch relativ untrainierten Kräften erlauben, arbeitsfähige Programme zu erstellen. Diese Lösungen versprechen auf lange Sicht die wirklich entscheidende Lösung des Produktivitätsproblems. Die Namen dieser Lösungen heißen: Interaktive Programmiersysteme und Objektprogrammierung.

Interaktive Programmiersysteme wurden anfangs in Universitäten und Forschungslaboratorien unter Namen wie "Smalltalk" und "Interlisp" entwickelt. BASIC war ein früher Vertreter eines interaktiven Programmiersystemes, der sehr schnell weite Verbreitung fand. Heute dringen sie mit Computern wie dem Macintosh als MacBASIC oder MacPASCAL in die Bereiche der normalen Anwendung vor. Objektsprachen fingen mit Systemen wie DBASE II an, entwickelten sich dann über integrierte Umgebungen wie LOTUS oder OPEN ACCESS weiter und lassen für die Zukunft Baukastensysteme für Programmieranwendungen erwarten, ähnlich wie man heute mit FISCHER-TECHNIK Baukästen-Ingenieur-Systeme aufbauen kann.

Die Problemlösung als System

Das Prozedursystem

Die Erstellung einer Problemlösung auf einem Computer ist ein komplexes System einer Mensch-Maschine-Interaktion. Die Betrachtung einer Problemlösung unter dem Aspekt einer Programmiersprache herkömmlicher Art, sei es COBOL, PASCAL oder ADA, beschränkt sich auf einen spezifischen, und heute immer mehr zurückgehenden Anteil in diesem großen Gesamtsytem. Eine Programmiersprache definiert die Prozedurinformation, die nötig ist, um ein Problem zu lösen. Die Syntaxdefinition einer Programmiersprache tut im wesentlichen nichts anderes, als den Spielraum einzuschränken, den ein Programmierer hat, um ein Problem zu lösen. Die Prozedur stellt innerhalb des Gesamtsystems einer Problemlösung eine Untermenge, oder ein Untersystem dar, das man auch das Prozedursystem nennen kann.

Das Objektsystem

Neben den prozeduralen Anteilen der Problemlösung existiert noch der große Bereich des Objektproblems. (Daher auch der oben verwendete Name Objektprogrammierung) Es ist nämlich so, daß die Objektsysteme immer eine gewisse, und normalerweise sehr komplizierte Logik aufweisen, die mit Hilfe des Prozedursystems nachgebildet werden. Der Programmierer stößt bei der Lösung seiner Aufgabe hier aber auf die Klippe, daß er als Spezialist für das Prozedursystem normalerweise wenig Ahnung von dem Objektsystem hat. Will man in der speziellen Logik des Prozedursystems nun eine Lösung für ein Objektproblem schreiben, dann stellt man sehr schnell fest, daß die logischen Probleme des Prozedursystems sich mit denen des Objektsystems mutliplizieren. Es existiert eine große Barriere zwischen Anwendern, die alles von ihrem Problem verstehen, aber fast nichts von der Art, wie es zu lösen ist, und den EDV-Kräften, die viel von Prozeduren verstehen, aber wenig von dem Problem, das zu lösen ist. Das Ergebnis ist, wie überall leicht festzustellen, aber meist schamhaft verschwiegen, oft weniger als berauschend. Heutige Ansätze für Objektsysteme gehen in die Richtung, die zur Verfügung gestellten Prozeduren in eine Logik zu bringen, die der Logik des Objektsystems angepaßt ist, dadurch wird das Prozedursystem "transparent", und es wird den Fachleuten der Anwendung möglich, ihre Problemlösung selber zu spezifizieren oder sogar zu erstellen.

Das Subjektsystem

Die herkömmliche Programmiersprachenentwicklung hat sich lange einseitig auf das Prozedursystem konzentriert und in dem Bestreben, möglichst gute Prozedursyteme zu entwickeln, die Objektbezogenheit einer Problemlösung außer Acht gelassen. Ebenso außer Acht gelassen wurde der Aspekt des Problemlösers, des Menschen. Dieser Faktor findet heute seinen Eingang als "Software-Ergonomie". Es ist ein entscheidender Faktor in der Produktivität, wie einfach oder schwierig die Interaktion des Menschen mit dem Computer/Softwaresystem ist.

Interaktivität und Ergonomie

Es wurde oben schon gesagt, daß "mächtige" Programmiertools zwar denen, die sie beherrschen, eine bessere Produktivität erlauben, aber sie sind äußerst schwer zu beherrschen, so daß **paradoxerweise die Gesamtproduktivität sinkt, je "mächtiger" die Softwaretools werden.** So zum Beispiel kann man mit der strukturierten Sprache PASCAL zwar komplexe Probleme leichter lösen, aber man muß diese Sprache erst beherrschen, und das erzeugt einen Auslese-Effekt, der die Zahl der PASCAL Programmierer stark einschränkt.

Auf der anderen Seite ist BASIC. BASIC kann sehr leicht gelernt werden, aber es ist schwer, in BASIC komplexe Programme zu schreiben, und noch viel, viel schwerer, sie auch zu verstehen. Das Problem, daß ein langes BASIC-Programm unleserlich ist, wird im Volksmund auch der "Spaghetti-Effekt" genannt, eine sehr deskriptive Beschreibung. GOTOs und GOSUBS winden sich wie Laokoon-Schlangen über Seiten und Seiten, und der geplagte Leser des Programms findet sich vor allem als hochbezahlter Seitenumblätterer bei dem Versuch, die wahre Logik des Programms zu entschlingen. Das optische Merkmal eines Spaghettigerichts ist, daß alles ein ziemlich einheitliches Gewühl von ineinander verschlungenen Faden ist. Ein BASIC-Programm steht diesem in nichts nach.

Der Interpreter: Eine interaktive Programmierumgebung

Trotz dieser offensichtlichen Mängel ist BASIC aber die populärste Sprache. Es stellt sich heraus, daß das überhaupt nicht an der Sprache, also an der Syntax liegt, sondern an BASICs Interaktivit. **BASIC ist eine interpretierte Sprache.** Das heißt, der Benutzer braucht nur eine Zeile einzutippen, und er bekommt sofort Antwort vom Computer, er bekommt ein **direktes feedback.** Wie bei allen Dingen, die man lernen kann, geht es besser, wenn man es sofort ausprobieren kann. Dadurch hat sich BASIC seinen ersten Platz unter den Computersprachen geschaffen.

Interaktivitätshindernis Nr. 1 Der ECLG Zyklus

PASCAL brachte zwar den Fortschritt der logischen Strukturierung in die Programmierung, aber es brachte gegenüber BASIC gleichzeitig einen Rückschritt. Alle bekannten PASCALs sind compiliert. Es gibt eine große Schranke zwischen dem Programmierer und seinem Programm. Diese Schranke heißt: **Der ECLG-Zyklus.** Dieses Akronym (Abkürzung aus Anfangsbuchstaben) heißt **Edit-Compile-Link-and-Go.** Ein Programm wird zuerst editiert, dann wird es compiliert, dann muß man

es linken, und endlich darf man es starten. Nur um zu erleben, daß da noch ein paar Fehler waren, und man muß wieder zurück, zum Editor, das Programm ändern, dann wieder compilieren, wobei meist noch ein paar Fehler auftreten, und so weiter ad Vergasung.

Hindernis Nr. 2 — Die Babylonische Sprachverwirrung

Jeder dieser Arbeitsschritte aus Edit, Compile, Link und Go, wird in einer anderen Umgebung gemacht. Der Editor benötigt bestimmte Kommandos, (seine eigene Syntax), der Compiler wieder andere, und das Betriebssystem, in dem all das passiert, noch einmal andere. Alles in allem muß der PASCAL-Programmierer nicht nur PASCAL können, sondern noch etwa 3 bis 6 andere Sprachen, die Spezialsprachen seines Editors, seines Betriebssystems, etc. Wie ist das zu vergleichen, mit einem Apple-Computer, wo das Applesoft-BASIC in dem Augenblick verfügbar ist, wenn der Computer angeschaltet wird, und wo man nie in einer anderen Umgebung ist, als in dieser?

Hindernis Nr. 3 — Die Zeitschranke

Es ist also nicht allein die spezielle Logik der Programmiersprache, die Probleme macht. Es ist ebenso der Umstand, daß es viele, und höchst verschiedene Umgebungen sind, in denen sich der Programmierer bewegen muß, und die er alle beherrschen muß. Und dann gibt es noch das Zeitproblem. Um den Editor aufzurufen, muß man warten. Das Editieren dauert seine Zeit. Das Compilieren dauert bei den üblichen Microcomputern meist lange genug, um einen Kaffee zu kochen, ihn zu trinken, und meist noch, um die Tasse abzuwaschen, und das Geschirr zu trocknen. Inzwischen ist das Problem, an dem man gearbeitet hat, schon ein wenig in den Hintergrund gerückt. Menschen, die lieber direkt mit dem Werkstück arbeiten, manuelle Typen, werden von einer solchen Arbeitsweise frustriert. Sie werden durch die lange Zeitspanne zwischen dem Entwerfen eines Programms, und seinem letztendlichen Lauf entscheidend in ihrer Kreativität gestört. Es ist nicht, daß solche Menschen nicht die Begabung zum Programmieren hätten, ihre Kreativit ist nur viel direkter, als etwa die eines mathematischen Typus, der sein Problem sowieso schon in voller Gänze im Kopf oder auf dem Papier gelöst hat und der den Computer in Wirklichkeit eigentlich gar nicht mehr braucht.

Hindernis Nr. 4 — Komplexitätsprobleme

Und mit dem Zeitproblem schnappt noch eine sehr große Falle zu: Da es ja ziemlich lange dauert, um eine Compilation zu machen oder um den Editor zu laden, macht man gerne viel auf einmal. Es ist nun eine alte Erfahrungstatsache: Wer viel arbeitet, macht viele Fehler. Leider ist das bei einem Programm noch schwerwiegender als bei anderen Dingen des Lebens. In der normalen Welt hat man die Dinge fein säuberlich voneinander getrennt. Hier das Werkzeug, da das Werkstück. In einem Programm aber wirken alle Bestandteile aufeinander ein. Sie interagieren, und ihre Interaktion ist komplex. Und alle Programmteile haben ihren Zweck, es gibt keine Redundanz, die irgendwelche Fehler ausgleichen würde. Es ist eine Eigenschaft eines Systems von vielen Einzelteilen, daß die Komplexität mit der Zahl der Einzelteile nicht linear wächst, sondern exponentiell. Hat man ein Teil, ist die Komplexität auch eins. Hat man zwei Bestandteile, ist die Komplexität nicht zwei, sondern vier, also viermal so viel Möglichkeiten, etwas falsch zu machen. Bei drei Teilen ist die Komplexität 9 und so weiter.

Modularität

Deshalb ist es beim Programmieren sehr gut, wenn man alle Einzelteile des Programms einzeln schreibt und austestet. Man nennt das auch **Modularisierung**, und einige noch neuere Programmiersprachen als PASCAL (zum Teil von demselben Autor) konzentrieren sich ja bekanntermaßen auf dieses Konzept. Programm-Module sollten am besten separat ausgetestet werden. Und dann sollten sie jeweils in Kombination ausgetestet werden, aber möglichst nicht alle zusammen. Nur ist eine solche goldene Regel nicht viel wert, wenn es jedesmal 10 Minuten dauert, um einen solchen Durchgang des ECLG-Zyklus zu machen, und dann schreibt man lieber eine ganze Menge auf einmal, compiliert sie und vergißt dann meist, daß das Debuggen dieses Komplexes Stunden und Tage dauert, auf jeden Fall mehr, als wenn man das Programm säuberlich in Teile zerlegt hätte, und sie einzeln getestet hätte.

So gesehen, liegen die Produktivitätshindernisse beim heutigen Programmieren ganz woanders, als erwartet. Und Abhilfe werden wir auf jeden Fall nicht durch "noch besser" strukturierte und "noch mächtigere" Sprachen bekommen, wie ADA. Um es mit einem Wort von Weizenbaum zu sagen: "Es wurde noch keinem, der im Begriff war, in ein Loch zu fallen, dadurch geholfen, daß er dies zehnmal schneller und effektiver tun konnte." (Weizenbaum, Computer Power and Human Reason)

Teil II : Die neuen Programmiersprachen

Inzwischen sind einige Firmen dabei, etwas ganz Neues zu bieten. Diese Neuheiten heißen **Mac-PASCAL, MAC-BASIC, True BASIC, BetterBASIC, Professional BASIC**. Richtig, BASIC ist stark vertreten und sogar wieder gewaltig auf dem Vormarsch. Diese alte, längst von den Experten in die Abfalltonne verbannte Sprache erweist sich als die Triebfeder des Fortschritts in der Programmertechnologie. Wie kann das kommen?

Basics Wiedergeburt

Eigentlich haben diese neuen BASICs mit den alten BASICs auf den Commodore, Tandy und APPLE Maschinen noch soviel zu tun, wie eine Dampfwalze mit einem Sportwagen. Sie haben auf einmal viel Ähnlichkeiten mit PASCAL, aber sie haben die alte Haupteigenschaft von BASIC, die Nähe zum Benutzer, nicht verloren, und sogar noch mehr dazubekommen. Die neuen Eigenschaften dieser Sprachen heißen: **Strukturierte Programmierung, inkrementelle Compilation, Graphik, Fenster, lange Namen, echte Prozeduren, dynamische Syntaxkorrektur, und, als großes Letzes: Die konsistente Programmierumgebung.**

Dies ist nun ein ganzer Haufen Technologie-Schlagworte, von denen nur einige bis jetzt überhaupt bekannt sind, und auch nur in anderem Zusammenhang. Was bedeuten sie und was hat sich da auf dem Gebiet der Programmiersprachen getan?

Strukturierte Programmierung

Dieser Begriff ist natürlich inzwischen nicht mehr ganz so taufrjsch. PASCAL ist ja die sprichwörtliche strukturierte Sprache. Neu ist, daß heute die modernen BASICs die strukturierten Konstrukte erlauben, als wären sie PASCAL. Den obligaten und sehr hinderlichen Zeilennummern von BASIC sind auch die Zähne gezogen worden: Wer will, darf sie immer noch verwenden, aber man muß nicht mehr, und als Sprungaddressen konnen Namen verwendet werden. Das gibt einem wesentlich mehr Freiheiten, zum Beispiel mit Einrücken, und vorbei sind die Tage, wo man mangels RENUMBER neu hinzugefügte Codestücke in irgendwelche obskuren Ecken des Programms drücken mußte, weil sie nicht mehr dahin paßten, wo sie eigentlich hingehörten, was den Spaghettieffekt noch verstärkte. Die Unterschiede zu PASCAL haben sich in der Tat gewaltig verringert, und man kann sagen, die neuen BASICs sind eigentlich BASCAL. PASCAL-Struktur mit BASIC-Kommandos und BASIC-Interaktivität.

Interaktivität

Und diese Interaktivität hat sich im Vergleich zu den früheren BASIC Versionen noch erhöht. Es war ja der enorme Vorteil von BASIC, daß man nur eine Kommandozeile einzugeben brauchte, und schon gab der Computer einem das Ergebnis, sei es nun richtig oder falsch. Man wußte sofort, was die Maschine auf genau die Eingabe tut, die man ihr gab. Man mußte nicht in dicken Handbüchern herumblättern, um mühselig herauszufinden, was es wohl mit diesem Befehl auf sich hatte. Man brauchte es nur auszuprobieren. Bei all der Wissenschaftlichkeit, die man der PASCAL-Programmierung anhängte, wurde doch vergessen, **daß alle Naturwissenschaft unserer Zivilisation damit anfing, daß sich jemand hinsetzte und es ausprobierte.**

Empirisches Programmieren

Unsere heutige Zivilisation ist auf die **empirische** Wissenschaft aufgebaut, **empirisch** ist das wissenschaftliche Wort für **Ausprobieren**. Und das Mittelalter, das man damals ja hinter sich gelassen hatte, war die **scholastische** Zeit, wo man auf alles spekulierte, weil es keine Moglichkeit gab, es durch den Augenschein zu beweisen. So hat die PASCAL-Programmiermethodik einen entscheidenden Nachteil, nämlich, daß man nicht sofort sehen kann, was ein Kommando bewirkt. Klar, im Prinzip kann man es, in dem man ein einzeiliges Programm entwirft, es editiert, compiliert, linkt und dann laufen läßt. Aber welcher vernünftige Mensch macht sowas schon? Und dann noch mit vielen verschiednen Kommandos. Dazu hat man normalerweise keine Zeit. PASCAL ist scholastisch, nicht empirisch. BASIC war empirisch, und hat heute nur noch dazugewonnen.

Inkrementelle Compilation

BASIC war interpretiert, das heißt, jedes Kommando wurde vom Computer, so wie es war, ausgeführt. Das dauerte natürlich wesentlich länger als mit einer compilierten Sprache wie PASCAL, und auch deshalb war BASIC für größere Programme nicht geeignet. Heute hat man einen intelligenten Weg gefunden, wie man sich um diese Klippe herumschifft: **Die inkrementelle Compilation.** Das heißt nichts anderes, als daß man jede einzelne Zeile, wenn sie eingetippt wird, schon compiliert, also in ein

optimiertes Maschinenformat übersetzt. (Es handelt sich dabei um einen Zwischencode, ähnlich dem, den ein PASCAL-Compiler erzeugt, auch p-code genannt. Dieser ist zwar nicht so schnell wie ein Maschinencode, aber doch etwa 10 mal schneller als interpretiertes BASIC). Dadurch gibt es noch eine ganze Menge anderer Vorteile.

Sofortige Syntaxprüfung

Der Compiler macht gleichzeitig bei der Eingabe die Syntaxprüfung. Also keine Fehler mehr mit falsch geschriebenen Kommandos, die der Compiler früher immer nur schön der Reihe nach finden konnte, und man so, nur um die Syntaxfehler herauszukriegen, etwa 10 Mal den ECLG-Zyklus durchgehen mußte. Das **Professional BASIC** ist sogar so hilfreich, daß es bei einem Fehler obendrein noch eine Liste aller in dieser Situation korrekten Möglichkeiten gibt. Was will man mehr ?

Variablen-anzeige, Cross-Reference

Als Hilfsmittel beim Programmdebugging sind Cross-Reference-Programme gebräuchlich. Cross-Reference heißt: ein Programm, das das Vorkommen aller Variablen auf allen Seiten des Programm-Listings auffindet, und als sortierte Liste ausgibt. Cross-Reference, oder ihr Fehlen, macht den Unterschied, ob man mit einem Programm von 10 Seiten Länge etwas anfangen konnte oder nicht. Wenn irgendeine Änderung im Programm gemacht werden mußte, dann war es unbedingt wichtig, daß man wußte, auf welche Variablen sich diese Änderung erstreckte. Somit war Cross-Reference ein unabdingbares Hilfsmittel beim Programmieren.

Die heutigen BASICs haben nicht nur ein Cross-Reference-Programm, sondern sie haben es gleich eingebaut. (Professional BASIC) Man braucht nur das Kommando FIND name geben, und schon gibt der Computer alle Programmzeilen, in denen die Variable **name** auftritt.

TRACE

Es war ein ganz besonderer Vorzug von einigen früheren BASIC-Versionen, daß man ein TRON Kommando geben konnte. Damit zeigte das Programm an, welche Zeilennummern es durchlief. Man konnte damit zumindest feststellen, wie der Kontrollfluß des Programms verlief. Allzu komfortabel war TRON nicht, denn man konnte damit noch nicht die wirklich wichtigeren Werte der Variablen erfahren, nach denen das Programm ja seinen Weg nahm.

Heute kann man ein Programm mit Hilfe der TRACE-Möglichkeit in ein paar Minuten testen. Man kann es nicht nur komplett mit all seinen Variablen darstellen, so daß man immer genau weiß, wo das Programm und aus welchem Grunde genau das tut, was es tut, sondern man kann es noch anhalten, selber per Hand die Variablen verändern und dann wieder starten. Professional BASIC kann sogar einen Programmablauf rückwärts machen.

Fenster, Prozeduren, Programmierumgebung

Als ob all diese Neuigkeiten nicht schon genug für eine kleine Revolution des Programmiergeschäfts wären, aber es kommt noch mehr. Man brauchte ja nur das Konzept der LISA weiter umzusetzen, und alles, was sich da in Richtung Programmieren verwerten ließ, auch einzusetzen. LISA hat Fenster. Wie kann man Fenster nutzbringend in der Programmierung einsetzen?

Fenster

Ohne ausdauerndes Blättern im Programmlisting ist heute Programmierung nicht zu machen. Wenn man irgendein Stück Code schreibt, dann bezieht sich dieses Stück Code erfahrungsgemäß immer auf ein paar andere Stücke, die mindestens 10 oder 20 Seiten voneinander in dem ganzen Listing verstreut sind. Ebenso erfahrungsgemäß hat man gerade kein aktuelles Listing zur Hand, so daß man genötigt ist, erst einmal wieder aus dem Editor herauszugehen, den Drucker anzustellen, die zehn oder zwanzig Minuten zu warten, bis sich der Matrixdrucker durch das Listing hindurchgequält hat, und dann darf man weiterprogrammieren, nur um nach etwa einer halben Stunde wieder vor demselben Problem zu stehen, denn man hat jetzt so viel verändert, daß das alte Listing nichts mehr taugt.

Mit Fenstern geht es viel einfacher. Da holt man sich die Stücke Code, die man im Augenblick gerade braucht, dazu, und schreibt die Routine fertig, an der man arbeitet. Das geht natürlich nur dann einfach, wenn man das oben erwähnte Suchkommando hat, das einem auch schnell alle Vorkommnisse einer bestimmten Variable auffindet. Denn sonst wäre das Auffinden aller relevanten Textstellen am Bildschirm allein auch sehr mühselig.

Fenster sind ungeheuer wichtig, wenn man die oben erwähnten Möglichkeiten des TRACE und andere ausnutzen will. Da ja solche Zusatzinformation nicht zu dem eigentlichen Programmlisting gehört, ist es optisch viel besser, wenn man diese Information abgesetzt präsentiert, eben in einem Fenster.

Prozeduren und Informationsverdichtung

Fenster sind allerdings nicht in allen Fällen sofortige Erleichterung beim Programmieren. Wer sich ein BASIC oder PASCAL Listing auf dem Bildschirm ansieht, der stellt fest, daß man kaum irgendwelche relevanten Zusammenhänge eines Programms auf einem Schirm darstellen kann. Programmiersprachen stellen gewöhnlich immer nur ein Kommando pro Zeile dar. Bei den heutigen Bildschirmen mit 24 Zeilen kommt da nicht sehr viel heraus. Zumindest ein BASIC löst aber auch diese Problem: Durch echte Prozeduren (Better BASIC).

Wenn man nämlich in einer echten Prozedur etwa 20 Zeilen Kommandos auf einen einzigen Namen abkürzen kann, (und dieser Name kann 10 oder 20 Zeichen haben, also wirklich aussagen, was diese 20 Zeilen tun), dann kann man auf einmal einen Extrakt von 10 Seiten Programmlisting auf einen Bildschirm bekommen, und damit ist man in der Lage, auch sehr komplexe und umfangreiche Programme souverän zu handhaben. **Better BASIC** erlaubt nicht nur Prozeduren, sondern auch Module. Damit hat man eine gewisse Form eines variablen Compilers, ein Konzept, zu dem später noch etwas gesagt werden soll.

Weitere Eigenschaften

Andere Möglichkeiten der neuen Sprachen sind mächtige Graphik-Befehle, in denen zum Beispiel Kommandos für einen Bildaufbau gegeben werden können, wie weiter oben erwähnt wurde, Hilfsfunktionen an jeder Ecke und Kante des Systems, automatische optische Formatierung (pretty printing), integrierter Editor, Datentypen. Die PASCALs der modernen Fertigung sind bis jetzt noch nicht erwähnt worden. Das Mac-PASCAL nutzt alle Benutzer-Eigenschaften des Macintosh aus und ist damit als Programmiersystem ebenso leistungsfähig wie das BASIC, wenn es auch scheint, daß es nicht wesentlich besser ist.

Fortsetzung folgt....

Literatur

Programmiersprachen und -Entwicklung

ACM,Proceedings of the ACM SIGSOFT/-SIGPLAN, Software Engineering Symposiumon Practical Software Development Environments, Peter Henderson, ed.
Pittsburgh, Pennsylvania, April 23-25, 1984

BACK78, Backus, John W., Can Programming be Liberated from the Von Neumann Style? Comm. ACM, 1978
BYTE Magazine, April 1984, p. 298-344, 6 Articles on new BASICs
Is BASIC Getting Better? / True BASIC / Better-BASIC / Macintosh BASIC / Professional BASIC / BASIC-09
BYTE Magazine, June 1984, p.136, Macintosh PASCAL
BYTE Magazine, December 1983, p.155, Making Life Easier for Professional and Novice Programmers, (Article on MICRO FOCUS COBOL)
Real Programmers Dont Use PASCAL, Datamation, 2 oder 3 84 (?)

über Systeme

BAT79, Bateson, Gregory, Mind and Nature, a Necessary Unity, E. P. Dutton, 1979
Stafford Beer:
BEER59, Cybernetics and Management, Wiley 1959, 60
BEER66, Decision and Control, John Wiley & Sons, Chichester & New York, 1966
BEER72, Brain of the Firm, Allen Lane, 1972
BEER79, The Heart of Enterprise, John Wiley & Sons, Chichester & New York, 1979

WIE48, Wiener, Norbert, Cybernetics, MIT Press, 1948, 1961
WEI75,Weinberg, Gerald, General Systems Thinking, Wiley, 1975

Diverses

Andreas Goppold:
GOP83-1, Micro-Computer als kultureller Faktor, (to be published)
GOP84-1, IBM oder PC, Microcomputerwelt 4, 5, 6 -84 p. 20

HOF79, Hofstadter, Douglas R., "Goedel, Escher, Bach", Basic Books, 1979
HOF81, Hofstadter, Douglas R., "The Minds I", Bantam Books, New York, 1981

Literatur zu FORTH

BRODIE, Brodie, Leo, Starting FORTH, Prentice-Hall, Englewood Cliffs, NJ 07632, USA
BOUT80, Boutelle, Jerry, "Does a FORTH Metacompiler represent a Metaprocess, or a Process of Self-Reference?", 1980 FORML Proceedings, p.110
BYTE MAGAZINE, August 1980, Issue on: Threaded Interpretive Languages
GOP-PROG, Goppold, Andreas, Was ist FORTH ?, FORTH GESELLSCHAFT DEUTSCHLAND, Juni 1984
GOPINT, Goppold, Andreas, Das Paradigma des Interaktiven Programmierens, FORTH GESELLSCHAFT DEUTSCHLAND, Juni 1984
HAY81, Haydon, Glen, "FORTH and the Nature of Ideographic thought", 1981 Rochester FORTH Standards Conference, p. 81
FORS81, Forsley, Lawrence P., "What Is FORTH?", 1981 Rochester FORTH Standards Conference, p. 75
MOO74, Moore, Charles, FORTH: A New Way To Program A Mini-Computer, 1974, Astron. Astrophys. Suppl 15, 497-511
KOGG82, Kogge, Peter M., An Architectural Trail to Threaded Code Systems, March 1982, IEEE Computer, p.22
ZECH84, Zech, Ronald, Die Programiersprache FORTH, Franzis, München 1984

AUS DER TRICKKISTE

z.B. ASCII

Bei der Programmierung von Menüs braucht man häufig den ASCII- Wert eines Zeichens. Anstatt nun jedesmal in der Tabelle nachzusehen und dann einen nichtssagenden Zahlenwert ins Programm zu schreiben, ist es sinnvoller, einen Befehl zu definieren, der ein nachfolgendes Zeichen automatisch in ASCII umgerechnet auf den Stack legt. Wenn dieser Befehl ASCII heißt, (der Autor benutzte hier die Bezeichnung $, im 83er Standard ist es jedoch bereits unter der Bezeichnung ASCII bekannt.) dann schreibt man also ASCII A statt 65 . Die folgende Definition bewerkstelligt das:

```
: ASCII  BL WORD HERE 1+ C§ ÄCOMPILEÜ LITERAL ;
IMMEDIATE
```

Das Wort holt das nächste Wort vom Input-Stream, pickt den ersten Buchstaben heraus und legt es auf den Stack. Wenn ASCII während des Kompilierens aufgerufen wird (es ist ja ein IMMEDIATE-Wort, d.h. es wird auch während des Kompilierens ausgeführt !), dann wird der auf dem Stack befindliche Wert als Literal in die gerade übersetzte Definition geschrieben. Der Effekt für die kompilierte Definition ist genau derselbe, als wenn man den ASCII-Wert direkt in den Quelltext schreiben würde.

z.B. RECURSE

In FORTH sind Rekursionen standardmäßig nicht möglich, weil ein Wort vom Compiler erst erkannt wird, wenn es fertig kompiliert ist. Manche Probleme lassen sich aber nun mal besser rekursiv darstellen. Diesem Mangel kann jedoch leicht abgeholfen werden:

```
: RECURSE  ( ruft laufende Definition rekursiv auf )
        ?COMP LATEST PFA CFA , ; IMMEDIATE
```

(Rekursion darf auf 6502-Systemen nicht ohne besondere Beachtung des Returnstacks angewendet werden.)

Der Befehl RECURSE bewirkt also einen Aufruf der Colondefinition , in der er auftritt. Er macht dies, indem er die Codefield-Adresse der laufenden Definition als Aufruf an die gerade aktuelle Speicherstelle schreibt. Vorher wird mit ?COMP noch geprüft, ob gerade kompiliert wird — außerhalb einer Colondefinition ergäbe RECURSE nämlich keinen Sinn.
RECURSE kann bedenkenlos auch innerhalb von Kontrollstrukturen (IF THEN, Schleifen, CASE etc.) verwendet werden — auf eines sollte man jedoch achten: Die Schachtelungstiefe darf nicht beliebig groß werden, da früher oder später (je nach Implementation) einer der beiden Stacks überläuft. Aber zehn bis zwanzig Ebenen dürften allemal drin sein, und mehr braucht man höchst selten.

Um gleich mal die Verwendung von RECURSE zu demonstrieren, folgende Definition:

```
: DEZ  ( dezimalzahl — 16-bit Integer )
        BASE § /MOD DUP IF RECURSE 10 * + ELSE DROP
THEN ;
```

DEZ bewirkt, daß eine 16-bit Zahl auf dem Stack so umgerechnet wird, als ob man sie dezimal eingegeben hätte — gleichgültig, unter welcher Zahlenbasis man sie tatsächlich eingegeben hat.. Wozu das? Nun: Ich fand es immer lästig, daß z.B. bei 10 LIST mal Screen 10 und mal Screen 16 erscheint — je nachdem, ob man dezimal oder in hex arbeitet. Die Definition
: LIST DEZ LIST ; löst das Problem. Jetzt kann ich mich darauf verlassen, daß auch wirklich die (dezimale) Screennummer kommt, die ich haben will — egal mit welcher Zalenbasis ich gerade arbeite.

Für diese TIPS & TRICKS sind wir DR. med.dent. Greiner höchst dankbar.
Natürlich könnt ihr alle hier eure kleinen "Helferlein" , die jeder von uns im Verlaufe seiner Forth-Tätigkeiten aus Faulheit entwickelt, loswerden und darauf hoffen , daß auch andere sie zu schätzen wissen.

Ein Decompiler für F.I.G.-FORTH

von Georg Rehfeldt (unserem C64 - Spezi)

Wie sicherlich die meisten von Euch wissen, erzeugt der FORTH- Compiler beim Schreiben von neuen Wortdefinitionen (z.B. : 2DUP OVER OVER ;) ins Wörterbuch zunächst einen Kopf, der außer dem Namen des Wortes (2DUP) und seiner Länge (4) noch andere Angaben enthält. Zunächst gibt es einen Zeiger auf ein vorhergehend compiliertes Wort; dieser Zeiger wird zum Durchsuchen des Wörterbuchs verwandt. Dann existiert noch ein Zeiger auf ausführbaren Maschinencode, der vom inneren Interpreter benötigt wird. Die Adresse dieses Zeigers heißt Compilationsadresse des Wortes (CFA in FIG-FORTH). Außer dem Kopf wird in der Regel noch ein Rumpf ins Wörterbuch geschrieben. Bei einer Colon-Definition, ein Wort, das mit ":" eingeleitet und in der Regel mit ";" abgeschlossen wird, werden die Compilationsadressen der nach dem Namen folgenden Worte (OVER OVER) im Wörtebuch nacheinander abgelegt. Eine ":"-Definition besteht also aus dem Kopf und einer Liste von Compilationsadressen, die den Rumpf bildet. Der innere Interpreter von FORTH kann aus diesen Zeigern bei der Ausführung des Wortes " 2DUP " die Worte " OVER OVER " wiedererkennen, denn jedes FORTH-Wort ist durch seine Compilationsadresse eindeutig gekennzeichnet.

Diese Tatsache macht sich der hier gezeigte Decompiler zunutze. Er ist in der Lage, compilierte Worte wieder lesbar auszulisten. Dies ist zur Fehlersuche in eigenen Worten ganz nützlich, eignet sich jedoch auch zur Analyse schlecht dokumentierter FORTH Versionen und hilft nicht zuletzt dem Anfänger beim Verständnis des Compilationsprozesses in FORTH.

Nicht alle FORTH-Worte compilieren einfach durch Ablage ihrer Compilationsadresse im Wörterbuch, es gibt zum Beispiel die IMMEDIATE-Worte, in der Regel Anweisungen an den Compiler, die etwas anderes ins Wörterbuch schreiben als ihre eigene Compilationsadresse. (So compiliert IF ein 0BRANCH, ELSE ein BRANCH, jeweils mit nachfolgender, relativer Sprungadresse und THEN überhaupt nichts ins Wörterbuch.) Eine ausführliche Beschreibung aller Sonderfälle führt hier zu weit, das Decompilieren selbstformulierter Worte zeigt das Verhalten dieser besonderen Worte am besten.

Die Bedienung des Decompilers ist einfach, einzutippen ist z.B.:

```
DEC 2DUP    (RETURNTASTE)
```

und der Decompiler antwortet mit :

```
2DUP
CFA 1234 INH 5678

1236 OVER
1238 OVER
123A ;S
```

Es gibt einige Worte, die nicht mit ;S oder (;CODE) enden, wie ABORT und QUIT, weil das Ende dieser Worte niemals erreicht wird. Bei diesen Worten kann der Decompiler das Ende des Wortes nicht erkennen und das Listing muß, wie beim vorzeitig gewünschten Abbruch, mit ?TERMINAL beendet werden.
Ein automatischer Decompiler hat einen großen Nachteil: Bei Erweiterungen des Compilers wie z.B. durch selbstdefinierte Definitionsworte, müßte der Decompiler entsprechend erweitert werden. Dies muß der Anwender jedoch selbst tun, oder die mit den grundlegenden Worten aus Screen Nr. 13 geschaffenen einfachen interaktiven Worte in Scren Nr. 17 benutzen, um solche Besonderheiten zu decompilieren.

Viel Spaß mit FORTH. Georg Rehfeld

Screen nr.:7

```
 0( Tools dummy def. ==              04.08.1984 Georg Rehfeld ) 0-->
 1
 2forth definitions hex
 3vocabulary tools immediate
 4tools definitions
 5
 6: col.def. :
 7
 80 constant const.
 90 variable var.
100 user usr.
11
12: == ( cfa -- )        2+ nfa  id.  space ;
13
14: case? ( n1 n2 -- tf / n1 ff ) over = dup if swap drop then ;
15-->
```

Screen nr.:8

```
 1
 2      Die folgenden Worte werden als "TOOLS" zusammengefasst.
 3      Dies sind Hilfsworte, die nur zu vergleichen gebraucht
 4      werden.
 5      == druckt aus der Compilationsadresse eines Wortes sei-
 6      nen Namen.
 7      Dank an Klaus Schleisiek, von dem die Worte "case?" und
 8      die Idee zu .clit, .string, .lit, .branch und .name
 9      stammen.
10      Wenn n1 = n2, dann hinterlässt case? "wahr", sonst
11      bleibt n1 erhalten und oben auf dem Stapel ist "falsch".
12      Danach Rücktehr ins Aufrufende Wort.
13
14
15
```

Screen nr.:9

```
 0( Tools                            04.08.1984 Georg Rehfeld ) 0
 1
 2: .clit   ( adr -- adr+1 )   count . ;
 3
 4: .string ( adr1 -- adr2 )   count 2dup dup . type + ;
 5
 6: .lit    ( adr -- adr+2 )   dup @ dup + u. 2+ ;
 7
 8: .branch ( adr -- adr+2 )   dup @ over + u. 2+ ;
 9
10: .name   ( adr -- adr+2 )   dup @ == 2+ ;
11
12
13
14
15-->
```

Screen nr.:10

```
 1
 2    .clit     - druckt das adressierte Byte als vorzeichenlose
 3                Zahl.
 4    .string   - druckt den count und den adressierten String.
 5
 6    .lit      - druckt die adressierte Zelle als vorzeichenlose
 7                und vorzeichenbehaftete Zahl.
 8    .branch   - druckt die adressierte Zelle als absolute
 9                Sprungadresse und als vorzeichenlose Zahl.
10    .name     - nimmt die adressierte Zelle als Compilations-
11                adresse eines Wortes und druckt daraus den
12                Namen des Wortes.
13
14    Bei allen Worten dieses Screens wird adr entsprechnd um 1,
15    2 oder mehr erhöht.
```

Screen nr.:11

```
( Tools  besonders?                 10.08.1984 Georg Rehfeld ) 0
 1
 2: besonders?   ( adr1 -- adr2 )
 3       dup 2+ swap @ ' compile   cfa case?
 4                     if .name    ;s  then
 5                     ' clit      cfa case?
 6                     if .clit    ;s  then
 7                     ' (.")      cfa case?
 8                     if .string  ;s  then
 9                     ' lit       cfa case?
10                     if .lit     ;s  then
11                     dup  ' 0branch  cfa =
12                     over ' branch   cfa = or
13                     over ' (loop)   cfa = or
14                     over ' (+loop)  cfa = or
15                     if drop .branch ;s then drop ; -->
```

Screen nr.:12

```
 1
 2    besonders? - prüft ob eine Sonderbehandlung für das gerade
 3                 Compilierte Wort erforderlich ist.
 4
 5       Nach compile, wird das nächste Wort in der gleichen
 6       Zeile gedruckt. Zu clit wird das folgende Byte vorzei-
 7       chenlos und zu lit die nächste Zelle sowohl vorzeichenlos
 8       als auch vorzeichenbehaftet ausgegeben. Bei (.") wird
 9       der count und der String ausgegeben. Bei branch,0branch,
10       (loop) und (+loop) wird die Sprungadresse nicht relativ,
11       sondern absolut angegeben.
12
13       adr1 wird entsprechend erhöht, sodaß adr2 auf die
14       nächste cfa zeigt.
15
```


Screen nr.:13

```
 0( Tools  identify                  04.08.1984 Georg Rehfeld ) 0
 1: liste ( adr -- )
 2       begin   cr dup u. .name 2- dup @ >r besonders?
 3               r  ' ;s         cfa =
 4               r> ' (;code)    cfa = or
 5               ?terminal or
 6       until   drop ;
 7
 8: identify ( pfa -- )
 9       dup cfa @ ' col.def. cfa @ case? if liste  ;s then
10       ' const. cfa @ case? if ." constant " @ u. ;s then
11       ' var.   cfa @ case? if ." variable " @ u. ;s then
12       ' usr.   cfa @ case? if ." user " c@ 10 +origin @ + @
13                                                 u. ;s then
14       ' tools  cfa @ case? if ." vocabulary " drop ;s then
15       = if ." code " ;s then ." ???" ;            -->
```

Screen nr.:14

```
 1
 2       liste  - listet ':' -Definitionen aus den Compilations-
 3                adressen. adr wird gedruckt, adr @ wird als cfa
 4                verstanden und daraus der Name gedruckt. Dann
 5                werden Besonderheiten geprüft, siehe besonders?
 6                Abbruch erfolgt mit ?terminal oder nach ;s und
 7                (;code). Achtung, bei Worten, die nicht mit ;s
 8                enden, muß rechtzeitig mit ?terminal abgebroch-
 9                en werden!
10
11    identify  - ermittelt die Art des Wortes und veranlaßt bei
12                ':' -Definitionen das Listen des Wortes, bei
13                User-,Variablen und Constanten wird der Inhalt
14                ausgegeben.
15
```

Screen nr.:15

```
 0( Tools  kopf, dec                 04.08.1984 Georg Rehfeld ) 0
 1
 2: kopf  ( pfa b -- pfa )
 3       cr cr over nfa id. space 40 and if ." immediate " then
 4       dup cfa dup ." cfa " u. ." inh " @ u. cr ;
 5
 6forth definitions
 7
 8
 9: dec   ( -- )
10       -find if tools kopf identify
11              else ." not found " then cr ;
12
13
14
15-->
```

Screen nr.:16

```
 1
 2       kopf   - druckt den Namen des zu Decompilierenden
 3                Wortes und einen Hinweis, wenn es "immediate"
 4                ist.
 5                Außerdem wird die Compilationsadresse und deren
 6                Inhalt gedruckt.
 7
 8
 9
10
11
12       dec    - decompiliert das folgende Wort, sofern es ge-
13                funden wird.
14
15
```

Screen nr.:17

```
 0( Tools                            04.08.1984 Georg Rehfeld ) 0
 1tools definitions
 2: n     ( adr -- adr+2 )
 3       dup u. .name   quit ;
 4
 5: l     ( adr -- adr+2 )
 6       dup u. .lit    quit ;
 7
 8: b     ( adr -- adr+2 )
 9       dup u. .branch quit ;
10
11: c     ( adr -- adr+2 )
12       dup u. .clit   quit ;
13
14: s     ( adr -- adr+2 )
15       dup u. .string quit ;     forth definitions    ;s
```

Screen nr.:18

```
 1
 2       n      - druckt adr und den Namen aus adr @ .
 3
 4       l      - druckt adr und adr @ als 16 bit Zahl.
 5
 6       b      - druckt adr und adr @ als Sprungziel bezogen auf
 7                adr .
 8       c      - druckt adr und adr c@ als 8 bit Zahl.
 9
10       s      - druckt adr und den adressierten String.
11
12       siehe auch Screen Nr.13
13
14       Hochhals, dank an Klaus !
15
```

ALTER SCHÜTZT VOR WANZEN NICHT ....

...und wenn es auch das Herze bricht. Diese bedauerliche Erfahrung, die ich bei der Installation von fig.FORTH auf einigen 8080/8085 und 6800 Computern machen durfte, möchte ich mit den folgenden Zeilen publik machen, um so die eine oder andere böse Überraschung vermeiden zu helfen.

Zuerst zum 8080 :

In Anlehnung an die bekannten fig.Dokumente habe ich vor geraumer Zeit eine PROM-fähige Sprachversion für den 8085 mit etlichen Hilfsmitteln und Erweiterungen für eine Industrie-Appliation erstellt, die nach ausgiebigen Tests mit den üblichen, demoralisierenden Intermezzos und nach mehreren erfolgreichen Applikationen fehlerfrei zu sein schien. Das erte Gesetz von Murphy für Software besagt jedoch: " Es gibt keine fehlerfreie Software". Der Beweis wurde angetreten. Als eine weitere Aplikation realisiert wurde, mußte erstmalig wegen ihres Umfangs auch in der oberen Hälfte des Adreßraumes gearbeitet werden. Bei der Gelegenheit ergab sich, daß die von fig für den 8080 angegebene Version von FORGET sich im Adreßraum über 8000H recht unfreundlich verhält (es funktioniert schlicht und einfach nicht). Grund hierfür ist ein Adressenvergleich, der mit dem " < "-Operator durchgeführt wird. Der Vergleich CFA < FENCE geht damit schief, wenn beide Adreßraumhälften involviert sind. Die Angelegenheit war durch Überschreiben des "<"-Operators mit "U<" zu bereinigen (Label ULESS anstelle von LESS).

Daß die Forth Interesst Group diesen antiken Prozessor offenbar noch niemals in einer größeren Applikation gesehen hat, ist verständlich. Ich mußte aber bei einer anderen Gelegenheit feststellen, daß dieses FORTH wohl auch noch niemals etwas anderes als 8"-Disketten und CP/M gesehen hat. Jedenfalls lautet das zweite Gesetz von Murphy für Software: " there's allways one bug more ". Der Beweis wurde ebenfalls angetreten. Als ich in einer neuen Applikation einen anderen Massenspeicher anzubinden hatte, wobei mit Blockgrößen ab 256 Bytes gearbeitet werden mußte, funktionierte der äußere Interpreter nicht mehr. Der Fehler liegt in ENCLOSE . Eine Möglichkeit, auch diese Wanze zu vernichten, ist im anliegenden Assemblerlisting dargestellt.

Zum 6800 :

Auch dieser Veteran kann leider immer noch keine fehlerfreie fig.FORTH-Dokumentation vorweisen. Wenn schon nicht fehlerfrei, so wäre doch zumindest Vollständigkeit des fig.Vokabulars nett. Leider fehlt das "M*" . Es empfiehlt sich, dies in Anlehnung an das 8080-Listing nachzuflicken, wobei aber beim 6800 das Fehlen der 32-bit-Erweiterungen 2DUP zu vermerken ist. Soviel zum ersten Murphyschen Gesetz für Software.

Infolge des zweiten Murphy-Gesetzes wurde dringend nach einem weiteren Fehler gesucht, wobei ich im Wort /MOD fündig wurde. Dieser Befehl funktioniert nach dem fig.Listing nicht, weil dort innerhalb von /MOD ein Aufruf von U/ erfolgt, während M/ richtig wäre. Vermutlich liegt die Ursache hierfür darin, daß M/ in dem Listing fehlt (bitte auch +- und D+- beachten !). Nach Erweiterung der 6800-fig.FORTH-Implementierung um dieses fehlende Wort sowie überschreiben von U/ hiermit in /MOD funktionierte, /MOD aber immer noch nicht. Das führte zu der Auffindung eines weiteren Fehlers im fig.Listing, diesmal im Befehl S->D Das Wort ist hier (obgleich im Installationsmanual in Screen 56 als Primitiv definiert) als Secondary realisiert worden, wobei aber eine bedingte Verzweigung nebst Literal 'verschütt' gegangen ist. Fügt man diese nachträglich ein, so ist der Befehl /MOD nun endlich fehlerfrei.

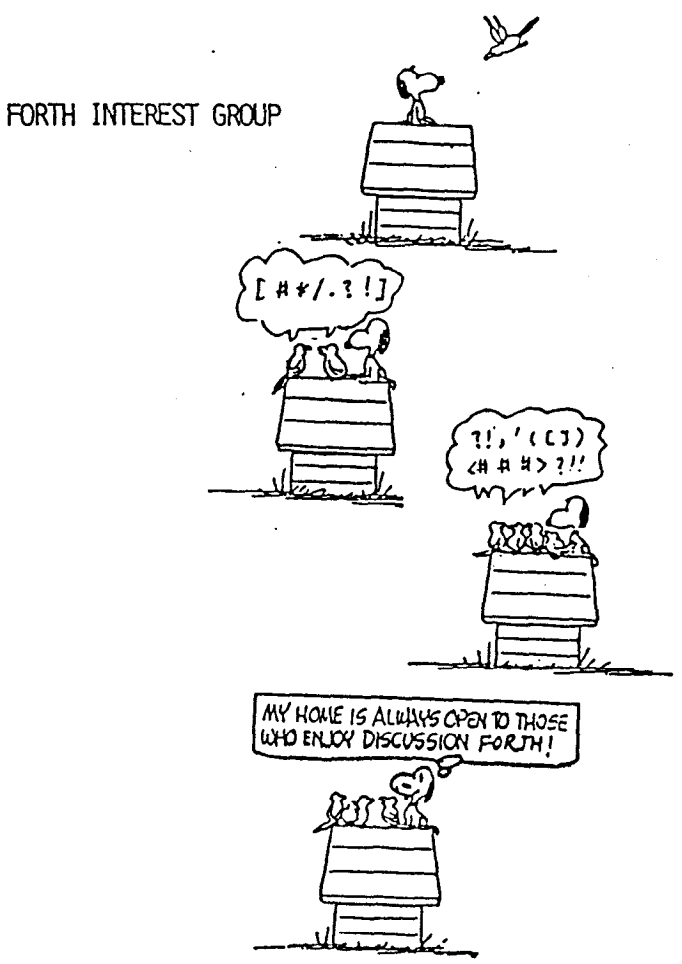
Allerdings ist hier eine grundsätzliche Anmerkung zu machen, die die high-level/low-level-Balance speziell dieser fig.Implementation betrifft. Der Versuch, mit high-level-Definitionen Platz zu sparen, hat bei elementaren Prozeduren natürliche Grenzen, die man aus Laufzeitgründen tunlichst vermeiden sollte. Darüber hinaus ist es im konkreten Falle so, daß eine saubere Assembler-Implementierung nicht einmal mehr Platz beanspruchte als die (korrigierte) high-level Prozedur.

Abschließend gebe ich daher das Wort S->D für den 6800 in Assembler formuliert an :

```
                count
                S->,D + 80
STOD            (* + 2)
                TSX
                LDAB #  255
                LDAA 0,X
                BMI  STOD1
                INCB
STOD1           TBA
                JMP PUSHBA
```

möchte aber abschließend den evtuell aufkommenden Eindruck wegwischen, die beiden fig.FORTH-Implementierungen seien generell unzuverlässig. Sowohl die 8080- als auch die 6800-Version sind im Laufe der Zeit ausgiebig getestet worden und funktionieren (mit den genannten Modifikationen) in einer Reihe von Applikationen nunmehr völlig einwandfrei.

Ronald Zech



DIE GOLDENE REGEL:

Sollte ihr Computer aus unerklärlichen Gründen allzu häufig abstürzen: Suchen Sie den Fehler immer genau dort, wo Sie am sichersten sind, daß er NICHT ist.

```
0192 4C
0193 4F
0194 53
0195 C5                 DB      'E'+X'80'
0196 4001               DW      PFIND-9
0198 9A01       ENCL    DW      *+2
019A 2A427B             LHLD    RPP             ; CHANGE FROM 8080 F.I.G. ENCLOSE
019D 2B                 DCX     H               ; ============================
019E 2B                 DCX     H               ; GREAT MASS-STORAGE-BUFFERS NEED A
019F 22427B             SHLD    RPP             ; COUNT CAPACITY GREATER 255    -->
01A2 71                 MOV     M,C             ; USE BC REG. ( BUT SAVE BEFORE USE )
01A3 23                 INX     H               ;
01A4 70                 MOV     M,B             ;
01A5 D1                 POP     D
01A6 E1                 POP     H
01A7 E5                 PUSH    H
01A8 7B                 MOV     A,E
01A9 57                 MOV     D,A
01AA 01FFFF             LXI     B,X'FFFF'       ; INITIALIZE CHR.OFFSET COUNTER TO -1
01AD 2B                 DCX     H
                                                ;
01AE 23         ENCL1   INX     H
01AF 03                 INX     B
01B0 BE                 CMP     M
01B1 CAAE01             JZ      ENCL1
                                                ;
01B4 C5                 PUSH    B
01B5 7E                 MOV     A,M
01B6 A7                 ANA     A
01B7 C2C101             JNZ     ENCL2
01BA 03                 INX     B
01BB C5                 PUSH    B
01BC 0B                 DCX     B
01BD C5                 PUSH    B
01BE C34803             JMP     SEMIS+2
                                                ;
01C1 7A         ENCL2   MOV     A,D
01C2 23                 INX     H
01C3 03                 INX     B
01C4 BE                 CMP     M
01C5 CAD201             JZ      ENCL4
01C8 7E                 MOV     A,M
01C9 A7                 ANA     A
01CA C2C101             JNZ     ENCL2

01CD C5         ENCL3   PUSH    B
01CE C5                 PUSH    B
01CF C34803             JMP     SEMIS+2

01D2 C5         ENCL4   PUSH    B
01D3 03                 INX     B
01D4 C5                 PUSH    B
01D5 C34803             JMP     SEMIS+2

01D8 84                 DB      X'84'
01D9 45                 DB      'EMI'
01DA 4D
01DB 49
01DC D4                 DB      'T'+X'80'
01DD 8E01               DW      ENCL-X'0A'
01DF 1605       EMIT    DW      DOCOL
01E1 FFFF               DW      -1
```

8085 fig FORTH korrigiertes ENCLOSE
(akzeptiert Block-größen ≥ 256 Bytes)

8086 FORTH

Wenn Sie Ihren DC 32/16 voll kontrollieren wollen . . . Wenn Sie sich für Grafik, Spiele, Kommunikation, Robotsteuerung, Datenerfassung oder Prozeßkontrolle interessieren . . . dann sollten Sie FORTH benutzen.

- FORTH ist genauso interaktiv und gesprächig wie BASIC, aber 50 mal schneller.
- FORTH Programme sind hoch strukturiert und modular, dabei einfach zu warten.
- FORTH gibt totale Kontrolle über alle Interrupts, Ein/Ausgabe, Bausteine und Speicherstellen
- Das Anwendungsprogramm kann als sofort ablauffähiges Programm übersetzt und ohne Lizensgebühr vertrieben werden.
- Billige Ergänzungen sowie freie Telefonunterstützung
- Schnellstes FORTH für 8086. Läuft auf Floppy wie auch auf Harddisk.

Das 8086 FORTH Paket enthält den FORTH Interpreter/Compiler mit virtueller Speicherverwaltung und Background multitasking, einen voll Bildschirm orientierten Wordstar kompatiblen Editor, einen 8086 Assembler mit Intel Mnemonics und lokalen Labels, einen Rückübersetzer, Testhilfsmittel, Tools und viele Demoprogramme. Das 180 seitige Handbuch besteht aus einer generellen Benutzer-Einführung, Gebrauchsanweisungen für Editor und Assembler, einem technischen Überblick über die Interna des FORTHes, einer einführenden Beschreibung mit Bibliographie sowie einer kompletten Wortbeschreibung.

8086 FORTH gehört zu den ersten FORTH Systemen, die durch Benutzung, der vom Betriebssystem zur Verfügung gestellten Files, total hardwareunabhängig sind. FORTH Programme und Daten können auf die Diskette gleichzeitig mit anderen Applikationen abgespeichert sein und mit den Betriebssystem-Hilfsmitteln kopiert bzw. gesichert werden. Das FORTH Vocabular wurde um Zusatzfunktionen wie Betriebssystemzugriff, Datumsabfrage oder Recordlocking erweitert.

Software-Entwickler: Mit dem FORTH Cross Compiler können Sie spezielle Disk- oder ROM- gestützte FORTH Applikationen für die verschiedensten Microprozessoren mit dem DC 32/16 als Entwicklungsrechner erzeugen. Für mit dem Cross Compiler erzeugte Programme muß keine Lizensgebühr bezahlt werden.

8086 FORTH	**DM 376.20**
Aufrüstbar auf 8086 FORTH+	
8086 FORTH+	**DM 991.80**
Programmlänge bis zu 1 MByte	
FORTH Cross Compiler	**ab DM 1250.—**
Choose target microprocessor from Z-80, 8080, 8086/88, IBM PC, 68000, 6502.	

DEMO Disk	**DM 20.—**
Extension Packages	
Advanced Color Graphics	DM 376.20
Intel 8087 Support	DM 376.20
Software Floating Point	DM 376.20
Curry FORTH Programming Aids	DM 752.40
B + Tree File and Index Manager	DM 752.40
Quad Precision Integer Math Pack	DM 98.—

Wir führen selbstverständlich auch FORTH-Compiler für C-64, VC-20, APPLE, Z80 CP/M sowie diverse andere Mikrocomputer.

Das FORTH-Literaturangebot von über 6500 Seiten erhalten Sie weiterhin über uns.

FORTH – SYSTEME
Angelika Flesch
Schützenstr. 3 · Tel. 07651/1665
D-7820 TITISEE-NEUSTADT

Produkte - Neuigkeiten - Innovationen

Unter dieser Rubrik hoffen wir euch in der Zukunft Informationen über neuere Entwicklungen am FORTH-Markt liefern zu können. Vorausgesetzt, es gibt genügend Anbieter, die uns mit verständlichen und zutreffenden Produktinformationen versorgen. Damit wir auch richtig verstanden werden: Diese Seite) soll keine Schleichwerbung für Ladenhüter o.ä. darstellen, Anzeigen können zu günstigen Tarifen im C.I.Alpha geordert werden. Auch Privatpersonen die etwas anzubieten haben, natürlich nur wenn es mit FORTH in Verbindung steht, sind herzlich eingeladen, ihr Produkt kritisch zu bewerten und uns eine Beschreibung desselben zuzusenden. Daneben bieten sich natürlich auch Kleinanzeigen an.
Wegen der vielen c64-User die sich bei uns gemeldet haben werden diesmal nur 64-FORTHe vorgestellt.

FORTHSYSTEM-MODUL FÜR C-64

Für den Commodore64 existiert eine Modulversion, die in den dafür vorgesehenen Einschub passt. Die Version für den C-64 enthält ein Programm mit der Länge von 16 Kbyte. Das Programm liegt ab $8000 bis $BFFF. 64FORTH hat über 500 eingebaute Wörter mit denen Sprites, und die Tonerzeugung gesteuert werden kann. Das Modul enthält einen Editor mit dem Source-Screens auf dem Bildschirm editiert werden können. 64FORTH verwendet Standard Screens mit 16 Zeilen a 64 Zeichen und ist damit voll Standard kompatibel. Auf dem Bildschirm wird dies mit Rechts oder Links scrollen der Zeilen erreicht. Der eingebaute 6502 Assembler erlaubt auch Maschinenroutinen in Forth einzubinden. Die Diskette wird natürlich ebenfalls unterstützt, dabei werden relative Files zum abspeichern der Screens benutzt. Dies bedeutet daß auch andere Files auf der gleichen Diskette sein können und Screens keine Programme überschreiben können. Zur besseren Programmerstellung können Worte auch im Single Step abgearbeitet werden. Ein Spriteeditor erlaubt einfaches erstellen von Single und Multicolorsprites. Diese Sprites können dann auch auf Diskette unter einem Namen abgespeichert und wieder eingelesen werden. Im Bereich $C000 bis SFFFF werden dabei auch 16 Screens zwischengespeichert, um den Datenverkehr zwischen Rechner und Diskette zu minimieren. Die Hiresgrafik wird nicht unterstützt, sie muß vom Anwender selbst definiert werden. Dieses Modul läuft nicht auf dem SX64.

FORTHSYSTEM FüR VC20

Das Forthsystem für den VC20 besteht aus einem ROM Cartridge für den Modulbereich. Es enthält ein Programm mit 8 KByte Länge. Es kann schon mit der 5 K Version laufen größerer Speicherplatz ist jedoch von Vorteil. Ein Screen orientierter Editor erleichtert das Eingebeben von eigenen Programmen. Disketten werden ebenfalls unterstützt. Um nach dem Buch Starting FORTH lernen zu können, werden die Unterschiede dargestellt. Dieses FigFORTH System ist eine grundlegende Version mit 250 Worten, der Anwender kann sie auf seine eigenen Bedürfnisse zuschneiden.

Die beiden Module stammen von der amerikanischen Firma HES.

64 FORTH AUF DISKETTE

Für 99,-DM ein vollständiges FigFORTH mit deutscher Beschreibung. Diese Diskette wird im VC1541 Format geliefert, und enthält außer dem Forthcompiler über 100 Kbyte an FORTH Software. Ein Forthstyle 6502 Assembler und ein Screeneditor sowie viele Demobeispiele für FORTH erleichtern das Erlernen dieser aufregenden Sprache. Mit einem sehr guten Grafikpaket können auch einfach und wirkungsvoll die Vorteile von FORTH erlernt werden. Die Geschwindigkeit der Grafik und die Einfachheit der Ansteuerung wird auch Sie begeistern. Das FORTH System kann sowohl absolut als auch relativ wie ein TURTLE die Grafik erzeugen. Das Forthsystem schaltet beim Anlaufen die Basicroms ab, deshalb steht für das Programmieren der Platz bis $D000 zur Verfügung. Für eigene Programme stehen deshalb 36 KByte zur Verfügung. Bei Benutzung der Grafik wird der Bereich auf 88000 limitiert. Um Commodore Dosfiles zu lesen sind Befehle vorhanden. Ein vorhandenes paralleles IEC-Bus Interface wird ebenfalls unterstützt. Das Handbuch beschreibt jedoch nur die Eigenheiten des 64FORTH, es wird empfohlen ein FORTH-Lehrbuch (z.B. Starting FORTH) zu benutzen.

Diese Systeme sind zu beziehen über die Firma FORTH-SYSTEME FLESCH, Schützenstr. 3, 7820 Titisee-Neustadt. Mitglieder der Forth-gesellschaft bekommen 5%-Rabatt. Bei Bestellung bitte die Mitgliedsnummer angeben.

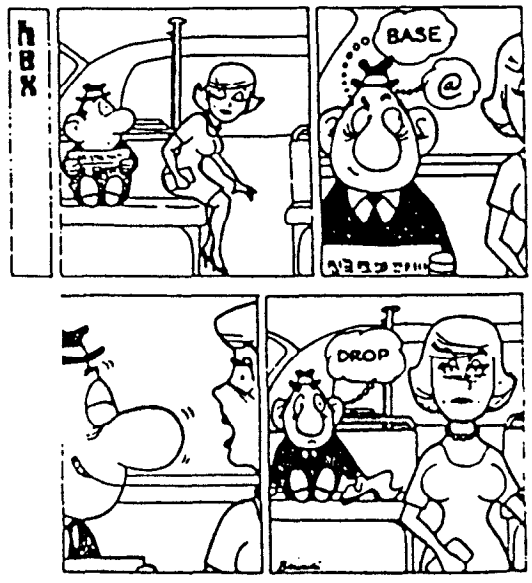
FIG-FORTH64 von compu-lab

FIG-FORTH64 ist eine in vielen Teilen erweiterte Version des von der FIG vorgeschlagenen Standards. Es enthält einen Editor, einen Assembler (in FORTH - Notation) und ein komfortables String-Paket. Mit diesen Werkzeugen ist es möglich Spracherweiterungen beliebiger Komplexität vorzunehmen. Ein Floating-Point - Paket ist in Vorbereitung.

Der Editor:
Der Editor von FIG-FORTH64 entspricht voll dem FIG-Editor. Er verarbeitet bis zu 100 Screens mit 16 Zeilen 'a 64 Zeichen. Davon werden bis zu 5 Screens im Rechner gehalten. Alle Screens sind in einem einzigen Relativfile gespeichert, so daß der Anwender auf die editierten Texte auch mit Basic-Programmen zugreifen kann.

Der Assembler:
Ein weiterer Zusatz des FIG-FORTH64 ist der Assembler. Mit ihm wird es dem Anwender ermöglicht, alle besonders zeitkritischen Routinen in Maschinensprache zu programmieren und diese sofort und mit den Mitteln des FORTH zu testen. Der Assembler verwendet selbsverständlich die übliche UPN-Notation des FORTH.

String-Handling:
FIG-FORTH64 enthält ein Paket, das ähnlich wie in BASIC die Verarbeiotung von Zeichenketten erleichert. Es sind Funktionen wie Konkatenation, Längenermittlung, Substitution, Zahlen-Zeichenketten- und Zeichen-Zahlenkettenumwandlung sowie die aus dem BASIC bekannten Befehle LEFT$, RIGHT$ und MID$ realisiert.
compu-lab Erlangen, Fürther Strasse 50a, 8520 Erlangen zu beziehen. Mitgliedern der Forth Gesellschaft werden 5%-Rabatt auf deren FORTH-Produkte eingeräumt. Bitte die Mitgliedsnummer angeben.



Forth Gesellschaft ** INTERN **

Unter dieser Rubrik wollen wir, das Team vom C.I.A und sonstige an der Organisation Beteiligte, euch jeweils Aktuelles, Erfreuliches & Problematisches aus dem Hintergrund unserer Gesellschaft mitteilen. Damit hoffen wir uns einerseits Luft zu machen und EUCH anderseits die Gelegenheit zu geben, auf wichtige Entscheidungen einzuwirken.

EXPANSION: Die FORTH Gesellschaft (BGB) ist offensichtlich auf dem richtigen Dampfer. Oder wie sollen wir den 555 %igen Mitgliederzuwachs innerhalb der letzten 4 Monate erklären ? Stolze 90 FORTHler aus der BRD, Schweiz und Östereich haben sich zur FORTH Gesellschaft bekannt. Wenn ihr diese Zeilen lest dürften wir schon 100 Forthler sein. Apropos: Da wir zu faul sind sog. Mitgliedsbestätigungen auszustellen geben wir euch folgenden Geheimtip: Jeder der auf seinem Adreßkleber ein " M " in seiner Mitgliedsnummer findet, ist auch ein solches. Wer meint das er ein " M " zuwenig hat, der sollte sich beschweren oder aber eine Überweisung auf unser Konto vornehmen.

NEUE ADRESSE !!!
Wem diese Tatsache bisher entgangen ist, den weisen wir hiermit nocheinmal ausdrücklich darauf hin, daß das COMMON INTERFACE ALPHA seinen Standort geändert hat. Die neue Adresse lautet:

COMMON INTERFACE ALPHA
SCHANZENSTRASSE 27
2000 HAMBURG 6
Um die Verwirrung perfekt zu machen: Die alte Telefonnummer ist weiterhin gültig. Die ist nämlich mit uns umgezogen !

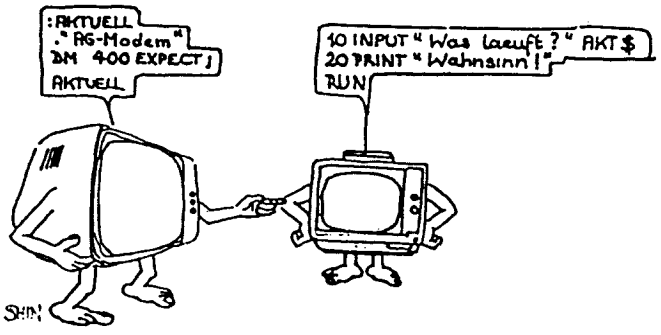
ÜBERWEISUNGEN: Einige von euch werden es sicher selber schon festgestellt haben: Der Forth Gesellschaft Geld zu überweisen ist manchmal gar nicht so einfach (wie es unserer Meinung sein sollte). Gelder wurden und werden von der Post zurückgewiesen, trotz diese an die FORTH GESELLSCHAFT gesendet wurden. Daß es uns gibt, kann man ja deutlich sehen (resp. lesen). Einzig die Post weigert sich an unsere Existenz zu glauben, und das nur weil wir kein mit amtlichem Stempel versehenes Papier über die Bestätigung unserer Existenz vorweisen können. Dieser Misere soll ja auch auf dem Dachsberg - Treffen abgeholfen werden. Aber solange bitten wir euch alle Gelder auf das SONDERKONTO 4, Horst-Günter Lynsche, Schanzenstr. 27, 2000 Hamburg 6 zu überweisen. Wir bitten euch weiterhin von Postanweisungen oder Einschreibebriefen abzusehen, oder diese an Horst-Günter Lynsche zu adressieren. Der hat nämlich im Gegensatz zur Forth Gesellschaft einen Personalausweis !

INTERAKTION: Gute Kontakte zur Firma **FORTH SYSTEME FLESCH** und persönliche Gespräche zwischen Klaus Flesch und A. Goppold (unserem Meta-Programmierer) haben dazu geführt, daß alle FORTH - Gesellschafts **Mitglieder 5 % - Rabatt** auf deren Produkte bekommen. Traritrara.....
Nötig ist dazu nur, daß ihr eure Mitgliedsnummer vollständig angebt. Eure Mitgliedsnummer ist die erste " Zahl " auf eurem Adresskleber.
Dem wollte denn auch die Firma **COMPU LAB** nicht nachstehen und bietet ebenfalls 5 % Rabatt für alle Mitglieder der FORTH - GESELLSCHAFT auf deren FORTH-Produkte an. (siehe auch Rubrik: Neue Produkte)..

PUBLIC DOMAIN: Für alle die nicht geduldig genug sind um auf die erste Version unseres Programmiersystems LEIBNIZ zu warten und unbedingt ein Forth im 83er Standard haben wollen, eine freudige Nachricht. Das F83-Model von Laxen $ Perry ist für die Prozessoren 8080,8086 sowie 68000 erhältlich. Es wird auf 8" CP/M-Standard Disketten gegen Einsendung von 44,- DM plus 2 Disketten geliefert. Mit etwas Mühen ist dieses Forth auch auf einige 51/4 " Formate gebracht worden. Dieses Vergnügen ist etwas teurer (66,- DM) und erfordert eine vorherige schriftliche Anfrage im C.I.A. Bitte gebt euer Format an, oder sendet uns eine formatierte Diskette mit einem Probetext darauf ein.Vieleicht schaffen wir es auch euer Format zu implementieren.

Das erste deutsche FORTH - TREFFEN
Einige von euch werden es schon wissen. Am 27. und 28.10 1984 findet im Zentrum Dachsberg am Niederrhein das erste deutsche Forth Treffen statt. Themen sind u.a.:
-Konsolidierung der Forth Gesellschaft als eingetragener und hoffentlich gemeinnütziger Verein.
-Großes Planungs- und Informationsmeeting
-Am Nachmittag/Abend des 28.10 ein Treffen der an der COMPUTER GILDE Interessierten

Diese Gelegenheit die Begründer der Forth Gesellschaft kennzulernen oder selber einer von ihnen zu werden und die Möglichkeit sich mit Spezialisten im lockeren Plausch zu unterhalten, sollte sich eigentlich keiner entgehen lassen. In der Hoffnung, daß dieses Heft noch vor dem 5.10 erscheint:
Anmeldeschluss: 5.10 1984 Danach gibt es keine Gewähr für freie Plätze.
Kosten: 80.- DM (incl. Unterbringung in Mehrbettzimmern und Verpflegung). Einen Schlafsack o.ä. müßt ihr selber mitbringen. Gegen Aufpreis werden natürlich alle anderen Unterbringungswünsche erfüllt. (z.B. Flughafenabholung und Unterbringung im Hotel) Wer eine Mitfahrgelegenheit sucht oder hat : Bitte im C.I.A melden, wir versuchen bis zu einer Woche vorher alles zu arrangieren.



KONTAKTE: Auch zu diesem Thema tut sich einiges, daher haben wir uns entschlossen, für Chapter-Gründungs-willige folgendes Procedere vorzuschlagen: Jedes Mitglied der FORTH - Gesellschaft, das bereit ist, eine Ortgruppe (Chapter) zu gründen, möge uns bitte eine Adresse senden, unter der er/sie Anfragen entgegennimmt. Diese wird dann in der VIERTE DIMENSION veröffentlicht. Wir hoffen, daß auf diese Weise mehr Kontakte zustande kommen, die uns davor schützen, Anfragen wie " was wißt ihr denn alles über FORTH - Compiler speziell mit FLP für den Prozessor XYZ ", schriftlich beantworten zu müssen. Ehrlich gesagt, tun wir das auch (noch) nicht, eben weil wir so stark wachsen! Aber auf Dauer ist das ja auch kein Zustand. Wir fänden es toll, wenn wir solche Anfragen lokal verteilen könnten. Also: Freiwillige vor....

Apropos: Jeder, der mal sehen möchte, was denn nun tatsächlich " Hinter den Kulissen" geschieht und bereit ist, beim Verschieben und Einrichten derselbigen Hand anzulegen, also zum Beispiel selber Anfragen zu beantworten, der kann dies ab sofort **JEDEN MONTAG IM Common Interface Alpha tun.** Ihr seid herzlichst eingeladen zur Entwicklung der FORTH Gesellschaft beizutragen. Also dann: Bis nächsten Montag. Ach ja, sollte wider Erwarten doch jemand kommen: Ein kleiner Anruf vorher ist immer ganz nützlich. Dann wissen wir, was auf uns zukommt, und ihr, ob wir nicht gerade Urlaub in "Grande Balkonia" machen.

LESERBRIEFE

In einer unserer August-Aussendung hatten wir nach euren " ersten Erfahrungen in FORTH "gefragt. Tatsächlich bekamen wir fünf Antworten darauf. Hier einige Auszüge:

Erste Erfahrungen in FORTH

Günther Scholz aus Tiefenbronn schrieb sinngemäß:
Im April dieses Jahres besorgte ich mir ein FORTH-System für meinen Z80-CP/M Computer. Ich wußte noch nicht viel über FORTH, was mir gefiel, war die Nähe zur Assemblerprogrammierung, welche mir aus meiner Zeit als Nachrichtentechniker (Digital- + Mikroprozessortechnik) bekannt war. Ich entschloß mich also, das LMI - FORTH von Flesch zu kaufen. Um die Wartezeit zu überbrücken, kaufte ich mir das Buch " Die Programmiersprache FORTH" von Zech. Zunächst einmal verstand ich gar nichts. Später dann noch weniger. 1.) ich bin kein Systemprogrammierer, und der Aufbau der Hochsprachen war mir bis dahin völlig unbekannt. Und warum muß ich schon bei meinen ersten Schritten in einer neuen Programmiersprache wissen, wie diese aufgebaut ist und wie der Code im Speicher abgelegt wird ? Also dachte ich mir, Du hast dir das falsche Buch gekauft. Auf der Suche nach weiterer Literatur stieß ich auf zwei englische Bücher, in deutsch war nichts zu finden. Und was ich dort fand, entsprach im wesentlichen dem, was Herr Zech geschrieben hatte. Nach zwei frustrierenden Wochen war mir eins klar geworden: FORTH in Deutschland ist eine Programmiersprache für Spezialisten.
In der Zwischenzeit erhielt ich das bestellte FORTH mit einem schönen Ordener, alles sehr dekorativ. Ich war überwältigt. Vor allem als sich mein FORTH das erste mal mit "ok" meldete. Aber nach ein paar simplen Rechenoperationen und anderen eindrucksvollen Spielchen war ich mit meinem Latein so ziemlich am Ende. Gut, ich nahm also mit neuem Elan und guten Mutes das Buch von Zech in die Hand und begann in Kapitel 3.

Ich führte alle dort gezeigten Beispiele durch und lernte so immer mehr verwirrende FORTH - Worte kennen. Und dann kam (ein Wunder war geschehen) nach drei Monaten der Augenblick, wo ich mein erstes Programm oder besser gesagt selbstdefiniertes FORTH - Wort fehlerfrei laufen ließ, welches beliebig viele Screens auf dem Drucker mit Seitennummerierung ausgibt. Was für ein Augenblick.
Fassen wir also zusammen: Es dauerte fast drei Monate, um ein kleines Programm in FORTH zu schreiben. Ich glaube hier ist für die FORTH - Gesellschaft viel zu tun. Wenn sich FORTH in Deutschland verbreiten soll, so muß erst einmal ein Weg gefunden werden, wie man die Erlernung der Sprache vereinfachen kann. Sie werden mir doch wohl recht geben, wenn ich sage, daß die Syntax von FORTH für Anfänger doch recht kompliziert ist. (NEIN!! siehe Antwort) Vor allem das Umdenken in UPN - Technik erwies sich als gewöhnungsbedürftig.
Ich schätze, daß Anfänger im Programmieren, ohne Erfahrungen in Hardwaretechnik und Assemblerprogrammierung, hier doch hoffnungslos überfordert sein werden. Auch das AHA-Erlebnis , eine sehr wichtige Funktion für den Lernenden kommt hier zu kurz, da man sehr viele Worte lernen muß, um irgendein kleines Programm zu schreiben. Vor allem vergißt man die Syntax, (! § . U. R usw.) sehr schnell. Man muß halt immer eine Liste aller Worte aufgeschlagen haben, als Erinnerungshilfe.
Auch das Literaturdefizit in deutscher Sprache wird wohl nicht dazu beitragen das FORTH sich schnell verbreitet. Mir sind auch Fälle bekannt, wo Firmen die FORTH einführten, es auch schnell wieder absetzten, spätestens dann, wenn der Programmierer der Firma diese verließ, da es ja kaum Fachleute in dieser Sprache gibt. Und dann bekam ich letzte Woche ihre Einladung zum FORTH - Treffen. Sofort war mir alles klar: Sie sprechen hier nur Leute an, die FORTH kennen und damit programmieren, die den Durchblick haben. Vor allem die professionelle Art der Jobs usw. zielen ja auf Leute ab, die mit FORTH ihr Geld verdienen wollen. Also ein Verein von Jungunternehmern ?
Ich hoffe sie zerknüllen hier nicht den Rest meiner Ausführungen. Dies soll keine Kritik für ihre Gesellschaft sein, sondern als Diskussionsbeitrag verstanden werden. Ich glaube nämlich nicht, daß ich der Einzige bin,der diese FORTH - Erfahrungen gemacht hat, (herausgefunden bei anderen FORTH - Anfängern,die nach zwei Wochen frustriert aufgaben.)
Darum eine Bitte: Bringen sie mehr für Programmierneulinge in Sachen FORTH. Vielleicht sollten sie sogar zwei Ausgaben der VIERTE DIMENSION herausgeben. Eine für Neulinge und eine für FORTH - Profis.
Na ja, trotz der vielen Schwierigkeiten komme ich jetzt langsam mit FORTH klar. Aus diesem Grunde bin ja auch ihrem Verein beigetreten. Sie sollten vor allem nicht alles so tierisch ernst nehmen, was ich bisher geschrieben habe, vieles ist auch mehr satirisch gemeint.
(Sodann beschreibt Herr Scholz noch ein spezielles Problem .....)

Antwort: Was soll man dazu sagen ? Nun, zunächst einmal muß ich Herrn Scholz recht geben. Auch bei uns hat es drei Monate gedauert, bis wir unser erstes Programm fertig hatten. Das war eine Datenerfassungsroutine für Literatur, mit der wir jetzt die FORTH PUBLIC DOMAIN SCREENS erfassen. (Formblatt FIG 001) Es hat ca. 40 Screens und kann soviele Datensätze verwalten, wie unsere Diskette fassen kann. Damals sagte ich mir: Nicht kleckern, sondern klotzen. Was die Syntax von FORTH betrifft, so stimme ich Herrn Scholz nur in Bezug auf die Kürzel zu (z.B. § ! ." etc). Obwohl, auch hier kann man sich mittels Umbenennung dieser Worte leicht behelfen. Die meisten FORTH - Standard - Worte finde ich jedoch descriptiv genug, um mit ihnen zu arbeiten. Und für die, die der englischen Sprache nicht mächtig sind, wird LEIBNIZ hoffentlich nützlich sein.
Allerdings bezweifle ich, daß Herr Scholz für alle Anfänger spricht, wenn er sagt, daß diese es wohl noch schwieriger haben würden als er. Schließlich hat er ja schon Hardware entwickelt und Assembler programmiert. Weiß also längst Bescheid. Und genau hier irrt er sich. WIRKLICHE ANFÄNGER, also solche, die vorher kaum oder keinerlei Kontakt zu Computern hatten, lernen (insbesondere) FORTH sehr viel leichter und auch schneller als " Profis" aus anderen Bereichen der Computerei. Meiner Meinung nach liegt das daran, daß Anfänger völlig unvorbelastet an die Sprache herangehen und ihr keinerlei Konzepte aufdrängen wollen. Ebendazu neigen aber die Profis, suchen in FORTH bekannte Konzepte und sind frustriert, wenn diese nicht vorhanden oder anders implementiert sind. Sie gehen mit einer bestimmten Erwartungshaltung an FORTH heran.
In diesem Falle hat Herr Scholz eigentlich einen intelligenteren Assembler erwartet (das geht zumindest aus dem Rest des Briefes hervor), und weniger eine IN SICH ABGESCHLOSSENEN PROGRAMMIERUMBEBUNG. Warum althergebrachte Konzepte in FORTH nicht zu finden sind, lese man in A. Goppolds Artikel: Trends in der Entwicklung ... in diesem Heft nach.
Daß Anfänger es leichter haben ergibt sich auch daraus, daß jene "simplen Spielereien " für die meisten Anfänger in sich schon ein Erfolgserlebnis bilden. Wir alle kennen das: Der Computer macht, was ich von ihm verlange. Solche Erfolgserlebnisse stellen sich für die "Profis" natürlich erst ein, wenn sie ein "umfangreiches und irgendwie nützliches Programm" geschrieben haben.
Daß Anfänger leichter und schneller lernen haben wir hier in Hamburg letzes Jahr selber feststellen dürfen. In der Praxis hat sich gezeigt, daß Änfänger sogar "besseren" Code produzieren können, als Fast-Profis, einfach weil sie den Kopf nicht voller "Basic-Spaghetti" oder ähnlichem haben und direkter denken.
Demnach bräuchten wir eigentlich DREI Ausgaben der VIERTE DIMENSION: Eine für die Profis, eine für die "Verbildeten" und eine für die "Ungebildeten". Den Profis kann leicht geholfen werden. Unsere amerikanische Schwesternzeitschrift **FORTH DIMENSIONS** , ist eine unerläßliche Quelle, wenn man wissen will, was sich an der vordersten FORTH - Front alles tut. Daher auch die Empfehlung von uns, sich als Mitglied bei der FIG USA eintragen zu lassen. Für US 27,-/Jahr bekommt man dann alle zwei Monate eine absolut tolle $ informative Zeitschrift ins Haus. (Adresse: FORTH Interest Group, PO BOX 1105 * San Carlos, CA 94070). Den anderen hoffen wir mit der VIERTE DIMENSION und sonstigen Aktionen hier in der BRD zu helfen.
z.B. wird das Literaturdefizit unseren Informationen nach von einigen Verlagen in Angriff genommen. Die bereits für Juni angekündigte Übersetzung des Starting FORTH von Leo Brodie, soll nun endgültig im Oktober erscheinen (HANSER - Verlag, München, für DM 48.- ist dieses Buch natürlich im Common Interface Alpha erhältlich). Weitere Übersetzungen werden folgen.
Andreas Goppold bringt in der EDITION ARAGON zum Frühjahr nächsten Jahres das Buch " FORTH - Ein Programmiersystem ohne Grenzen" heraus. Und im Herbst erscheint vom glei-

chen Autor in der cf eine Serie mit dem Titel: "Systematisch Programmieren in FORTH".
Wir vom C.I.Alpha haben mit einer Liste der gängigen PUBLIC DOMAIN SOFTWARE, die dann als FOTOKOPIEN bei uns zu beziehen sein werden, einen Anfang getan, um den Neulingen 1.) Einblick in den Programmierstil anderer FORTHler und 2.) von Anfang an einen Pool wichtiger Utilities - zu bieten.
Jungunternehmer sind wir natürlich, jung wie wir sind und bei dem, was wir so unternehmen. Dennoch: Glauben sie im Ernst von 90 * 23,- DM pro Jahr läßt sich eine superschnelle allumfassende Verwaltung am Leben erhalten ? Ohne die Mitglieder die, wie Sie, freiwillig mehr als 23,- DM zahlen oder die an die Forth - Gesellschaft spenden, sowie etwas profitableren Aktionen als diese Zeitschrift es ist, gäbe es gar keine FORTH GESELLSCHAFT.
Auch sie sind angehalten, diese Zeilen mit Humor zu versehen. Und für ihre kritischen Anmerkungen sind wir Ihnen dankbar. Jedoch sollten sie bedenken, daß wir für die Umsetzung ihrer Anregungen vor allem zwei Dinge brauchen:
1. viel Zeit und 2.) weiteres FEEDBACK von allen Beteiligten.
Und darum: Vielen Dank, Herr Scholz.

Horst Kroker aus Mainz schreibt:

Seit ich Computerbesitzer bin (1980) war mein Hauptinteresse die Systemprogrammierung. (Zur Beachtung: Erst kommt das Interesse, die Fähigkeiten kommen irgendwann später). Daher beschloß ich schon früh FORTH zu implementieren, da ich den Eindruck hatte, daß FORTH im Aufbau einfach sei. Man sollte vielleicht noch erwähnen, daß ich FORTH für die geeignetste Sprache zum Betrieb von "Krüppelkomputern" (Zitat nach K. Schleisiek) wie ich einen hatte (Sharp MZ80-K jetzt: MZ700), halte.
(Hinweis für Kritiker: Jedesmal wenn ich diese Behauptung in einem Computerclub aufstelle, ist da ein anderer der sagt: Ja, ich weiß daß die Möglichkeiten von FORTH am besten mit Diskettenstation genutzt werden können. Aber OHNE ? Nun, ich bin der Meinung auch für Systeme ohne Diskstation ist FORTH die geeignetste Sprache, weil: schnell, kompakt und flexibel !!)
Bei der Implementation mußte ich nun feststellen, daß der Sourcecode nicht in die 48 kbyte Hauptspeicher passte. Aufgrund meiner damals nicht ganz ausreichenden Englischkenntnisse verstand ich weder wie bei diesem Assembler ein (drei)-geteiltes Programm bearbeitet wird, noch wie die Implementation mittels eines minimalen FORTH-Systems (siehe Implentationmanual) funktionieren soll. Letzteres verstehe ich heute noch nicht.
Nach mitterweile 3 Assemblern, 5 mal kompletten Source eintippen und 2 Jahren läuft mein fig-FORTH. Allerdings ohne Diskettenstation, bisher ohne Tapesupport (also nur im directmode), ohne vollen ASCII - Standard und ohne einige notwendige Änderungen, um im Kassettenbetrieb sinnvoll arbeiten zu können. (z.B. Abspeichern des erweiterten FORTH, Redefinieren früherer Worte (siehe hierzu Jupiter Ace Editor.)
Diesen Stand erreichte ich am 24.12.83 (Fröhliche Weihnachten !!d.RED) Bisher habe
ich nichts weiter gemacht, weil ich noch wissen wollte, was sich durch den 83-Standard ändert und ob ein paar sinnvolle "features" neu dazugekommen sind. Demnächst geht es weiter. Wenn mein FORTH (vielleicht -83) lauffähig sein wird, sind meine Pläne: Supertape (siehe auch cf), Betriebssystem für "Krüppelkomputer" (in FORTH?) mit Support, des weiteren FORTH - Support für Z80 -"Krüppelkomputer", Hilfestellung bei anderen CPUs für FORTH-"Krüppelkomputer" (lokale Anlaufstelle MAINZ). Dies alles braucht seine Zeit, da es auch Leute geben muß, deren Haupthobby nicht der Computer ist.

Der Hrsg. meint: Mutig, mutig. Viel Ausdauer und wenig Klagen. Das lob ich mir. Das Angebot als FORTH-Hilfesteller in Mainz zu gelten, nehmen wir dankend an. Allerdings glaube ich, daß du nur deshalb zwei Jahre gebraucht hast, weil Du so oft von der Kassette laden musstest. Für seine Absturzsicherheit ist FORTH ja nicht gerade berühmt geworden. Was die Implementation mittels eines minimalen Kernels anbelangt: Dazu werden wir bald in der VIERTE DIMENION etwas schreiben. Bis dann: MAY FORTH BE WITH YOU !! H.G.L.

KONTAKTE:

RAUM MAINZ / FRANKFURT
Horst Kroker, Heinrich-v.-Meißen Str. 37, 6500 Mainz 42
Andreas Soeder, Am Landbach 5, 6104 Seeheim-Jugenheim

RAUM FRANKFURT/KOBLENZ
Detlef Schwarzer, Haupstr. 65, 5431 Dreikirchen

RAUM STUTTGART/KARLSRUHE
Klaus Peter Gadacz, Saphirweg 18, 7143 Vaihingen/Enz

RAUM OSTWESTFALEN / LIPPE
Bernfried Molte, 4920 Lemgo 1, Steinstoß 24

RAUM BERLIN
Gerd Blanke, Ackerstr. 71-76, 1000 Berlin

** KURSE **

SCHWEIZ

DR. W. Wejgaard, Stickereiweg, CH - 5615 Fahrwangen
Abendkurse ab Herbst 84, Praktikum an der ETH im Frühjahr

Hamburg Zentrum:

Common Interface Alpha, Schanzenstr. 27, 2000 Hamburg 6
Abendkurse f. Anfänger, evtl. Wochenendkurse, alles in Vorbereitung, Fragen kostet nichts.

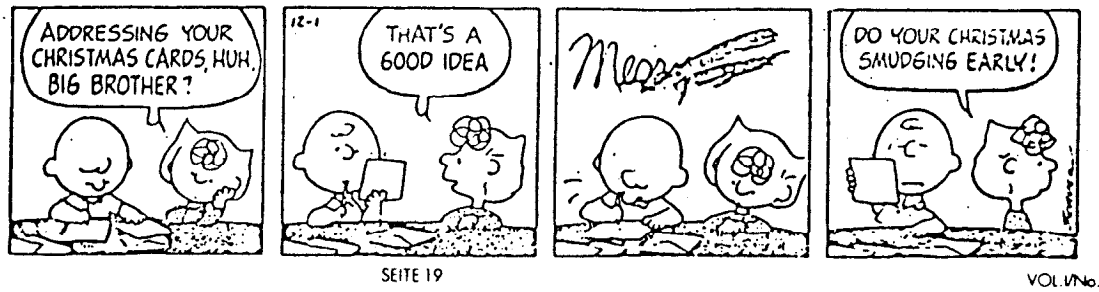
Hamburg - Norderstedt

ab Mitte 85:
Eckart W.F. Schmidt, Am Birkenhof 53, 2000 Noderstedt

Hamburg Ahrensburg:

Roland Löhr, Hausdorfer Str. 4, 2070 Ahrensburg

SMUDGE



DIE SIEBEN TODSÜNDEN WIDER DAS EFFEKTIVE PROGRAMMIEREN

1. TEIL: DIE ERSTEN BEIDEN SÜNDENFÄLLE

Ich will einmal die Grundsätze effektiven Programmierens, und ihre glänzendsten Verfehlungen, unter diesem polemischen Titel auflisten. Wir hätten hier also erste Hauptsünde wider das effektive Programmieren:

Du Sollst Nicht Den Zweiten Schritt Vor Dem Ersten Tun

Die Verdrehung der Zeitabläufe. Ein Computer ist eine einfache Maschine, die alles, was sie tut, streng hintereinander tut. Der einzige Grund, warum man nicht verstehen sollte, was diese Maschine aufgrund ihrer Anweisungen tut, ist, daß die Struktur dieser Anweisungen derartig verdreht und verworfen wurde, daß man ihrer Sequenz nicht mehr folgen kann. Infixnotation, also die Konvention, erst zu sagen, was getan werden soll, ohne zu wissen, womit es getan werden soll, ist der Hauptgrund dafür. Die Notation selber verdeckt das Denkschema, das dahinter steht, und für uns ist es umso schwerer, das zu sehen, weil wir selber so denken. Was für einzelne Anweisungen gilt, gilt für die gesamte Kontrollstruktur des Programms. FORTH wendet, da es mit dem return-stack arbeitet, dasselbe Prinzip auf allen Ebenen an, und vermeidet es damit optimal, den Computer oder den Programmierer mit irgendwelchen Vorhersagen zu belasten, von Bedingungen, die noch nicht erfüllt sind.

Die Zweite Todsünde wider das effektive Programmieren

Stellen Sie sich vor, lieber Leser, wenn Sie eines Morgens nach sehr schlecht geschlafener Nacht an Ihren Computer gingen, und das Programmlisting anschauten, das Sie gestern noch sehr, sehr spät von dem endlich korrekten Programm gemacht hatten, und wenn Sie da lesen würden:

```
100 A DIJKSTRA B WIRTH C AIKEN

200 BABBAGE I DIJKSTRA  ADA 10

300 K DIJKSTRA I WIRTH I AIKEN
```

— — — —

Sie erinnern sich aber noch dunkel, daß der Anfang Ihres Programms gestern noch gelautet hatte:

```
100 A = B * C ;

200 FOR I = 1 TO 10

300 K = I * I ;
```

— — — —

Sie würden natürlich ersteinmal blitzschnell den Programmierer-Standard-Test auf Realität machen, um festzustellen, ob Sie das nicht alles träumen, und Sie stellen sich aufs rechte Bein, greifen sich mit dem rechten Arm unter das linke Knie durch, und zwicken sich ins rechte Ohr. - Es kneift, also sind Sie wach, und diese Welt ist real.

Nun fällt Ihnen ein, was Sie in Ihrem vorletzten SF-Roman gelesen haben, nämlich daß hier offensichtlich eine leichte Realitätsverschiebung stattgefunden hat, und einige Parameter der Geschichtsentwicklung gegen andere ausgetauscht wurden. Sie schauen sich das Listing noch einmal an, und da geht Ihnen ein Licht auf: Ja, das erkennen Sie doch, die bekannten Zeichen

```
= * FOR TO ;
```

und so weiter sind einfach gegen die Namen von berühmten Forschern aus der Informatik ausgetauscht worden. Und nun ist alles klar. Hier muß eine Parallelwelt vorliegen, in der die mathematische Fakultät etwas stärkeren Einfluß auf die Computerei genommen hat, und in ihrem gewohnten Stil alles, was an neuen Operationen gefunden wurde, nach den Namen ihrer verdienten Forscher benannt hat.
Das kennen Sie ja noch aus der Schule: Peano-Axiom, Eulersche Zahl, Pythagoras-Satz. Für Sie ist das kein Problem, Sie schreiben einfach ein kleines Programm, das diese Namen gegen Ihre gewohnten Standard-Symbole austauscht, und Sie programmieren weiter.

Leider gibt es nur in dieser Science-Fiction ein happy-end. In unserer realen Welt wissen nur ganz wenige Leute, was das Peano-Axiom ist, und deshalb gibt es auch nur so wenige Mathematiker. Sie meinen, da hätte ich eine Logik falsch herum geführt ? Dann versuchen Sie doch einmal, in einer Programmiersprache effektiv zu werden, die die obige Notation verwendet. Ich wette mit Ihnen, Sie können es, und ich könnte es auch, aber wieviele Leute würden es aufgeben, irgendetwas davon zu lernen, weil ihnen das, womit sie umgehen, überhaupt nichts darüber aussagt, was sie da eigentlich tun. Das ist also die Zweite Todsünde wider das effektive Programmieren:

Du Sollst Nicht Falschen Namen Nennen

Es ist im Prinzip die Anwendung des ersten Beispiels von FORTH, als wir einen Namen gegen einen anderen austauschten. Die Bedeutung dieses Gebots und seiner Verletzung ist in normalen Programmiersprachen garnicht zu erfassen, weil die Konventionen der Syntax überhaupt keine Freiheit lassen, etwas anderes zu sagen, als der Compiler verlangt. Da ist eben Addition immer auf "+" festgelegt, Subtraktion auf "-" und so weiter. In FORTH aber besteht die Möglichkeit, alles so zu nennen, daß es für den Menschen optimal verständlich ist. Dem Computer ist es sowieso egal, wie es heißt, denn er arbeitet nur mit Nummern.

Die Soziologie der Programmierung

Wir greifen hier in Bereiche, die mit purer Methodik nichts mehr zu tun haben, wohl aber damit, warum wir das tun, was wir tun, und wieso wir es so tun und nicht anders. Also das Feld der Anthropologie und der Soziologie. Was wir hier ansprechen, kann man auch die **Soziologie eines Computersystems** nennen.
Charles Moore hatte möglicherweise einen Mißgriff getan, als er sein System FORTH nannte, implizierend, daß sein System die vierte Generation von etwas war, wovon COBOL und FORTRAN vielleicht die zweite Generation waren, und PASCAL die dritte. (man verzeihe mir diesen faux pas. Als FORTH erfunden wurde, gab es PASCAL noch nicht. Aber ich konnte es nicht über mich bringen, PASCAL evolutionsmäßig auf dieselbe Stufe wie COBOL zu stellen.) Es kommt nicht darauf an. Worauf es ankommt ist: Von einem System der vierten Generation nimmt man an, daß es die Charakteristiken der Generationen vor ihm irgendwie weiterführen, und verbessern sollte. Dies ist aber bei FORTH absolut nicht der Fall, und dieser implizite Anspruch mag das größte konzeptuelle Hindernis für jeden sein, der irgendwann schon einmal mit Programmierung in FORTRAN, COBOL, oder BASIC zu tun gehabt hatte, überhaupt einzuschätzen, was FORTH ist. FORTH ist keine "Fortsetzung" irgendeines "Konzepts" aus der EDV der IBMs, der UNIVACS, und auch nicht der DECs und DGs. FORTH hat mit dieser Programmierung nur den Namen gemeinsam, und dieser Name hat eine andere Bedeutung. FORTH stellt sozusagen eine separate Entwicklungslinie in der Evolution der technischen Sprachen der Menschheit dar. Daher hat es auch nur eingeschränkt einen Sinn, FORTH in einen Vergleich mit einer Compilersprache wie PL/I, oder sogar einer Interpretersprache wie BASIC zu setzen. FORTH entstand als Ein-Mann-Programmiersystem.

Durchweg alle anderen Programmiersprachen sind das Produkt von vielen, ungeheuer teuren Mann-Jahren an Entwicklungsarbeit. Eine gewaltige Investition, die man so lange, wie es geht, nutzen möchte.
Man betrachte die Situation: Da werden riesige Investitionen in Computersysteme gesteckt, damit diese Computersysteme, Hardware und Software und alles, ungeheure Arbeitseinsparungen in allen Bereichen der Wirtschaft und der Technik ermöglichen. Nur eins ermöglichen sie nicht: Arbeitseinsparungen in ihrer eigenen Programmierung.

Die unvorstellbare Komplexität der EDV-Systeme ist so allverbreitet, daß es undenkbar ist, die Notwendigkeit dieser Situation in Frage zu stellen. Ebenso unvorstellbar ist es für einen Praktiker auf diesen Systemen, eine Möglichkeit, daß es anders sein könnte, auch nur im Bereich seines geistigen Horizonts zu dulden. Und eben genau das repräsentiert FORTH. Und deshalb ist es unmöglich, in der EDV-Szene ein Wort über FORTH zu sagen, und ausgehört zu werden.

Es gibt Gedanken, die sind in der Welt der 10-Millionen-Dollar Mainframes und 4-Stunden Turnaround-zeiten für Batch-Jobs nicht denkbar. Zum Beispiel der Gedanke eines Ein-Mann-Programmiersystems. Wer hätte Zeit, so etwas zu entwickeln, und wer wäre hinterher daran interessiert, es zu gebrauchen? Heutige kommerzielle timesharing-Systeme sind, wie das TSO/MVS, auf die alten batch-OS Konzepte aufgebaut und simulieren im wesentlichen nur Batch Processing mit vielen gleichzeitigen Job-queues.

Es gibt viele Faktoren, die die Erscheinung der Groß-EDV prägen. Ein wesentlicher Faktor ist, daß Groß-EDV in Großkonzernen eingesetzt wird, und da ist kein Platz für Ein-Mann-Unternehmungen. Höchstens für den Mann an der Spitze, aber der soll ja nicht programmieren. Der Hauptfaktor aber ist ökonomischer Art. Dies mag eine Milchmädchenrechnung beispielhaft belegen: Die Grundannahme ist, daß die Programmierstunde auf einem IBM-System etwa 200 DM kostet. Daher muß man Computersprachen verwenden, die den wenigen vorhandenen Spezialisten so wirksame Werkzeuge in die Hand geben, daß sie eine maximale Produktivität in ihrer knappen Zeit erreichen. Die maximale Produktivität haben sie nur, wenn sie das programmieren können, was sie sowieso schon am besten können, nänmlich COBOL oder PL/1. Es werden also immer mehr Verbesserungen in den PL/I-Compiler eingebaut, die noch mächtigere Kommandos erlauben, und noch mehr Spezialfälle abdecken können. Manchmal lassen sich aber einschneidende Neuerungen nicht vermeiden. Das Erscheinen von Pascal hat die Notwendigkeit von strukturierter Programmierung und von Datentypen deutlich gemacht. Daraufhin wurde ADA entwickelt, die neue Standardsprache. Diese Sprache ist so mächtig, und so komplex, daß es heute noch keinen Compiler gibt, der ihren vollen Umfang implementiert hat. Zudem braucht ein Programmierer etwa vier Jahre Training, um in ADA effizient arbeiten zu können. Nach diesen vier Jahren sind nur etwa 10 % der Leute, die angefangen hatten, es zu lernen, übriggeblieben, und um diese 10%, nun äußerst effiziente ADA-Programmierer, reißt sich der Markt, so daß diese Spezialisten in der glücklichen Lage sind, für ihre Dienste 250 DM pro Stunde verlangen zu können. Diesen Mechanismus könnte man auch den **Goldenden Schnitt** der EDV nennen.

Andreas Goppold

AUS DEM WÖRTERBUCH DER EDV

batch:
(Stapelbetrieb) Verfahren aus dem vor-personal-Zeitalter der Computerei, bei dem Programme dem Computer in Form von großen Lochkartenstapeln eingegeben wurden. Da ein Programm in seiner Herstellung zig Durchläufe durch den Computer erforderte, die jedesmal zwischen einer halben Stunde und einen halben Tag erforderten, bürgerte sich die Methodik der Programmierung ein, daß ein Programmierer jeweils mehrere Programme gleichzeitig bearbeitete. Dieses Verfahren nannte man timesharing. Später kam man darauf, daß man auch, anstatt den Programmierer viele Programme gleichzeitig bearbeiten zu lassen, den Computer das machen lassen konnte. Von da an hatte timesharing seine weiter unten definierte Bedeutung.
job:
biblische Person, die von der Hand ihres Meta-Programmierers unsägliche Qualen erleiden mußte.

queue:
Menschenschlange. Haupt-Gesellschaftsspiel der angelsächsischen Kulturen. Es ist ein besonderes Ziel, den Rekord für die maximale Verweildauer in einer queue zu erringen. Mit der Einführung der Computer wurden viele Spiele auf den Computer übertragen. Dies war eines der ersten.

timesharing:
viele Benutzer teilen sich die Restmaschinenzeit, gemessen in Microsekunden, die das Betriebssystem, das ihnen diese Computerzeit zuteilen soll, noch davon übrigläßt.

turnaround:
Mittlere Verweildauer, gemessen in Stunden, die ein batch-job (siehe dort) in verschiedenen Stadien der Maschinenverarbeitung zubringt.
Anmerkung: verweilt ein Job länger als 3 Tage in der Maschine, wird er abgebrochen.

AG? - AG!

Bericht aus der Leibniz AG

Die Arbeitsgruppe hat sich einigemale getroffen und eine Vorschlagsliste deutscher Worte für LEIBNIZ auf der Basis von FORTH-83 erarbeitet.
Im Einzelnen hat die AG bisher folgende Beschlüsse gefaßt:

Umlaute können in den Namen nicht benutzt werden.

Groß/kleinschreibung soll vom Compiler nicht unterschieden werden, so, wie es z.B. CP/M macht. Dadurch hat jeder die Freiheit, Groß/kleinschreibung nach eigenem Gusto zu benutzen.

Offen ist noch die Benutzung von Konjugation und Deklination. Dies sind Ausdrucksmöglichkeiten, die im Englischen nicht existieren aber vermutlich vorteilhaft eingesetzt werden können, um die Lesbarkeit von Programmen zu erhöhen.

Eine weitere Möglichkeit, die ernsthaft untersucht werden muß, ist die Verwendung von Sonderzeichenkombinationen als 'Ikonen'. Siehe dazu auch den Beitrag von A. Goppold in diesem Heft.

Zur Zeit befindet sich die Arbeitsgruppe in einer Phase der Stagnation. Hieraus wird sie hoffentlich massives Feedback befreien, das dadurch zustande kommt, daß den Mitgliedern der Forth Gesellschaft eine ausführliche Dokumentation der bisherigen Arbeit zugeht. Nichtmitglieder können diese Dokumentation gegen Voreinsendung von DM 5,- in Briefmarken bei der Forth Gesellschaft beziehen.

Für die AG LEIBNIZ

K. Schleisiek

Buchbesprechung

Das war eigentlich eine kleine Lüge. Aber Mangels Belegexemplaren oder von euch eingesandten Buchbesprechungen (zum Thema FORTH), werde ich euch heute ein recht interessantes Magazin vorstellen. Es handelt sich um das MICRO-MAG von Roland Löhr. Bis vor kurzem nannte es sich noch 65xx MICRO-MAG, ein Magazin also welches sich vorrangig mit allen 65xx-Prozessoren befasste. Dieses Konzept wurde nun erweitert auf alle Prozessoren die mit 6 beginnen. Jochen Dahmke von der AG-Doku sagte einmal, daß das MICRO-MAG wohl das deutsche Magazin mit den meisten FORTH-Artikeln ist und damit hat er wohl recht (Er hat diese schließlich alle analysiert). Der Schwerpunkt des Magazins liegt dabei auf der Besprechung / Erläuterung neuester CPU-Typen(z.b. 68000, 65816, Einchipper etc.) und deren Programmierung in Assembler. In der August - Ausgabe zum Beispiel wird ein Cross-assembler für den MC6809 vorgestellt, der natürlich in FORTH geschrieben ist. Aus dem Hauptinteresse des Herausgebers, der sich seit einigen Jahrzehnten mit Sprachen beschäftigt, ergibt sich, daß besonders viel Mühe auf die Beschreibung der Handhabung aller möglichen Assembler sowie Hochsprachen verwendet wird. In jedem Falle ist dieses Magazin für jeden bundesdeutschen Entwickler oder solche die es werden wolleneine unerläßliche Informationsquelle. MICRO MAG erscheint zweimonatlich und kann für 54,- DM / Jahr abonniert werden. Herausgeber : Dipl.Volkswirt Roland Löhr, Hansdorfer Strasse 4, 2070 Ahrensburg

Das Allerletze

WITZ DES MONATS

Oder: Wie man aus einer Not eine Tugend macht.

Die Not: Nachfolgender " Text ", der nur dadurch entstanden ist, daß wir alles selber machen wollten. Insbesondere natürlich die Fehler. Ein solcher liegt hier vor, weil wir die falsche Schrift angewählt hatten.

Die Tugend: Dieser Witz ist gar kein Witz, sondern ein

*** PREISAUSCHREIBEN ***

GEWINNER dieses Preisausschreibens wird man, indem man herausfindet, was dort eigentlich geschrieben steht (In ASCII-Standard) Um die Sache etwas zu erleichtern: Derselbe Text ist an anderer Stelle, in diesem Heft in " lesbarerer " Form zu finden.

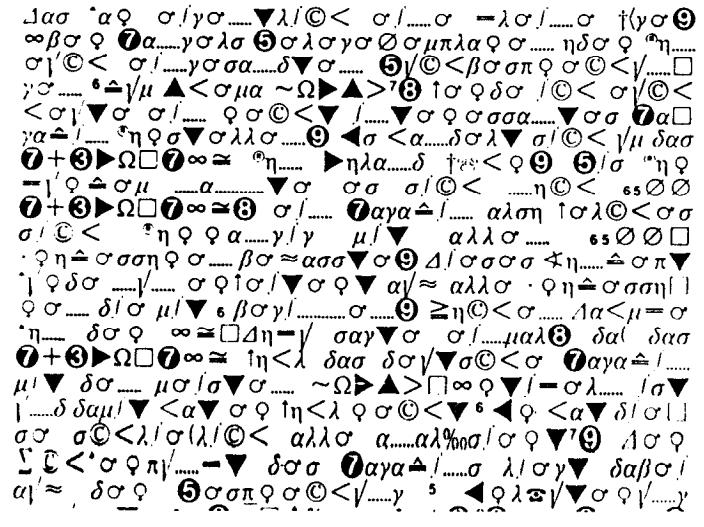
DIE PREISE :

1. Preis: 3 Tage kostenloses Mitarbeiten im C.I.Alpha
2. Preis: 2 Tage kostenloses Mitarbeiten im C.I.Alpha
3. Preis: 1 Tag kostenloses Mitarbeiten im C.I.Alpha

Die Namen der Gewinner werden natürlich in der nächsten Ausgabe veröffentlicht.

Vielleicht doch ein Witz ?

❺¯❸>❺◀Σ·▶◀❸>¯❻≅|



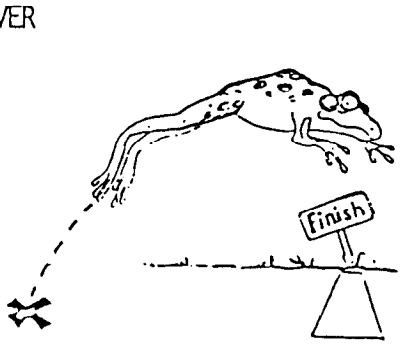
KLEINANZEIGEN

Diese allgemein verbreitete Rubrik, darf natürlich auch in diesem Blatt nicht fehlen. Die folgenden Kleinanzeigen haben wir umsonst veröffentlicht. Als Startangebot, sozusagen. Ab der nächsten Ausgabe kostet eine Kleinanzeige DM 5,-, maximal 30 Worte sind zulässig. Wir behalten uns das Recht vor eine Anzeige zurückzuweisen. Natürlich erhaltet ihr evtl. eingesendetes Geld zurück. Kommerzielle Inserenten mögen bitte eine Anzeige buchen. Preise sind im C.I.A zu erfahren. Einsendeschluß: 23.12

Biete: Funktionierendes FIG 1.1 Forth für CBM 8032 mit Floppy, auf 5 1/4" Diskette für 30,- DM Mit Editor,Assembler, Loader und weiteren Utilities im Source bei: Wolfgang Mües, Hans-Sommer-Str. 25/512, 3300 Braunschweig

Suche: Wer hat Erfahrungen mit dem MVP-Forth-Expertensystem ? Kontakte / Adressen für Jochen Dahmke an das Common Interface Alpha schicken. Vielen Dank !

OVER



Fragebogen 1/84

Mit diesem Fragebogen wollen wir ein Forum einrichten , um die unterschiedlichen (??), erhältlichen FORTH - Versionen anhand praktischer Erfahrungen beurteilen zu können. Solange der '83 er Standard noch nicht fertig ist, hoffen wir Euch mit der Auswertung der Antworten wenigsten vor allzu schlechten/teuren oder sonstwie ungeigneten Implementationen bewahren zu können. Versucht also bitte objektiv zu sein, damit wir hier nichts falsches in Umlauf setzten.

Meine Erfahrungen mit FORTH-IMPLEMENTATIONEN würde ich wie folgt bewerten:

Anbieter	Version	Jahr	Preis
1._____	_____	_____	_____

Prozessor:_____

Doku: gut/mittlm./ungenügend/keine/unleserlich

zusätzlich über FIG-FORTH hinaus:

O Floating Point
O Graphik
O String Stack
O OS-Interface
O Trace
O Fullscreen Editor
O Dateiverwaltung
O Utilities/welche_____

_____

Anbieter	Version	Jahr	Preis
_____	_____	_____	_____

Prozessor:_____

Doku: gut/mittelm./ungenügend/keine/unleserlich

zusätzlich über FIG-FORTH hinaus:

O Floating Point
O Graphik
O String Stack
O OS-Interface
O Trace
O Fullscreen Editor
O Dateiverwaltung
O Utilities/welche_____

_____

Name :
Vorname:
Strasse:
PLZ/Ort:
Tel. :

Sonstiges:_____

_____

_____