FORTH – eine moderne Softwarephilosophie

Thomas Höhenleitner, Berlin

1. Einführung

Mit dieser Veröffentlichung soll ein Eindruck von den Möglichkeiten des Softwarewerkzeuges FORTH vermittelt werden. Die Hinwendung zu neuen leistungsfähigen Programmierumgebungen fällt dem Assemblerprogrammierer nicht immer leicht. Einerseits ist die Vielfalt der angebotenen Werkzeuge recht groß, andererseits ist die Einarbeitungszeit nicht unerheblich und in Anbetracht zwingender Terminstellungen schwer einzuplanen. Dazu kommt eine gewisse Unsicherheit bei der Wahl des weiteren Weges, denn seriöse vergleichende Untersuchungen zu den einzelnen Programmiersprachen sind aufwendig und fehlen im allgemeinen. Wenn beim Leser Interesse und Bereitschaft geweckt werden, FORTH in den Variantenvergleich der Mittel für die Lösung eigener Aufgaben einzubeziehen, ist das Ziel dieser Arbeit erreicht. Die hohe Portabilität von FORTH-Systemen, ihre Modularität und die damit verbundene Effizienz bei der Programmierung, die Kompaktheit und die Leistungsfähigkeit dieser Sprache führen zu einer neuen Qualität in der Softwareentwicklung. Auch der Hardwareentwickler wird sich dieses Werkzeuges zur Bearbeitung seiner Probleme (Testroutinen, Hilfsprogramme, Schnittstellentreiber) bedienen wollen.
FORTH wurde Ender der 60er Jahre in den USA erfunden. Der Schöpfer Charles Moore benötigte eine effizient zu programmierende und schnell laufende Sprache für seine Steuerungsaufgaben. Das Konzept war so überzeugend, daß es seither durch eine Interessengemeinschaft, die Forth Interest Group, gefördert, verbreitet und weiterentwickelt wird. Der erste Standardisierungsversuch wurde figFORTH genannt und hat inzwischen viele Anwendungen gefunden. 1979 folgte FORTH79, erfreute sich jedoch [illegible] so breiter Akzeptanz. Gewachsen aus den Erfahrungen bei der Arbeit mit FORTH, wurde 1983 der Standard FORTH83 definiert, der sich zunehmend verbreitet, da er eine ganze Reihe von Verbesserungen gegenüber figFORTH und auch FORTH79 enthält. Bei den angeführten Beispielen wird bei Bedarf auf die Unterschiede zwischen figFORTH und FORTH83 hingewiesen.

1.1. Die FORTH-Maschine

Die Abarbeitung der FORTH-Instruktionen übernimmt eine Maschine, deren Befehlssatz aus den FORTH-Worten besteht. Dabei ist es für die Funktion ohne Belang, ob die Maschinenregister und ihr Zusammenspiel (innerer Interpreter) direkt in Hardware oder softwaregestützt durch einen Mikroprozessor realisiert sind. Der FORTH-Prozessor ist im wesentlichen eine Stackmaschine, die auf die Abarbeitung des FORTH-Codes abgestimmt ist. Die wichtigsten Register des FORTH-Prozessors seien kurz vorgestellt:

***IP** Instruction Pointer*
zeigt in die Wortadreßkette (s. 3.5.1.) des gerade in Abarbeitung befindlichen Wortes.

***W** Wortadreßregister*
hält die gerade aktuelle Wortadresse, das ist der Inhalt der von **IP** indizierten Zelle.

***PC** Programm-Counter*
Befehlszähler

***RP** Returnstack Pointer*
zeigt auf TOR (**T**op **o**f **R**eturnstack).
Hier befinden sich Rückkehradressen bzw. Schleifenparameter der rufenden Ebenen.

***DP** Datenstack Pointer*
zeigt auf TOS (**T**op **o**f **S**tack).
Das ist der Puffer der gerade zu verarbeitenden Daten. Der Parameterstack übernimmt gleichzeitig die Arbeitsregisterfunktion.

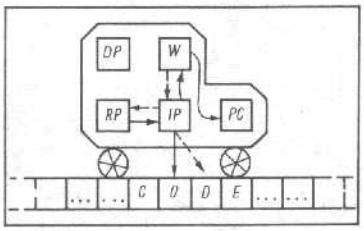

Bild 1 Eine FORTH-Maschine

Die funktionalen Zusammenhänge der Register der FORTH-Maschine (siehe hierzu Bild 1) werden im Abschnitt 3.5.5. skizziert. Ein genaues Verständnis an dieser Stelle ermöglicht dem FORTH-Anwender den Einsatz spezieller Programmiertechniken, mit denen sich u. a. auch besonders zeitkritische Probleme beherrschen lassen.

1.2. FORTH in Hardware

Der Trend zu immer größeren Verarbeitungsbreiten hat technologische und wohl auch programmtechnische Grenzen. Jüngste Entwicklungen auf dem Gebiet der Prozessortechnik zeigen, daß mit intelligenteren Architekturen Rechengeschwindigkeiten erreicht werden, die weit über den Werten üblicher Prozessoren liegen. RISC-Prozessoren (**R**educed **I**struction **S**et **C**omputing) seien kurz genannt. Eine in /2/ und /3/ beschriebene Hardwarerealisierung eines FORTH-Prozessors in CMOS-Gate-Array-Technik zeigt eine weitere interessante Entwicklungsrichtung. Die 25fache Abarbeitungsgeschwindigkeit (!) dieser FORTH-Maschine gegenüber einem C-Programm auf einem 16-Bit-Prozessor (INTEL 8088), wie sie am Beispiel des Siebs des Erasthotenes (Primzahlermittlung) festgestellt wurde (/2/), gibt eine Vorstellung von der Leistungsfähigkeit einer solcherart strukturierten Hardware, deren Hersteller für die mit 4,5 MHz getaktete CPU 6 MIPS (**M**illionen **I**nstruktionen **p**er **S**ekunde) angibt. Weitere Ausführungen zu Hardware-Realisierungen von FORTH findet man u. a. in /4/.

1.3. FORTH auf einem Hostrechner

FORTH-Systeme wurden für alle gängigen Prozessoren und Rechner implementiert. Der Assemblercode umfaßt i. a. nicht mehr als 2 KByte und ist allgemein zugänglich (z. B./5/, /13/). Etwa 80 % des FORTH-Kernes bestehen aus FORTH-Secondaries, d. h. Worten, die ihrerseits aus FORTH-Worten definiert wurden (vgl. 2.1.1.). Die Anpaßarbeiten an einen beliebigen Rechner, dessen Prozessor dem einer vorhandenen FORTH-Version entspricht, beschränken sich auf die Systemschnittstellen bei einigen wenigen FORTH-Worten. Damit wird klar, wie die hohe Portabilität von FORTH zustande kommt. Oft gibt es für einen Rechner verschiedene Implementierungen, welche sich in ihrem Auf- und Ausbau erheblich unterscheiden können.
Für die Verwendung der prozessorinternen Register gibt es Empfehlungen, die jedoch nicht unbedingt eingehalten werden müssen. Wie gut sich verschiedene Mikroprozessoren als Stackmaschine einsetzen lassen, hängt von der speziellen Registersteuerung ab /1/, /6/, /7/. Hier findet man auch den Schlüssel für Programmlaufzeituntersuchungen, wenn derartige Betrachtungen bei zeitkritischen Applikationen erforderlich werden sollten.

2. Die Programmiersprache FORTH

2.1. Offenes System

FORTH ist erweiterbar – nicht nur im Sinne von zusätzlichen Funktionen schlechthin. Programmieren in FORTH heißt FORTH erweitern, bis letztendlich der Programmname Bestandteil des FORTH-Vokabulars geworden ist. Der unterschiedliche Begriffsinhalt von Worten wie Operand, Befehl, Routine, Konstante, Variable, Feld, Programm, Vokabular verschmilzt zu einer eine neue Denkweise provozierenden Einheitlichkeit. Man kann sich FORTH als ein Baukastensystem vorstellen, aus dessen Elementen beliebige neue Elemente erzeugt werden. Alle Elemente, vorgegebene wie neu erzeugte, sind ständig verfügbar; sowohl für den interaktiven Dialog als auch für die Generierung neuer Elemente. Das erzeugte Programm stellt letztendlich auch nur ein weiteres Element neben beliebigen anderen dar. In dieser Modularität offenbart sich ein entscheidender Vorteil für die Programmentwicklung. Probleme werden in ihre Bestandteile zerlegt und damit besser verstanden. Jedes Element kann für sich analysiert und bearbeitet werden, ist einzeln testbar, wartbar und jederzeit wiederverwendbar. Unterprogrammaufrufe etwa in Form von „Call name" sind unbekannt. Die FORTH-Worte werden wie in einer natürlichen Sprache aneinandergefügt.

2.1.2. Definitionswörter

FORTH bietet verschiedene Definitionswörter, mit deren Hilfe das System jederzeit um neue Befehle bereichert werden kann. Die gebräuchlichste Erweiterungsform ist die Colon-Definition, deren Anwendung bei der Definition eines Hilfswortes .S zur Anzeige der oberen 3 Stackzellen demonstriert werden soll:

: .S 3 0 DO ROT DUP . LOOP ;⟨RET⟩**OK**

Eingaben werden generell jeweils durch mindestens ein Leerzeichen getrennt. ⟨RET⟩ bedeutet RETURN-Taste drücken. Terminalausgaben sind unterstrichen.

:	eröffnet die Definition
.S	ist ein frei wählbarer Name (dot Stack)

3 0 DO eröffnet DO ... LOOP Schleife für 3 Durchläufe
ROT ROTiert die oberen 3 Stackelemente
DUP DUPliziert das obere Stackelement
. dot transportiert oberes Stackelement zum Terminal
LOOP Ende der Schleife
; Ende der Definition

.S kann sofort getestet werden ...

7 2049 12⟨RET⟩ **OK** (3 Werte auf Stack)
.S⟨RET⟩**7 2049 12 OK**
138 .S⟨RET⟩ **2049 12 138 OK**

... und in alle weiteren Definitionen eingebaut werden. Wird die Stackanzeige in mehreren Zahlenbasen gewünscht, kann man weiter definieren (Einbau des eben kreierten Wortes .S):

```
: DS BASE @          (aktuelle Zahlenbasis)
   >R                          (Retten)
   CR DECIMAL    ."dezimal      ".S
   CR HEX        ."hexadezimal  ".S
   CR 2 BASE !   ."binaer       ".S
   R>            (Restaurieren der)
   BASE ! ;      (Zahlenbasis und
                  Ende)
```

Test des neuen Wortes:

DS⟨RET⟩
dezimal **2049 12 138**
hexadezimal **801 C 8A**
binaer **100000000001 1100**
10001010 OK

Es gibt komfortablere Stackdumps, darum geht es hier jedoch nicht. Diesen mit einer Doppelpunktdefinition generierten Worttyp bezeichnet man als Secondary.
Ein weiteres Definitionswort ist z. B. **VARIABLE**. Damit kann man beliebige RAM-Zellen symbolisch unter einem Namen ansprechen.

VARIABLE MINUTEN⟨RET⟩ **OK**[1]

weist dem neuen FORTH-Wort MINUTEN eine bestimmte Adresse im RAM zu. Der Aufruf 'MINUTEN' stellt diese Adresse auf dem Stack bereit und ermöglicht zusammen mit Worten wie

!	RAM-Schreiben	(wert addr→)
+!	RAM-Addieren	(wert addr→)
@	RAM-Lesen	(addr→ wert)
?	RAM-Ausgeben	(addr→)

eine bequeme Handhabung dieser Variablen:

4 MINUTEN !⟨RET⟩ **OK** (setzt die Zeit)
3 MINUTEN +!⟨RET⟩ **OK** (erhöht die Zeit)
MINUTEN ?⟨RET⟩ **7 OK** (zeigt die Zeit)

(Das Wort **?** findet man nicht unbedingt in allen FORTH-Implementierungen, da es nur im figFORTH standardisiert wurde. Man kann es im Bedarfsfall mit **: ? @ . ;** selbst definieren.)
FORTH bietet darüber hinaus die Möglichkeit, eigene Definitionsworte zu erzeugen. Die Verarbeitung sehr spezieller Datentypen ist damit problemlos möglich. Zum Beispiel lassen sich mit der **CREATE ... DOES**⟩-Konstruktion auf einfache Weise Definitionsworte generieren, mit denen man den erzeugten Worten bestimmte Compile- und Laufzeiteigenschaften zuordnen kann. Wie das in etwa funktioniert, soll ein Beispiel verdeutlichen:

[1] Gilt für FORTH83. **VARIABLE** in figFORTH-Versionen erwartet einen Initialisierungswert auf dem Stack, also:
0 VARIABLE MINUTEN⟨RET⟩ **OK**

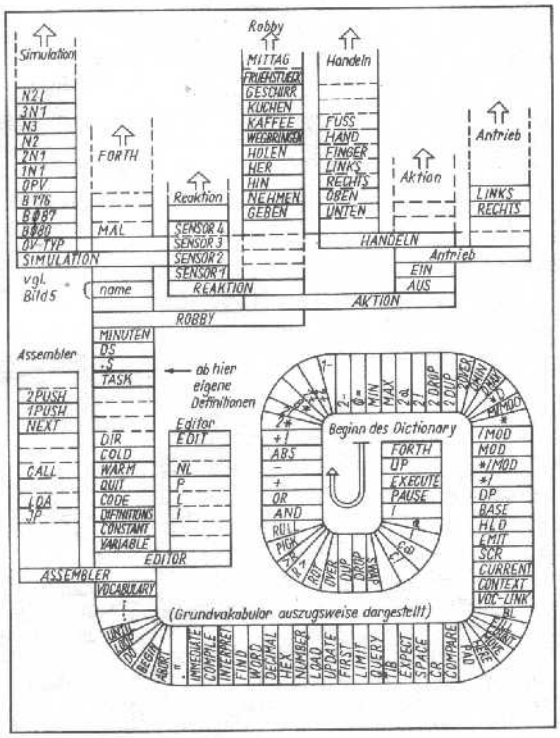

Bild 2 Baumstruktur eines FORTH-Dictionary

Bei der Erstellung eines Simulationsprogrammes für elektronische Schaltungen sind u. a. verschiedene Operationsverstärker auf bequeme Art und Weise einzubinden. Der Aufruf der Typbezeichnung soll die Adresse der speziellen Operationsverstärkereigenschaften, z. B. Offsetspannung U_{Off} und Leerlaufverstärkung v_{leer}, im vereinbarten Zahlenformat auf dem Stack hinterlegen. Für die bequeme Definition vieler solcher Typen wird das Definitionswort OV-TYP generiert (wie, führt hier zu weit), dessen Anwendung etwa in der Form

0.015 15000. OV-TYP B080
0.0005 25000. OV-TYP B087
0.006 25000. OV-TYP B176

erfolgt. Das Compilezeitverhalten dieses neuen Definitionswortes besteht in der Compilierung der jeweils auf dem Stack bereitgestellten OV-Eigenschaften in das Parameterfeld (s. 3.5.1.) der damit definierten OV-Typbezeichnungen. Sein Laufzeitverhalten bewirkt, daß der Aufruf einer solchen OV-Typbezeichnung die Adresse seines eigenen Parameterfeldes auf dem Stack hinterläßt, also einen Pointer auf die jeweiligen OV-Eigenschaften. Weiter läßt sich ein Definitionswort OPV für die einzelnen OV-Bauelementebezeichnungen der Schaltung mit den folgenden Eigenschaften deklarieren.

Compile-Zeitverhalten:

(Aktivität bei Aufruf des Definitionswortes OPV).
Auf dem Stack wird ein Pointer auf die speziellen Datenblattkennwerte des betreffenden Bauelementetyps erwartet (bereitgestellt durch den Aufruf einer definierten Typbezeichnung) und in das Parameterfeld der zu erzeugenden Worte compiliert. Die einzelnen Operationsverstärker der Schaltung werden also mit den „individuellen Eigenschaften" eines OV-TYPs versehen. Die Anwendung bei der Definition der Bauelementebezeichnungen sieht dann so aus:

B080 OPV 1N1 **B080 OPV 2N1**
B080 OPV 3N1
B087 OPV N2 **B176 OPV N3**

Laufzeitverhalten:

(Aktivität bei Aufruf eines mittels OPV definierten Wortes).
Der Aufruf der einzelnen definierten Bauelemente könnte dann folgendes bewirken: Alle Spannungspunkte der zu simulierenden Schaltung sind z. B. als Variablen vereinbart. Mit dem Aufruf von beispielsweise N2 werden die die zugehörige Eingangsspannung haltenden Variablen N2U+ und N2U− entsprechend der im Definitionswort OPV vereinbarten Simulationsfunktion verknüpft, und das Ergebnis wird in die zugehörige Ausgangsspannungsvariable N2UA gespeichert. Dieses Laufzeitverhalten ist für alle OV-Bauelementebezeichnungen entsprechend gleich. Der Code existiert nur einmal (im Parameterfeld von OPV).
Das Wesen dieser Programmiertechnik besteht in einer Erweiterung des FORTH-Compilers (s. 3.5.4.) mit der genannten **CREATE ... DOES**⟩-Konstruktion. Die Vorteile liegen in der Vermeidung von redundantem Code und der besseren Programmlesbarkeit. Die Zusammenhänge sind vielleicht nicht sofort erfaßbar. Es geht hierbei auch nur um die An-

deutung der Vielfältigkeit der Möglichkeiten, die sich dem Programmierer bei der Softwaregestaltung mit FORTH bieten.

2.1.2. Vokabulare

Die gesamte Wortmenge eines FORTH-Systems, das Dictionary, ist in einzelne Vokabulare unterteilt und baumähnlich strukturiert (Bild 2). Ein Vokabular kann man sich als einen Zweig vorstellen. Jederzeit kann mit **VOCABULARY vocname IMMEDIATE** (s. 3.5.4.) ein weiteres, vorerst leeres Vokabular an der gerade in Erweiterung befindlichen Stelle vereinbart werden. Die Verwendung unterschiedlicher Vokabulare ermöglicht eine übersichtliche Befehlsstruktur sowie kollisionsfreie mehrfache Vergabe des gleichen Namens. Das wird z. B. schon dann interessant, wenn bei unterschiedlichen Zahlenformaten die gleichen Symbole für die entsprechenden Operatoren Verwendung finden sollen. An welcher Stelle mit der nächsten Definition zu erweitern ist, entnimmt das System der Variablen **CURRENT**, die mit der Anweisung **vocname DEFINITIONS** auf das gewünschte Vokabular gesetzt wird. Wörtersuchläufe beginnen immer an der mit der Variablen **CONTEXT** spezifizierten Stelle. Diese Variable zeigt auf das zuletzt aufgerufene Vokabular.
Ein Beispiel soll die Arbeitsweise mit Vokabularen zeigen:
Einem Roboter soll der Begriff HOLEN beigebracht werden. Die Vokabulare AKTION, ANTRIEB, REAKTION gelten als vereinbart. Die Konstanten RECHTS und LINKS stellen die I/O-Adressen zur Ansteuerung zweier Motoren. EIN sendet an die auf dem Stack vorgefundene I/O-Adresse eine 1 und AUS erledigt analog das Gegenteil (hier ist u. U. bereits Maschinencode). Die Worte SENSOR1 und SENSOR2 erzeugen ein Flag auf dem Stack, welches von **UNTIL** verwertet wird. War es 0, so noch einmal **BEGIN**nen, bis die erwartete 1 kommt und die Schleife verlassen wird. Mit zwei einfachen Doppelpunktdefinitionen lassen wir nun unseren hypothetischen Roboter HIN und HER fahren.

```
: AKTION ANTRIEB RECHTS EIN
    BEGIN REAKTION SENSOR1 UNTIL
    AKTION ANTRIEB RECHTS AUS ;
: HER AKTION ANTRIEB LINKS EIN
    BEGIN REAKTION SENSOR2 UNTIL
    AKTION ANTRIEB LINKS AUS ;
```

Analog hat „Robby" noch NEHMEN (d. h., er nimmt ein Teil, dessen Kennzahl er auf dem Stack erwartet) und GEBEN (er läßt alles fallen) gelernt. Mit der folgenden Zeile kann er dann alles HOLEN, was ihm irgend erreichbar ist und natürlich wieder WEGBRINGEN:

```
: HOLEN       HIN NEHMEN HER GEBEN ;
: WEGBRINGEN  HER NEHMEN HIN
              GEBEN ;
```

Mit dem Definitionswort **CONSTANT** deklariert man Kennzahlen, unter denen man z. B. Schubfachnummern vereinbaren kann:

```
7 CONSTANT KAFFEE
12 CONSTANT KUCHEN
3 CONSTANT GESCHIRR
```

Noch eine kleine Definition, und es kann FRUEHSTUECK geben:

Bild 3 Robby (Zeichnung: Jörg Olberg)

```
: FRUEHSTUECK  KAFFEE HOLEN
KUCHEN HOLEN ." Bitte zu Tisch!" ;
FRUEHSTUECK ⟨RET⟩ (Kaffee und Kuchen kommen) Bitte zu Tisch! OK
GESCHIRR WEGBRINGEN⟨RET⟩ (klirr) OK
```

Vergleiche hierzu die Bilder 2 und 3. Natürlich ist in so einem einfachen Fall der Einsatz so vieler Vokabulare nicht gerechtfertigt. Man denke jedoch auch an sehr umfangreiche Applikationen, wo die Möglichkeit, Namen mehrfach vergeben zu können, unschätzbare Vorteile bringen kann.

2.1.3. Assemblerprogrammierung

FORTH-Systeme enthalten i. a. einen Assembler, der die direkte Verwendung der prozessorspezifischen Mnemonik bei der Definition von Primitives ermöglicht. Primitives sind FORTH-Worte, die keine weiteren FORTH-Worte mehr aufrufen, sondern direkt Maschinencode zur Ausführung bringen. Der Assembler ist in einem Vokabular mit dem Namen **ASSEMBLER** untergebracht. Er enthält zusätzlich Strukturierungselemente wie die **IF... ELSE... THEN**[2] oder **BEGIN... WHILE... UNTIL**-Konstruktionen. Die Definition eines Primitives erfolgt mit der Anweisung **CODE name... (Mnemonics)... END-CODE**. Assemblercode kann auch zusammen mit FORTH-Worten innerhalb einer Definition generiert werden. Man formuliert:
: name... (FORTH-Worte)... ;CODE... (Mnemonics)... END-CODE. Es lassen sich auch umfangreiche bereits vorhandene Assemblerprogramme unter einem beliebigen Namen in das System einbinden, wenn man diese in den Speicher lädt und mit einem CALL die Kontrolle an das jeweilige Programm übergibt. Die Rückkehr aus der Primitive-Ebene in das FORTH-System erfolgt mit dem Sprung nach **NEXT**, womit die Kontrolle wieder an den inneren Interpreter (3.5.5.) übergeben wird.

2.2. Datenhandling

2.2.1. Parameterübergabe

Bei Softwareprojekten ist u. a. die Frage der Parameterübergabe zwischen den einzelnen Modulen zu klären. Jede Routine muß wissen, wo die Quelldaten liegen und auf welche Art und Weise die Ergebnisparameter übergeben werden. Hierfür können Variablen oder Register vereinbart werden. Im Multitaskbetrieb sowie bei Interruptanforderung ist Zwischenspeicherung erforderlich. Daten und Adressen liegen völlig durcheinander auf einem Stack und ergeben recht unübersichtliche Systemzustände, was u. a. die Fehlersuche nicht gerade erleichtert. FORTH begegnet diesem Problem wirkungsvoll durch die Verwendung von 2 Stacks. Die getrennte Ablage von Rückkehradressen auf einem Returnstack und Daten auf einem Parameterstack (auch Datenstack oder einfach nur Stack genannt) machen das System überschaubarer. Die generelle Vereinbarung des Datenstacks als Übergabemedium ermöglicht einfachen, unsichtbaren Parametertransfer und entlastet den Programmierer von der Deklaration der Art und Weise der Datenübergabe zwischen den einzelnen Modulen. Das FORTH-Grundvokabular ist so angelegt, daß die jeweils bereitgestellten Parameter von den aufgerufenen Worten vernichtet und Ergebnisparameter für nachfolgende Worte in verarbeitungsgerechter Reihenfolge bereitgestellt werden. So nimmt z. B. das Wort **C@** (Byte-Lesebefehl) eine Adresse von der obersten Stackzeile und hinterläßt stattdessen dort im Low-Byte den gelesenen Wert (High-Byte = 0). Der Returnstack ist für die internen Rücksprungadressen und Schleifenparameter reserviert und braucht i. allg. nicht beachtet zu werden. Dieser Stack kann, unter Berücksichtigung der Strukturierungsanweisungen (**IF THEN, DO LOOP,...**), innerhalb eines Wortes als Zwischenspeicher Verwendung finden, aber auch zum Überspringen von Verarbeitungsebenen gezielt manipuliert werden. Man verwendet dazu die Worte
⟩R (Top of Stack → **R**eturnstack)
R⟩ (**R**eturnstack → Top of Stack)

2.2.2. Massenspeicher

FORTH-Dialogsysteme sind i. a. floppyorientiert. Die Massenspeicheranbindung erfolgt nach dem Prinzip des virtuellen Speichers und wird in den verschiedenen FORTH-Systemen unterschiedlich gehandhabt. figFORTH und ähnliche Systeme kennen i. allg. kein Dateihandling und teilen die Disketten in gleichgroße Blöcke entsprechend dem vereinbarten Diskettenformat auf. Der Zugriff auf die Daten ohne Verwendung einer Directory direkt über Spur- und Sektornummer ermöglicht einfache Disktreiberroutinen und extrem kurze Zugriffszeiten. Andere Implementierungen sind an ein Betriebssystem gebunden und arbeiten mit Dateien (z. B. /12/). Speicherbereiche lassen sich so einfacher zwischen unterschiedlichen Systemen transferieren, die Übersicht über den Diskinhalt ist besser, und es brauchen keine besonderen Disketten eingerichtet zu werden. Innerhalb der gerade vereinbarten Datei verhält sich das System genauso wie eine Implementierung mit Massenspeicheranbindung ohne Dateistrukturierung. Betriebssystemspezifische Befehle lassen sich über einen System-Call problemlos als FORTH-Worte einbinden, so daß z. B. bei Diskettenwechsel aus FORTH heraus mit DIR die Directory gelistet werden kann.

[2] Oft ist anstelle von **THEN** (FORTH83) die Bezeichnung **ENDIF** anzutreffen (figFORTH).

[3] Kleinste Stackeinheit, umfaßt 2 Byte

```
B:ROOT.SCR  Screen 26
  0 ( ASCII-DUMP   .ASCII   .ADR                          hoe 28.07.87 )
  1 : .ADR   BASE @ >R HEX 0 4 D.R R> BASE ! ;  ( gibt Adresse aus)
  2
  3 : .ASCII                ( adr --> ) ( gibt ASCII oder SPACE aus)
  4     C@ DUP 32 < OVER 127 > OR
  5     IF DROP 32 ENDIF EMIT ;
  6
  7 : ASCII-DUMP  ( start laenge --> ) ( gibt ASCII oder SPACE aus)
  8         BASE @ >R HEX OVER + OVER          ( start ende start --)
  9     DO  I OVER -  64 MOD 0=              ( 64 Zeichen pro Zeile)
 10         IF     CR 8 SPACES                        ( neue Zeile)
 11                I .ADR  SPACE                ( Adresse ausgeben)
 12         ENDIF
 13         I .ASCII                        ( Drucke ASCII oder SPACE)
 14     LOOP
 15         DROP R> BASE ! ;                              ;S
 ok
```

Bild 4 Screenbeispiel

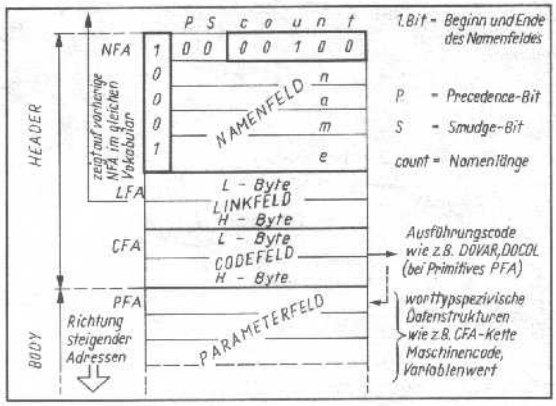

Bild 5 Lexikoneintrag

2.2.3. Screens

Ein Screen ist ein 1 KByte großer Speicherbereich und dient der Aufnahme von Quelltext oder Daten. Im RAM des Systems ist ein Bereich vereinbart (Block-Buffer-Area), der einen oder mehrere Screens aufnehmen kann. Die Screens werden vom Massenspeicher (i. allg. Diskette) geladen und dort z. B. mit einem Editor manipuliert oder entsprechend anderen Anweisungen verarbeitet. Der Anwender braucht sich um die Verwaltung der Buffer-Area nicht zu kümmern. FORTH kennzeichnet mit **UPDATE** einen Screenpuffer bei Veränderung desselben, wodurch der Screen automatisch zurückgeschrieben wird, wenn der Puffer für einen anderen Screen benötigt wird. Die Größe der Block-Buffer-Area kann aus den Variablen **FIRST** (Beginn) und **LIMIT** (Ende) errechnet werden. Ein Screen ist i. allg. in 16 Zeilen zu 64 Zeichen unterteilt. Zeile 0 sollte als Kommentarzeile reserviert werden. Mit z. B. **15 45 INDEX'** (listet die Zeilen 0 der Screens 15 bis 45 auf) ist dann eine schnelle Inhaltsübersicht über den ausgewählten Bereich möglich.

2.2.4. Editieren des Quelltextes

Üblicherweise wird der FORTH-Quelltext in Screens (Bild 4) notiert. Das geschieht unter Verwendung eines Screen-Editors, i. allg. selbst in FORTH geschrieben. Einen Line-Editor (zeilenweises Editieren) bietet fast jedes System; Full-Screen-Editoren (Bildschirm-Editoren) ermöglichen wesentlich komfortableres Arbeiten, sind aber nicht immer problemlos auf anderen Rechnern lauffähig. Die Anpassung bereitet jedoch aufgrund der Modularität von FORTH keine Schwierigkeiten. Die Verwendung von Screens zur Aufnahme von Quelltext entspricht der FORTH-Philosophie des modularen Aufbaus aller Bestandteile (also auch des Quelltextes), findet jedoch nicht uneingeschränkte Zustimmung. Prinzipiell ist es dem System egal, aus welcher Quelle (Tastatur, Diskette, serieller Datenkanal) der Text-Eingabe-Strom kommt. Einzig die FORTH-Konventionen (Spaces zwischen den Worten, keine unbekannten Zeichenfolgen, also Steuerzeichen) müssen eingehalten sein. Der Programmierer kann seinen hauseigenen Lieblingseditor weiterverwenden und muß lediglich seinen Quelltext im nachhinein entsprechend aufbereiten. In /8/ wird das am Beispiel eines allgemein bekannten Textverarbeitungsprogramms demonstriert, indem ein Softwarefilter für die entsprechenden Steuerzeichen installiert wird.

2.2.5. Transient Program Area und vorcompilierter FORTH-Code

Bei der Arbeit mit FORTH werden bestimmte Programmteile immer wieder benötigt. So z. B. ein Editor für die Editierphase und ein Debugger für die Testphase. Diese Tools werden in fertigen Applikationen i. allg. nicht mehr benötigt. Hat man sie vor der Compilierung des erstellten Programms geladen, geht mit z. B. **FORGET ASSEMBLER** auch das Arbeitsergebnis über Bord. Die im Arbeitsergebnis enthaltenen Primitive-Definitionen lassen sich aber nur mit einem bereits geladenen Assembler compilieren ... Bei kleineren Rechnerkonfigurationen kann auch der verfügbare Speicherbereich schnell ausgeschöpft sein. Will man dann z. B. Editor und Debugger abwechselnd laden, entsteht aufgrund der wiederholten Compilierung immer wieder Wartezeit, und vorbei ist es mit dem kurzen Wechsel zwischen Codierung und Test. Man kann also schnell in die mißliche Lage kommen, mehr Zeit und Aufmerksamkeit in die Verwaltung des eigenen Systems zu investieren als für die eigentliche Aufgabe. Um dem Programmierer an dieser Stelle das Leben zu erleichtern, bieten verschiedene FORTH-Systeme (z. B. /12/) eine sogenannte Transient Program Area (TPA) für unabhängig vom Wörterbuchausbau ladbare Vokabulare. Aufgrund der fest vereinbarten Anfangsadresse der TPA ist auch das Laden von vorcompiliertem FORTH-Code möglich und so ein spürbarer Zeitgewinn erreichbar.

2.3. Arithmetik

2.3.1. Notation

In-fix-Anweisungen (d. h. Operator zwischen den Operanden), wie z. B. in BASIC, müssen vom jeweiligen System vor Funktionsausführung intern in eine post-fix-Form (Operator nach den Operanden) umgesetzt werden. Aufgrund der Datenübergabe über einen Stack (2.2.1.) und der streng sequentiellen Textauswertung des Textinterpreters **INTERPRET** werden in FORTH arithmetische Operationen post-fix notiert. Diese Schreibweise ist unüblich und bedarf einiger Gewöhnung. Oft wird diese auch als UPN (umgekehrte polnische Notation) bekannte Schreibweise als Hauptargument bei einer ablehnenden Haltung gegenüber FORTH gebraucht. Wenn man die Philosophie dieser Sprache richtig verstanden hat, erkennt man in logischer Konsequenz die Notwendigkeit der post-fix-Notation. Soll auf die in-fix-Schreibweise partout nicht verzichtet werden, ist eine Erweiterung des Textinterpreters **INTERPRET** erforderlich – eine aufgrund der Zugänglichkeit von FORTH lösbare Aufgabe. Allerdings erhöht man damit den Overhead des Systems beträchtlich. Eine andere Möglichkeit zeigt dieses kleine Beispiel:

4 3 * .⟨RET⟩ **12 OK** (post-fix-MAL)
: MAL 32 WORD NUMBER DROP * ;⟨RET⟩
OK[4]
4 MAL 3 . ⟨RET⟩ **12 OK** (in-fix-MAL)

Dieses MAL ist jedoch innerhalb einer Colon-Definition nicht anwendbar und sollte mit **FORGET MAL** wieder „vergessen" werden.

2.3.2. Integerarithmetik

FORTH zielt nicht explizit auf die Offline-Verarbeitung numerischer Datenmengen, obwohl es prinzipiell auch dafür einsetzbar ist. Von Hause aus ist eine 2- und 4-Byte-Arithmetik verfügbar, die eine hohe Rechengeschwindigkeit bietet und einen Zahlenbereich von 65536 (2 Byte) bzw. über 4 Millionen (4 Byte) überstreicht. Für sein Steuerungsproblem wünscht man sich meist kurze Reaktionszeiten. Ist z. B. für eine schnelle Pegelanpassung ein Meßwert innerhalb kürzester

[4] Diese Definition bezieht sich auf FORTH83. Auf figFORTH-Systemen gilt aufgrund der etwas anderen Definition von **WORD** folgende Sequenz:
: MAL 32 WORD HERE NUMBER DROP * ;

Tafel 1 Scalierungsquotienten

Konstante	Approximation	Fehler
$\pi = 3{,}14159\ldots$	355/113	$< 10^{-7}$
	22/7	$< 10^{-3}$
$\sqrt{2} = 1{,}41421\ldots$	19601/13860	$< 10^{-8}$
	99/70	$< 10^{-4}$
$\sqrt{3} = 1{,}73205\ldots$	18817/10864	$< 10^{-8}$
	97/56	$< 10^{-4}$
$\sqrt{10} = 3{,}16227\ldots$	22936/7253	$< 10^{-8}$
	117/37	$< 10^{-4}$

Zeit zu logarithmieren, wäre das Auslesen einer vorberechneten Tabelle (und evtl. Interpolation) eine einfache und akzeptable Möglichkeit. In /15/ ist das anhand der Sinusfunktion gezeigt.
Der 2-Byte-Zahlenbereich ist für die Verarbeitung von Meßwerten z. B. aus einem Analog-Digital-Umsetzer ideal geeignet. Soll nun der Meßwert oder irgendein anderer Parameter mit einem beliebigen Faktor multipliziert werden (Skalierung), findet sich mit Sicherheit ein Quotient, der dem gewünschten Skalierungsfaktor bis auf einen sehr kleinen Fehler nahekommt. Tafel 1 gibt einige solche Quotienten für gängige Konstanten wieder. Weitere Werte sind z. B. in /1/, /8/ und /9/ zu finden. Mit Hilfe von */ wird aus dem Zähler und dem Stackwert ein doppelt genaues Zwischenprodukt berechnet und die Division durch den Nenner zu einer wieder einfachgenauen Zahl ausgeführt. Wie genau diese Rechnung wird, sollte man einmal mit Hilfe eines Taschenrechners nachvollziehen.

2.4. *Erweiterungen*

Nichtsdestotrotz wird so mancher potentielle [illegible]wender sagen: Ich habe genügend Rechenzeit, aber keine Zeit, mir über die Realisierung von arithmetischen Funktionen und die Überschreitung von Zahlenbereichen Gedanken zu machen. Nun, es läßt sich immer ein geeignetes Gleitkommapaket finden (siehe z. B. /10/), falls es nicht ohnehin schon vorhanden ist; entweder als FORTH-Erweiterung oder, wie in /11/ demonstriert, als Einbindung einer vorhandenen Gleitkommaarithmetik. Welches Format man einsetzen wird, ergibt sich aus den Erfordernissen und vorhandenen Möglichkeiten. Für hochgenaue Berechnungen findet man z. B. in der von der WPU Rostock angebotenen com-FORTH-Version /7/, /12/ eine überlange Integerarithmetik, deren bis zu 22 Byte langes Format keine Wünsche offenlassen dürfte. Wer mit komplexen Zahlen rechnet oder mit Strings hantiert, findet ebenfalls solche speziellen Systemergänzungen. Die Erweiterungsmöglichkeiten von FORTH sind bereits jetzt recht vielfältig und werden mit der weiteren Verbreitung dieser Softwarephilosophie [illegible]n bereichert werden. Dabei sind Bestrebungen zur Standardisierung zu beobachten /16/. Bevor der Anwender sich der Mühe unterzieht, selbst Standardfunktionen zu programmieren, sollte er die verfügbare Software kennen, um unnötigen Aufwand zu vermeiden.

3. Das FORTH-System

3.1. *Kompaktheit*

Das System ist äußerst kompakt. Schon in 2 KByte sind Grundstrukturen realisierbar. 8 KByte reichen für ein dialogfähiges System einschließlich Editor und Assembler, und in 16 bis 32 KByte lassen sich bereits übliche Applikationen unterbringen. Die Compilierung einzelner Module in ein neues Wort benötigt nur jeweils 2 Byte Speicherplatz in dessen Parameterfeld (s. 3.5.1.). Für turn-key-Systeme[5] und Applikationen für Zielsysteme ohne Dialogfähigkeit kann man bei Bedarf alle dann überflüssigen Konstruktionen wie Wortheader (vgl. 3.5.1.) und nicht (mehr) benötigte Definitionen entfernen und eine kaum noch zu überbietende Speicherausnutzung erreichen (vgl. z. B. /14/).

Dipl.-Ing. Thomas Hönenleitner beendete 1983 sein Studium an der Humboldt-Universität zu Berlin in der Fachrichtung Elektronik-Technologie mit der Entwicklung eines digitalen Reglers für einen Kristallzüchtungsofen. Sein bisheriges Tätigkeitsfeld umfaßt Steuerelektronik von Elektrofahrzeugen und prozeßnahe Automatisierungstechnik. Seit 1988 im Zentralinstitut für Kybernetik und Informationsprozesse der AdW tätig, liegt sein neues Aufgabengebiet im Bereich der Algorithmenentwicklung und Softwareerstellung für Systeme des maschinellen Sehens.

3.2. *Geschwindigkeit*

Die Abarbeitung des compilierten FORTH-Code durch die emulierte FORTH-Maschine benötigt nur etwa die vierfache Zeit /8/ gegenüber einem optimierten Assemblerprogramm. Dieser Wert variiert je nach eingesetztem Prozessortyp /1/, /6/, /7/ und gegebener Problemstellung. Die Anwendung der post-fix-Notation (vgl. 2.3.1.), das Konzept des gefädelten FORTH-Codes und die Arbeitsweise des inneren Interpreters halten den Overhead bei der Programmabarbeitung gering. Dazu kommen aufgrund der Maschinennähe und des modularen Aufbaus weitere Möglichkeiten. Angenommen, in einer Applikation werden 90 % der Programmlaufzeit von 5 % des Codes in Anspruch genommen (z. B. eine Such-, Sortier- oder Konvertierungsprozedur) und man möchte die Abarbeitungsgeschwindigkeit erhöhen, dann kann man, ohne die restlichen 95 % des Codes irgendwie modifizieren zu müssen, diese bisher als Secondary definierte Prozedur in Assembler neu codieren und als Primitive (s. 2.1.3.) in die Applikation einbinden. Die Gesamtlaufzeit des Programms erreicht dann fast die Werte, die optimierter Maschinencode bietet.

3.3. *Multitasking*

Für Echtzeitprobleme, aber auch bei der Offline-Verarbeitung von Daten, ist Multitaskingfähigkeit oft unverzichtbar. FORTH bietet auch das. Alle taskspezifischen Pointer sind als **USER-VARIABLE** angelegt, also über einen Offset, der in einer Variablen gehalten wird, erreichbar. Das Umschalten zwischen den einzelnen Tasks erfolgt einfach durch Verändern dieses Offsets. Zwei Verfahren sollen kurz genannt werden. Das einfachere, z. B. in der public domain Version F83 /8/ installierte, besteht darin, daß jede einzelne Task möglichst oft einen Befehl **PAUSE** enthält, der ein Weiterschalten zur nächsten Task ermöglicht. Zu diesem Zweck enthalten in F83 verschiedene, oft auftretende Anweisungen bereits dieses Wort (z. B. die Zeichen **EMIT**tierende Prozedur). Dieses unter der Bezeichnung **robin round** bekannte Prinzip /8/ ist für die meisten Anwendungsfälle völlig ausreichend. Für zeitkritische Applikationen kann man einen sogenannten Scheduler installieren, der die Tasksteuerung entsprechend einem Zeitplan und vorgegebene Prioritäten erledigt (Timesharing). Ein für den Softwareentwickler relevantes Multitask-Anwendungsbeispiel könnte beim Debugging die Anzeige aller wichtigen Systemzustände auf einem Zweitmonitor sein.

3.4. Interruptbehandlung

Wird vom Prozessor ein Interrupt akzeptiert, so kann nach dem Retten der notwendigen Register die Serviceroutine in Maschinencode ablaufen und nach der Rekonstruktion des ursprünglichen Prozessorstatus wieder an das FORTH-System übergeben werden. Soweit der einfache Fall. Interessant wird nun die Möglichkeit, Interruptserviceroutinen in FORTH zu programmieren. Nach dem bisher Gesagten kann man die Fragestellung darauf reduzieren, wie man von der Maschinenebene aus FORTH-Worte aufrufen und wieder in die Maschinenebene zurückkehren kann (um die Interruptserviceroutine mit der Registerrestaurierung und dem prozessorspezifischen Abschlußbefehl zu beenden.). FORTH ist auch in diesem Punkt offen: Das eine Doppelpunktdefinition beendende Wort ; (IMMEDIATE-Wort, s. 3.5.4.) compiliert eine Routine zum Abschluß der durch das später aktivierte Wort hervorgerufenen Prozesse (NEXT). Um ein aus der Maschinenebene heraus aufrufbares Wort zu kreieren (welches wieder dorthin zurückkehrt), kann genau hier angesetzt werden. Es ist lediglich der Sprungbefehl nach NEXT durch ein RETURN zu ersetzen. In /8/ wird dabei der unkomplizierte Weg über die Definition eines speziellen Abschlußbefehles **;RET**, der anstelle von ; eine Definition beschließt, beschritten. Mit einem Blick in das (oft nicht vorhandene) Listing oder mit Hilfe eines Discompilers erfährt man den Aufbau von ; und kann diese wenigen Zeilen (jetzt mit RETURN anstelle von NEXT!) neu codieren. Solche Worte sollten nie von FORTH aus direkt aufgerufen werden, wenn man sein System am Leben erhalten will. Der eigentliche Aufruf in der Maschinenebene kann nach Bereitstellung der Codefeldadresse (CFA, s. 3.5.1. u. 3.5.5.) über einen CALL in das Wort **EXECUTE** hinein realisiert werden. Die hier dargestellte Möglichkeit ist nicht die einzige. Weiteres ist u. a. in /1/ zu finden.

3.5. *FORTH von innen*

3.5.1. Struktur eines Lexikoneintrages

Bild 5 zeigt den prinzipiellen Aufbau eines figFORTH-Wortes. Die *Namenfeldadresse* (NFA) zeigt auf den Beginn des Namenfeldes, dessen erstes Byte neben der Längenangabe (count) zwei Kennungsbits enthält. Das *SMUDGE-Bit* S ist im allgemeinen 0. Noch nicht fertig definierte Wortstrukturen enthalten S=1 und sind damit vom System nicht aufrufbar. Das *Precedence-Bit* P ist bei allen IMMEDIATE-Worten (s. 3.5.4.) gesetzt. Alle FORTH-Worte eines Vokabulars sind über die jeweilige *Linkfeldadresse* (LFA) miteinander verkettet, was für die Suchläufe des Textinterpreters erforderlich ist. Sie hält einen Pointer auf die NFA des jeweils im gleichen Vokabular zuvor definierten Wortes. Die *Codefeldadresse* (CFA), auch mit Wortadresse bezeichnet, enthält die Startadresse einer worttypspezifischen Maschinenroutine. Gelangt ein FORTH-Wort zur Ausführung, übergibt der innere Interpreter (s. 3.5.5.) die Kontrolle an diesen Maschinencode. Beispielsweise transportiert DOCON, der Aus-

[5] Das fertige Programm wird als lauffähiger Code unter dem gewünschten Namen abgespeichert und läßt den Anwender von FORTH nichts mehr spüren.

führungscode des Worttyps Konstante, den Inhalt der *Parameterfeldadresse* (PFA) auf den Datenstack. Einige weitere symbolische CFA-Inhalte sind:

DOVAR: (DO VARiable) bringt die PFA auf den Stack (Worttyp Variable).

DOUSER: (DO USER-variable) bringt Summe von Inhalt der PFA und Inhalt von UP (User-Area-Pointer) zum Stack (Wortyp User-Variable).

DOCOL: (DO COLon) bringt den inneren Interpreter zur nächsttieferen Verarbeitungsebene. Dann Abarbeitung der Folge von CFAs im Parameterfeld (Worttyp Secondary).

PFA: Maschinencode befindet sich im Parameterfeld (Worttyp Primitive).

Prinzipiell kann der Maschinencode an beliebiger Stelle im RAM liegen. Er muß lediglich mit einem Sprung nach NEXT, dem Eintrittspunkt des inneren Interpreters, enden. Das Parameterfeld beginnt an der PFA. Hier ist der eigentliche Wortinhalt verborgen: Daten beliebiger Struktur, direkt ausführbarer Maschinencode oder eine CFA-Folge. Zum Beispiel:

Konstante:	Der Wert selbst.
Variable:	Der Wert selbst. (Beachte unterschiedliches Laufzeitverhalten!)
User-Variable:	Offset zum UP
Secondary:	CFA-Kette der definierten Wortfolge.
Primitive:	Maschinencode.

Der in Bild 5 gezeigte Wortaufbau ist nicht für alle FORTH-Versionen identisch. Bei F83 /8/ befindet sich die LFA vor der NFA, wodurch z. B. schnellere Dictionarysuchläufe möglich werden. Vor der LFA ist ein weiterer Zeiger VFA (*View Field-Address*) angeordnet, der auf den Quellscreen verweist, aus dem das betreffende Wort compiliert wurde. Mit **VIEW name** erreicht man den zugehörigen Quelltext. In der in /14/ beschriebenen Lösung liegt stattdessen zwischen LFA und CFA ein Pointer auf das betreffende Parameterfeld. Damit ist es möglich, eventuellen Assemblerhilfscode direkt im Codefeld unterzubringen, da dieses jetzt mehr als 2 Byte umfassen darf.

3.5.2. Memory Map

Bild 6 zeigt eine typische Memory Map. Diese Aufteilung ist stark implementations- und rechnerabhängig. Es soll lediglich eine Vorstellung vom inneren Aufbau vermittelt werden. Interessiert die Speicheraufteilung einer vorliegenden Implementation, kann man sich die Werte der einzelnen Zeiger vom System erfragen. Zum Beispiel:

RP0 ?⟨RET⟩ **3F74 OK**

RP0 ist eine User-Variable, welche den Kaltstartwert für den Returnstack-Pointer **RP** hält.

3.5.3. Zum Beispiel der äußere Interpreter

FORTH ist nicht nur offen nach außen, sondern auch nach innen. Es gibt keine Blackbox für den Programmierer, wie das bei anderen Programmiersprachen der Fall ist. Man kann sich überall umsehen und bei Bedarf auf einfache Art und Weise Modifikationen vornehmen. Zum Beispiel der äußere Interpreter: Nach der Systeminitialisierungsphase (Worte **COLD**, **WARM** und **ABORT**) befindet FORTH sich in einer Endlosschleife, dem äußeren Interpreter **QUIT**. Beide Stacks sind leer, und das System erwartet Zeichen (meist) von der Tastatur (**QUERY**). Sobald welche eintreffen, werden sie in den **TIB** (**T**erminal-**I**nput-**B**uffer) transportiert. Mit dem Erkennen eines ⟨RET⟩ (LFCR-Sequenz OAH ODH) wird die im **TIB** aufgebaute Stringfolge **INTERPRET**iert (Aufruf des Textinterpreters **INTERPRET**) und danach wieder mit **QUERY** auf Text gelauert (dies ist eine etwas vereinfachte Darstellung). Analog kann man die Arbeitsweise von **INTERPRET** verfolgen, wo man dann u. a. auf **EXECUTE** stößt und

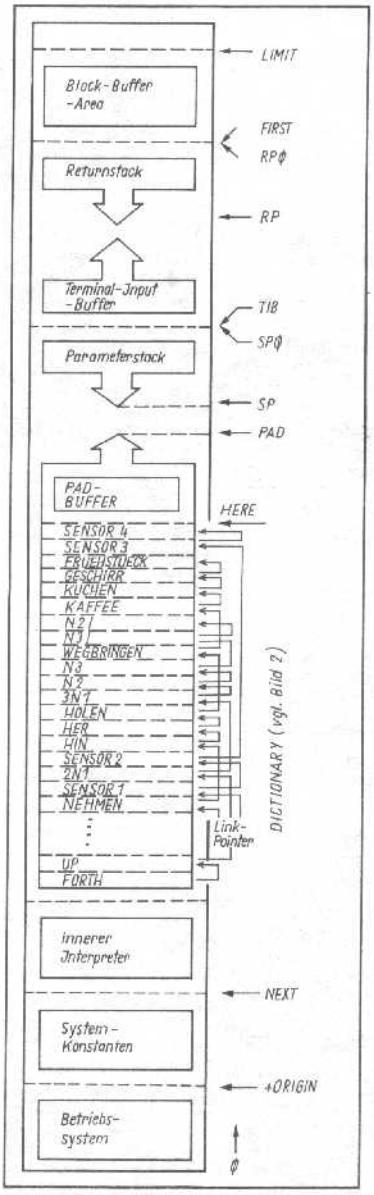

Bild 6 Memory Map

so zum inneren Interpreter gelangt. In gleicher Weise läßt sich das System bezüglich seiner Massenspeicheranbindung und aller anderen Komponenten beleuchten.

3.5.4. Der FORTH-Compiler

Der FORTH-Compiler wird durch verschiedene Compilerworte repräsentiert, welche als integrale Systemfunktionen im Zusammenwirken mit dem FORTH-Gesamtsystem die Quelltextcompilation vornehmen. Beispielsweise:

CREATE name
Ein Wortheader (vgl. Bild 5) wird im Dictionary aufgebaut.

, (n——)
Ein 2-Byte-Wort wird vom Stack in das Dictionary compiliert.

C, (n——)
Ein 1-Byte-Wort wird vom Stack in das Dictionary compiliert (High-Byte von n wird ignoriert).

ALLOT (n——)
Reserviert n Zellen im RAM an aktueller Erweiterungsposition im Dictionary.

IMMEDIATE
Das Precedence-Bit in der letzten Definition wird gesetzt (vgl. 3.5.1.).

Das FORTH-System unterscheidet anhand der User-Variablen **STATE**, ob es sich im compilierenden oder executierenden Mode befindet. Der Wechsel zwischen beiden Modi erfolgt mittels **]** (schaltet den Compile-Modus ein und **[** (suspendiert den Compiler, womit automatisch wieder der Execute- Mode erreicht wird). So verwendet z. B. das Definitionswort **:** u. a. die Worte **CREATE** und **]**. Die Compilerworte **[** und **SMUDGE** findet man u. a. im eine Colon-Definition beendenden Wort **;**. Der Textinterpreter **INTERPRET** prüft für jedes gefundene Wort den Systemstatus (**STATE** @) und übergibt die ermittelte CFA entweder an **EXECUTE** zur Ausführung oder ruft das Wort **, ,** welches dann die CFA ins Dictionary compiliert. Zahlen im Texteingabestrom werden im Execute-Mode zum Datenstack transportiert. Oder es erfolgt die Compilierung zusammen mit einem zahlenformattypischen Literalhandler.
Die streng sequentielle Textauswertung im FORTH-System ermöglicht u. a. das Umschalten in den Execute-Mode und wieder in den Compile-Mode mitten in einer eröffneten Colon-Definition. Diesen Umstand kann man z. B. nutzen, um Berechnungen während der Compilezeit ausführen zu lassen, um damit die Laufzeiten zu minimieren. Wurde nach der Definition eines Wortes mit **IMMEDIATE** dessen Precedence-Bit (Vorrangigkeits-Bit) gesetzt, wird es auch im Compile-Mode sofort executiert. Beispielsweise ist **;** ein solches IMMEDIATE-Wort. Es gibt noch weitere Compileranweisungen wie z. B. auch die selbstdefinierten Definitionsworte (vgl. 2.1.1.). Die hier erwähnten Worte verdeutlichen jedoch die prinzipielle Funktion für ein erstes Verständnis ausreichend.

3.5.5. Der innere Interpreter

Die prinzipielle Arbeitsweise des inneren Interpreters, das ist das Zusammenspiel der Register des FORTH-Prozessors (vgl. Bild 1) bei der Interpretation des gefädelten FORTH-Codes, ist in Tafel 2 skizziert.

Tafel 2 Funktionen des inneren Interpreters

```
Innerer Interpreter                 ; Kommentar                                          ! U880-Assembler

NEXT:                               ; Entry des inneren Interpreters                     ! NEXT:
                                    ; IP hält die aktuelle physische Adresse innerhalb des !        LD A,    (BC)
                                    ; Parameterfeldes des gerade in Abarbeitung befind-  !        LD L,    A
                                    ; lichen Wortes                                      !        INC      BC
      ((IP))→(W)                    ; Die dort befindliche CFA wird nach W gebracht. Dies !       LD A,    (BC)
                                    ; ist die nächste auszuführende Anweisung.           !        LD H,    A
      (IP) := (IP) + 2              ; Stelle IP auf nächste physische Adresse (Vorberei- !        INC      BC
                                    ; tung für NEXTes Wort).                             !
NEXT1:                              ; innerer Entry (z. B. für EXECUTE)                  ! NEXT1:
                                    ; W hält die zu executierende CFA.                   !        LD E     (HL)
      ((W))→ T                      ; Die Codeadresse CA, d. i. die Startadresse des wort- !      INC      HL
                                    ; spezifischen Ausführungscode, des jetzt zu execu- !        LD D,    (HL)
                                    ; tierenden Wortes (z. B. DOCOL, DOVAR oder anderer !        EX DE    ,HL
                                    ; Maschinencode), wird ausgelagert (Temporary)       !
      (W) := (W) +2                 ; W wird auf PFA des nun aktuellen Wortes gesetzt.   !        ; DE = CFA+1
      T→ PC                         ; Lade PC mit CA, damit Abarbeitung des Ausführungs- !        JP       (HL)
                                    ; code. Dieser endet i. a. mit einem Sprung nach NEXT !
                                    ; (nächstes Wort).                                   !

DOCOL:                              ; D. i. der allen Secondarys gemeinsame Ausführungscode ! DOCOL:
                                    ; (DO the COlon). Die Adresse DOCOL wird bei Aufruf  !        LD HL    ,(RPP)
                                    ; der Doppelpunktdefinition in neue CFA compiliert.  !        DEC      HL
      (RP) : = (RP) + 2             ; RP auf nächste Zelle setzen.                       !        DEC      HL
                                                                                         !        LD (RPP) ,HL
      (IP) → ((RP))                 ; Nächste physische Adresse der aktuellen Verarbei- !        LD (HL)  ,C
                                    ; tungsebene auf Returnstack retten.                 !        INC      HL
                                                                                         !        LD (HL)  ,B
      (W) → (IP)                    ; W wurde in NEXT auf PFA des aktuellen Wortes gesetzt. !     INC      DE
                                    ; Die dortige CFA wird dem IP übergeben und damit die !       LD C     ,E
                                    ; nächsttiefere Verarbeitungsebene erreicht.         !        LD B     ,D
      JUMP NEXT                     ; Und wieder von vorn (jetzt eine Ebene tiefer).     !        JP       NEXT

SEMIS:                              ; Diese Prozedur wird am Ende einer Doppelpunktdefini- ! SEMIS:
                                    ; tion mit dem Aufruf des Wortes ; in das Parameter- !        LD HL    ,(RPP)
                                    ; feld compiliert und bewirkt die Rückkehr in die    !        LD C     ,(HL)
                                    ; vorherige Verarbeitungsebene.                      !        INC      HL
      ((RP))→ IP                    ; IP zeigt ins Leere (Ende des Secondary) und wird mit !      LD B     ,(HL)
                                    ; dem letzten geretteten Wert (nächste physische     !        INC      HL
                                    ; Adresse der vorherigen Verarbeitungsebene) geladen. !       LD (RPP) ,HL
      (RP) := RP - 2                ; Rückstellen von RP.                                !        JP       NEXT
      JUMP NEXT                     ; NEXTes Wort in vorheriger Verarbeitungsebene.      !

PFADUP:                             ; Maschinencode des Wortes DUP (DUPliziere TOS)      ! DUP:
                                    ; befindet sich im Parameterfeld von DUP (2 Byte     !        POP      DE
                                    ; unter der CFA beginnend).                          !        PUSH     DE
      ((DP)+2) := ((DP))            ; Kopiere TOS in nächste RAM-Zelle.                  !        PUSH     DE
      (DP) : = (DP) +2              ; Setze Zeiger auf TOS neu.                          !        JP       NEXT
      JUMP NEXT                     ; Fertig zum NEXTen Wort.                            !
```

(x) -bezeichnet Inhalt von x
→ -symbolisiert eine Ladeoperation
:= -symbolisiert eine Neuzuweisung

! (IP) = BC – Doppelregister
! DP = SP – CPU-Stackpointer
! (W) = HL – Doppelregister
! T = DE – Doppelregister

Anhand der Routinen NEXT, DOCOL und SEMIS läßt sich gut verfolgen, wie der FORTH-Prozessor durch die einzelnen Verarbeitungsebenen zu den Primitive-Worten gelangt, diese ausführt (DUP steht als ein Beispiel) und letztlich in die höchste rufende Ebene zurückkehrt. Dies ist i. allg. der äußere Interpreter, die Endlosschleife **QUIT**, dessen compiliertes SEMIS nie erreicht wird. Zur Verdeutlichung der ablaufenden Prozesse sind die entsprechenden Code-Teile eines U880-Assemblerlistings mit dargestellt. Der modulare Aufbau von FORTH ist auch hier im Systemkern gut zu erkennen. Es bereitet kaum Mühe, die Codierung zu verstehen. Damit ist eine wesentliche Voraussetzung für eigene Modifikationen, Erweiterungen oder Neuimplementierungen gegeben. Die Wechselbeziehungen der einzelnen Register sind in /1/ ausführlicher dargestellt.

4. Schlußbetrachtung

Es ist nicht möglich, innerhalb eines Artikels auf alle Aspekte dieses Softwarewerkzeuges einzugehen. So mußten Betrachtungen zu verschiedenen verfügbaren Hilfsmitteln wie Discompiler und Tracer, Meta- und Cross-Compiler (z. B. /17/) oder turn-key-Funktionen ausgespart werden. Vektorielle Execution und Vorwärtsreferenzen seien ebenfalls nur genannt.

FORTH ermöglicht den Zugriff auf ausnahmslos alle Systemressourcen bei gleichzeitiger Verfügbarkeit als Hochsprache. Daraus resultiert eine erhöhte Verantwortung, denn das System erkauft diesen Komfort mit einer recht dürftigen Fehlerbehandlung. So gibt es z. B. keine Datentypenüberwachung wie etwa in der Sprache C. Aber auch hier kann bei Bedarf erweitert werden.

FORTH ist nicht leicht zugänglich. Der Einarbeitungsaufwand ist mit Sicherheit höher als z. B. bei Pascal. Der Anwender wird dafür jedoch voll entschädigt, wenn er einmal die Handhabung dieses Werkzeuges beherrscht und die dahinter stehende Philosophie /18/ verstanden hat.

FORTH-Quelltext ist nicht unbedingt leicht zu lesen. Die Zeit für eine hinreichende Dokumentation über jeden geschaffenen Modul sollte man im eigenen Interesse investieren. Rekapitulation ist bekanntlich recht mühselig.

Anwendungen sind u. a. auf dem Gebiet der Robotertechnik /19/, /20/ bekannt geworden. Das modulare Konzept, die Transparenz und die permanente Regenerierungsfähigkeit von FORTH-Systemen macht FORTH auch für die Mitgestaltung großer und größter Softwarepakete prädestiniert.

In FORTH Programmieren impliziert das systematische Aufbereiten der Aufgabenstellung. Gleichzeitig bleiben dem Programmierer viele Freiräume bei der Realisierung der einzelnen Komponenten. FORTH zeichnet sich durch sehr kurze Wechsel zwischen Codierung und Test aus. Diese interaktive Programmiertechnik ermöglicht schnelle Fehlerbeseitigung und hohe Produktivität /22/.

FORTH ist eine junge, noch im Wachsen begriffene Programmiersprache. Obwohl von Natur aus einer Standardisierung entgegenstehend, sind bisherige Bemühungen der Forth Interest Group (figFORTH, FORTH83) recht erfolgreich. Anpaßprobleme von Programmen an unterschiedliche FORTH-Versionen gibt es praktisch nicht, solange die gängigen Systemschnittstellen eingehalten werden. Fehlende Worte können schnell hinzu definiert werden.

Die bisher relativ geringe Akzeptanz von FORTH wird nicht unbedingt auf den höheren Einarbeitungsaufwand gegenüber anderen Programmiersprachen zurückgeführt, sondern ist in unzureichender Information über diese Programmiertechnik begründet. Diesem Umstand etwas zu begegnen, war Anlaß, diesen Artikel zu schreiben.

Literatur

/1/ Zech, R.: Die Programmiersprache FORTH. Franzis Verlag München 1984

/2/ Glasmacher, P.: FORTH in Silizium. c't Magazin für Computertechnik Heft 4/1987, S. 36–39

/3/ Forth direkt ausgeführt. mc Mikrocomputerzeitschrift Heft 6/1985, S. 63

/4/ Vack, G. U.: Hardware-Realisierung von FORTH. Mikroprozessortechnik, Berlin 1 (1987) 6, S. 169, 170

/5/ Odete, L.: Z8000 Forth. Dr. Dobb's Journal, Nr. 71, Sept. 1982, S. 48–63

/6/ Schiemann, B.: Ein fig-kompatibles FORTH für den U8000. Mikroprozessortechnik, Berlin 1 (1987) 6, S. 165, 166

/7/ Woitzel, E.: comFORTH – Ein flexibles Programmierwerkzeug zur Prozeßautomatisierung. Industriebeleg WPU Rostock 1985

/8/ Zech, R.: FORTH 83. Franzis Verlag München 1987

/9/ Brodie, L.: Programmieren in FORTH. Carl Hanser Verlag München 1986

/10/ Monroe, A. J.: Forth Floating-Point Package. Dr. Dobb's Journal, Nr. 71, Sept. 1982, S. 16 bis 29

(Fortsetzung auf Seite 44)

Universelles 3D-Grafikprogramm in einer Anwendung zur 2dimensionalen Schnellen Fourier-Transformation

Bodo Bachmann
VEB Mikroelektronik „Karl Marx" Erfurt

In vielen Fällen ist es günstig, eine Vorstellung vom Verlauf mathematischer Funktionen bzw. experimentell ermittelter Meßwerte zu gewinnen. Im folgenden wird ein Programm vorgestellt, welches Funktionen der Form z = f (x,y) räumlich, perspektivisch darstellt, wobei die grafischen Darstellungen mathematisch verdrehbar und somit aus beliebigen Richtungen betrachtbar sind. Daran anschließend wird ein Programm zur Schnellen Fourier-Transformation beschrieben, dessen Transformationspaare wirkungsvoll mit dem 3D-Grafikmodul dargestellt werden können.

3D-Darstellung

Um ein 3dimensionales Bild entsprechender Auflösung mit einem Mikrorechner auch in kurzer Zeit darzustellen, wurde die Programmiersprache FORTH verwendet. Dabei kamen die in /2/ und /3/ beschriebenen Erweiterungen des figFORTH-Standards zur Anwendung. Der Plotter robotron K 6418 diente als Zeichengerät. Es kann aber auch bei geringer Modifikation des Programmes ein Grafikdisplay angesteuert werden.
Das Programm setzt sich aus folgenden Hauptteilen zusammen:

1. Koordinatentransformation
2. Projektion (Parallelprojektion)
3. Zeichnen mit Weglassen der verdeckten Linien.

Die Theorie dazu ist ausführlich in /1/ beschrieben.
Vorausgesetzt wird eine Matrix von Funktionswerten des darzustellenden Objektes im x,y,z-Koordinatensystem. Durch die Koordinaten x und y wird eine Speicheradresse eindeutig charakterisiert. Auf dieser Adresse steht dann der Funktionswert z. Die Matrix wird also als Kette von Funktionswerten im Speicher abgelegt.
Mit den gewünschten Betrachtungswinkeln (Drehung und Kippung) transformieren sich die Koordinaten x,y,z zu den Koordinaten u,v,w.
Es werden gleich am Anfang die Schrittweiten in x- und y-Richtung transformiert, da dann in einem kartesischen Raster jeder Punkt durch deren Vektoraddition erreicht werden kann. Der Programmteil für die Transformation der y-Schrittweite ist in Bild 1 zu sehen. Es werden die transformierten Werte UY und VY, die die Schrittweite zum nächsten Punkt charakterisieren, auf dem Parameterstack (PS) bereitgestellt.
Das Programm entspricht den Formeln

$$UY = -\cos(\alpha)\,2000/(n-1)\,SGN(\sin[\alpha]) \quad (1)$$

$$VY = \sin(\alpha)\sin(\beta)\,2000/(n-1)\,SGN(\sin[\alpha]) \quad (2)$$

α bezeichnet den Drehwinkel, β den Kippwinkel, N die Dimension (max. 81), und 2000 ist dabei ein Normierungsfaktor. Schon hier ist der Vorteil eines extra Gleitkommastacks (GS) zu ersehen. So spart man sich einige SWAP-, ROT- oder OVER-Befehle.
Die x-Schrittweite wird analog transformiert. Nun kann die erste Matrixzeile in einen Zwischenspeicher geholt werden. Mittels des in Bild 2 gezeigten Wortes PROJ wird dann jeweils ein einzelner Punkt projiziert. Zu übergeben sind dabei auf dem Parameterstack UY, VY, LU, LV, I und auf dem Gleitkommastack cos(β).
LU und LV bezeichnen den aktuellen Stand des Plotters, I den Punktezähler.

Man erhält auf dem Parameterstack UY, VY (unverändert), LU', LV', U, V und auf dem Gleitkommastack wieder cos(β). U und V sind die vom Plotter anzufahrenden Koordinaten. Zu entscheiden wäre noch, ob die Linie auch gezeichnet wird. Das gleich an PROJ anzuschließende Wort LINE (Bild 3) überprüft den aktuellen Punkt mit den Koordinaten U und V, ob er evtl. durch andere Punkte verdeckt wird. Ist dies nicht der Fall, wird die Feder des Plotters gesenkt und die Linie bis zum Punkt gezogen, sonst bleibt die Feder angehoben.
Das Prinzip des Nichtzeichnens verdeckter Bildlinien beruht darauf, daß Bildlinien von Linien des Objektes, die dem Betrachter näher liegen, früher gezeichnet werden als die Bildlinien von Objektlinien, die weiter entfernt sind, und daß jede Bildlinie, die gezeichnet werden soll, mit einem unteren Grenzvektor verglichen wird. Sie wird nur dort gezeichnet, wo sie größer als der obere oder kleiner als der untere Grenzvektor ist. Beide Vektoren werden gleichzeitig auf den neuesten Stand gebracht, das heißt, die Bildlinie wird in die Grenzenbildung mit einbezogen.
Im Wort LINE werden die Adressen der V[illegible]gleichspunkte berechnet, und deren W[illegible] werden mit V verglichen. Ist der entsprechende Vergleichspunkt größer bzw. kleiner, so wird die Variable T inkrementiert. Nur wenn T den Wert 2 erreicht und wenn zusätzlich T beim vorherigen Punkt auch 2 erreicht hat (wird in 2T gemerkt), wird die Feder gesenkt. Jetzt werden die Koordinaten U und V für den Plotter normiert, erhalten einen Offset und können an den Plotter übergeben werden.
Es fehlen nun noch einige Rahmenwörter, die spezielle Winkelbereiche testen und die Schleifen zum Punktedurchlauf definieren. Diese sind relativ einfach und werden deshalb nicht extra erläutert.
Die Bilder 4 bis 6 zeigen einige Beispiele für mit diesem Programm gezeichnete Funktionen, deren Auflösung 41 mal 41 Punkte beträgt. In den Bildern 4 und 5 wurde allerdings nur jede zweite Linie gezeichnet. (Die Grafik aus Bild 5 verwendeten wir wegen ihrer Attraktivität bereits zur Gestaltung des

```
: Y-TRAFO                    (2 od. 1 -- UY VY)
-2000. N@ 1 -.*              ( (N-1)*2,wenn nur jede 2. Linie
                               gezeichnet wird          )
S->G                         (Zahl vom PS zum GS)
G/ VSA G@                    (VSA hält das Vorzeichen von sinα )
G* DUP SA G@                 (SA hält als Variable cosα )
G* W G@ G*                   (W=Maßstabsfaktor )
G->S SA G@ SB G@             (Zahl vom GS zum PS sinα, sinβ )
G* G* GVZ                    (GVZ = Vorzeichenwechsel )
W G@ G* G->S  ;
```

Bild 1 FORTH-Wort zur Schrittweitentransformation

```
: PROJ                       PS (I UY VY LU LV -- UY VY LU' LV' U V)
                             GS (      CB      --      CB          )
41700 + @                    ( z vom Zwischenspeicher          )
GDUP S->G G*                 (Multiplikation mit CB            )
SWAP DUP
5 PICK                       ( 5. Element auf oberste Stackpos.)
+ SWAP ROT DUP
5 PICK + ROT ROT
G->S +        ;
```

Bild 2 Berechnung der Projektionskoordinaten eines Punktes

```
: LINE                       ( U V -- )
0 T ! 2DUP SWAP S->G
D G@                         (De=Gleitkommavariable für Differenz
                               der U-Koordinaten zweier
                               benachbarter Punkte          )
G/ 0.5 G+                    (Offset zum Vergleichsvektor      )
G->S                         (Integer für Adreßrechnung        )
2* DUP 41070                 (Multiplikation mit 2, da 2-Byte-
                               Zahl                           )
+@ ROT <                     (1. Test auf Verdeckung           )
IF 2DUP 41070 + !
ELSE 1 T +!
ENDIF
2DUP 41490 + @ <             (2. Test auf Verdeckung           )
IF 2DUP 41490 + !
ELSE 1 T +!
ENDIF
DROP T @ 2 <
IF 2T@ 2 < IF PD;            (2T und T = 2, also Linie ziehen  )
           ELSE PU;
           ENDIF
ELSE PU;
ENDIF
SWAP 5 + 1C / 01@ +          (Normieren und Plotter Offset     )
SWAP 5 + 1C / 02@ +
PA T@ 2T !       ;           (T in 2T für nächster Punkt merken )
```

Bild 3 Zeichnen ohne verdeckte Linien