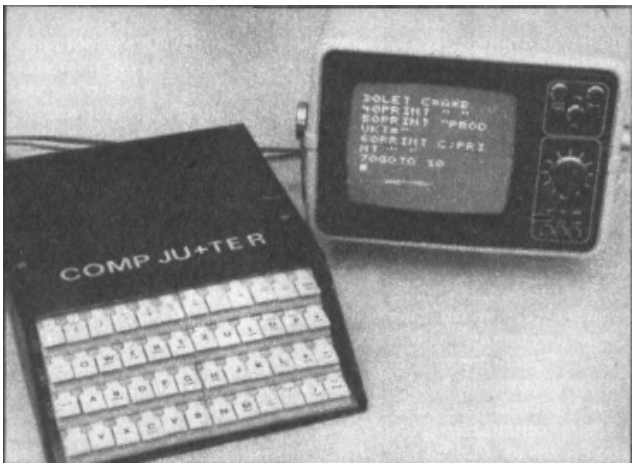
Dr. Helmut Hoyer

FORTH-83



Jugend+Technik 1/1990 - 8/1990

PDF-Version: Volker Pohlers, 2004

Inhalt

Teil 1, JU+TE 1/1990, S. 67-69

Programmieren in FORTH-83

FORTH zählt zu den höheren Programmiersprachen, bietet jedoch einige Besonderheiten. Das konsequente Aufbauen auf dem Dialog per Bildschirm und Tastatur vermeidet Mißverständnisse und sichert eine hohe Effektivität beim Umgang mit dem Rechner. Der Zwang zum strukturierten Programmieren führt zu klaren Denkweisen und sichert den Einstieg in moderne Programmiertechniken. FORTH ist damit interessant für Anfänger und Fachleute. Es existieren FORTH-Systeme für alle wichtigen Klein- und Personalcomputer (z. B. KC 85, C 64 und ZX Spectrum), nur leider oft, ohne Beschreibungen zur Hand zu haben. Unser ABC gibt deshalb eine Einführung in den Standard FORTH-83. Auf Unterschiede zu früheren Versionen (FIG, FORTH-79) wird hingewiesen. Die in den Text eingebauten Beispiele sollten sofort nachvollzogen und variiert werden. Dazu eignet sich auch das JU+TE-FORTH-System, das auf unserem Selbstbau-Computer läuft (vgl. JU+TE-Computerklub in diesem Heft, S. 70 bis S. 73).

Unser ABC soll bis Heft 6/1990 dauern. Wenn man dann die Programmiersprache FORTH verstanden hat, kann man Software entwickeln, die so schnell wie Maschinenprogramme ausgeführt wird, aber mit geradezu umgangssprachlichen Worten formuliert werden kann. Dazu stellen wir Euch nach einer kleinen Einführung Wortdefinitionen, die FORTH-Arithmetik, Strukturwörter und das Arbeiten mit Quelltextblökken vor. Auch Veröffentlichungen im JU+TE-Computerklub sollen das Verständnis erleichtern. Wie bei BASIC, braucht man auch bei FORTH nicht viel von der inneren Funktion des Computers zu wissen. Deshalb ist FORTH ebenso eine gute Anfängersprache. So setzt unser ABC also genaugenommen nur logisches Denken voraus.

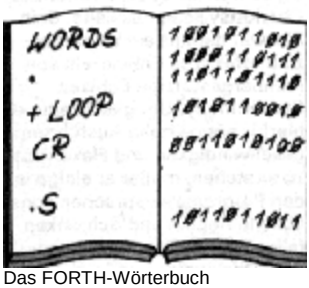
1. Einführung

FORTH ist eine ungewöhnliche Programmiersprache. Sie beruht auf sehr einfachen Prinzipien und läßt sich leicht erlernen. FORTH-Programme werden fast so schnell wie Maschinenprogramme ausgeführt. Trotzdem sind viel problemnähere Formulierungen als bei anderen Hochsprachen (BASIC, PASCAL, ...) möglich. FORTH ist nicht auf ein Anwendungsgebiet beschränkt, sondern hat einen überaus universellen Charakter. Es gibt kaum einen 8- oder 16-bit-Computer, für den kein FORTH-System existiert. Daraus resultieren eine sehr weitgehende Hardware-Unabhängigkeit und gute Voraussetzungen zur Anwendung entwickelter Software auf anderen Rechnern.
Für den Namen FORTH ist Charles Moore verantwortlich, der diese Sprache bereits vor zwanzig Jahren im Observatorium von Charlottsville entwikkelte. Sein Ziel war damals ein Werkzeug zum effektiven Entwickeln von rechnergestützten Echtzeit-Steuerungen. Um gleichzeitig die hierfür nötige hohe Ausführungsgeschwindigkeit und Flexibilität zu erreichen, mußte er einige in den Programmiersprachen sonst übliche Regeln und Schranken fallen lassen. Das hat eine andere Denkweise und Methode zur Folge, die das Umsteigen z. B. von BASIC auf FORTH erschweren. Neueinsteiger haben daher keinen großen Rückstand! Die Vorteile dieser Denkweise werden aber in hohem Maße aktuellen Forderungen (leichte Überschaubarkeit, gute Testmöglichkeiten und schnelle Veränderbarkeit der Programmteile) gerecht, so daß sich FORTH derzeit zunehmend verbreitet.

1.1. Das Wörterbuch

Die Aufgabe einer Programmiersprache besteht im Kern darin, zwischen der sprachlichen Formulierung des Problems und dem rechnereigenen Darstellungsformat (Dualzahlen) zu übersetzen. Das unterscheidet sich nicht viel vom Erlernen einer Fremdsprache zur Verständigung mit Menschen aus anderen Ländern. Computersprachen sind aber streng eindeutig und entschieden variantenärmer. Je mehr Begriffe der Aufgabenstellung in einer Computersprache enthalten sind, desto leichter fällt das Programmieren. Das entspricht auch der Abhängigkeit der Verständigungsschwie-

rigkeiten vom Umfang des Fremdwortschatzes.
Mit solchen Überlegungen wird klar, daß es eigentlich auf den Übersetzer ankommt. Er bestimmt den verfügbaren Wortschatz. In der jahrhundertelangen Erfahrung der Menschheit beim Umgang mit Fremdsprachen hat sich das Wörterbuch zum wichtigsten Hilfsmittel entwickelt. Was liegt nun näher, als auch mit Mikrorechnern auf diese Weise zu verfahren? Her-

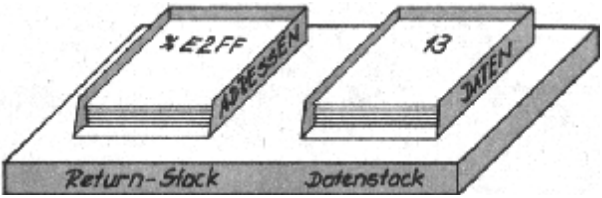


Das FORTH-Wörterbuch

kömmliche Programmiersprachen enthalten Tabellen für diesen Zweck, die im Prinzip jeder Anweisung die Startadresse des zugehörigen Maschinenprogramms zuordnen. Diese Wörterbücher sind jedoch beschränkt, beim TINY-BASIC-Interpreter des UB 8830 D z. B. auf 17 Eintragungen. Anwendungsspezifische Erweiterungen (z. B. Prozeduren) sind recht umständlich organisiert über eine weitere Tabelle zu vereinbaren. Beim praktischen Programmieren erfreuen sich diese Möglichkeiten wegen des Rechenzeitbedarfs und der Umstände bei der Übergabe variabler Größen (Parameter) keiner großen Beliebtheit.
Bei FORTH ist das Wörterbuch (engl.: **dictionary**) nach oben offen, es steht viel stärker im Mittelpunkt. Nicht nur die Standard-Anweisungen, sondern auch die Übersetzungsprogramme sind Bestandteile (**Wörter**) des Dictionary. Programmieren in FORTH heißt, das Wörterbuch so zu erweitern, daß die zu lösenden Aufgaben schließlich auch als ein Wort verzeichnet sind. Das Dictionary enthält somit alles, was der Übersetzer beherrscht. Natürlich gibt es Unterschiede wie zwischen Verben, Substantiven und dergleichen mehr. In FORTH ist durch die jeweilige Wortdefinition selbst festgelegt, ob es sich um eine Variable, Konstante oder Verarbeitungsaktion handelt, die sich hinter dem betreffenden Eintrag verbirgt. Wie wir mit diesen Kategorien umgehen, sehen wir später. Wesentlich ist, daß sich alle im Dictionary enthaltenen Wörter gleichermaßen schnell ausführen lassen, gleichgültig, ob sie zum Standard gehören oder nachgetragen wurden.

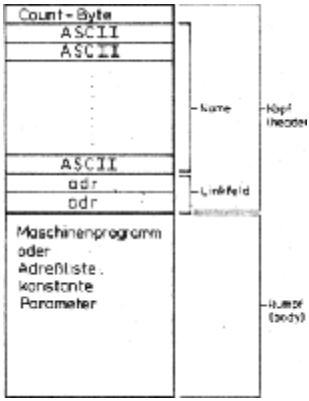
Abb. 1 zeigt den Aufbau einer Eintragung im Wörterbuch. Sie beginnt mit der Zahl (count) der Zeichen des **Wortnamens**. Diese können aus allen darstellbaren Buchstaben, Ziffern und Sonderzeichen (ASCII) bestehen. Nur das Leerzeichen (blank, space) ist verboten. Das gestattet das Verwenden von Fachbegriffen als Wortnamen und damit ausgesprochen anwendernahe Formulierungen. Da das Zählbyte im Wörterbuch neben der Zeichenzahl noch Zusatzinformationen enthält, ist die Länge des Wortnamens gewöhnlich auf 31 beschränkt, was trotzdem noch überaus viel Spielraum läßt. Dem Wortnamen folgt das **Linkfeld**, das die Speicheradresse der vorigen Eintragung angibt. Es gestattet dem Computer das Durchsuchen des Wörterbuches (link: verbinden). Der Maschinennahe Teil der Eintragung (body: Rumpf) besteht aus einer Folge von Unterprogrammsprüngen oder lediglich Unterprogramm-Startadressen (**Fadencode**) und zugeordneter Parameter (konstante Zahlen und Zeichenketten). Hierfür sind Bezeichnungen wie **Codefeld** und **Parameterfeld** üblich. Wie das im einzelnen organisiert ist, variiert von Rechner zu Rechner. Allgemein gilt, daß jede Eintragung dem Wortnamen (Kopf) das entsprechende Maschinenprogramm (Rumpf) zuordnet. Das Wörterbuch besteht aus vielen Eintragungen gemäß Abb. 1, die aneinandergereiht im Speicher stehen.
Die genaue Struktur interessiert eigentlich gar nicht. Wesentlich ist vielmehr, daß die Eintragungen den Wortschatz bilden, der für die Kommunikation mit dem Rechner taugt. Dazu gehört auch ein Wort, das den Inhalt des Wörterbuchs auf den Bildschirm bringt. Es heißt **WORDS** (Wörter), in früheren FORTH-Standards VLIST. Wir können uns also über den Wortschatz unseres FORTH-Systems informieren, indem. wir WORDS eingeben und mit der ENTER-Taste abschließen. Selbstverständlich können wir zunächst viel weniger mit dem Wortschatz anfangen, als der Rechner.

1.2. Der FORTH-Dialog



Die Unterhaltung mit dem FORTH-System hat ein ganz einfaches Prinzip: Wir geben ein Wort per Tastatur ein und bestätigen mit der ENTER-Taste. Der Rechner führt darauf das eingegebene Wort aus und meldet sich anschließend mit der Anzeige **OK**. Wenn wir nur die ENTER-Taste drücken, zeigt er sich zufrieden, nichts tun zu müssen und bringt auch sein **OK**. Nach Eingabe eines unbekannten Wortes (das nicht im Wörterbuch steht) macht er uns eine Mitteilung (message), die z. B. so aussieht:
OTTO (ENTER)
OTTO? #MSG 0
Anschließend ist er wieder für den normalen Dialog bereit. Übrigens steht (ENTER) für das Betätigen der ENTER-Taste, die Antwort des FORTH-Systems ist **halbfett** gedruckt. Nach Eingabe eines Wortes, dessen Ausführung andere Schwierigkeiten bereitet, erfolgt auch eine andere Mitteilung:
. (ENTER) **#MSG 1**
Der Punkt (dot) ist ein Wort, das jedes FORTH-System kennt. Es dient der Anzeige eines Zahlenwertes. Wir sehen, daß ein FORTH-Wort aus nur einem Zeichen, das nicht einmal ein Buchstabe ist, bestehen kann. Die Mitteilung #MSG 1 deutet auf einen leeren Datenstack (stack empty). Es ist also nichts da, was anzuzeigen ginge. Nach Eingabe einer Zahl kann der Punkt jedoch ausgeführt werden:
13 . (ENTER) **13 OK**
An diesem Beispiel zeigen sich zwei Dinge: Wir können erstens mehrere mit Leerzeichen getrennte Wörter eingeben, die das FORTH-System der Reihe nach ausführt, sobald die ENTER-Taste gedrückt wird. Zweitens versucht der Rechner, nicht bekannte Wörter als Zahleneingabe zu deuten und die sich zu merken. Dazu besitzt er eine Ablage (Datenstack oder einfach Stack).

•



,
: .S S0 @ 2 - SP@
DO CR I @ . 2 +LOOP ;
Achtung: bei FORTH-79:
'S statt SP@

1.3. Der Stack

Ein Stapelspeicher (engl.: stack) ist für einen Rechner etwas ganz Natürliches. Er braucht ihn grundsätzlich zum Aufbewahren der Rückkehradressen bei Unterprogrammsprüngen, also zum Organisieren grundlegender Funktionen. FORTH nutzt dieses Verfahren auch zur Datenübergabe von einem Wort zum anderen. Das FORTH-System verwaltet daher zwei Stapelspeicher: den ganz gewöhnlichen Return-Stack und den Datenstack. Beide sind Teil des vom Mikroprozessor adressierbaren Speichers (Hauptspeicher), der von Programme und anderes, wie z. B. das FORTH-Wörterbuch, enthält. Allgemein ist die Angabe der Speicheradresse nötig, um zu Speicherinhalten Zugang zu erhalten. Ein Stapelspeicher besitzt dazu einen Pointer (Zeigerregister), der die jeweils aktuelle Speicheradresse enthält. Bei jedem Eintrag in den Stapel erhöht sich die Stapeltiefe und entsprechend der Inhalt des Pointers. Die Entnahme einer Information aus dem Stack verringert die Stapeltiefe und den Pointerinhalt. Üblich ist allerdings, einen Stapel in Richtung fallender Adressen zu organisieren, so daß sich der Pointer stets entgegengesetzt zur Tiefe verändert, was jedoch auf das gleiche herauskommt. Wichtig ist, daß das Lesen vom Stapel in umgekehrter Reihenfolge zum Eintragen geschieht (last in - first out = LIFO) und daß gelesene Informationen vom Stapel verschwinden.
Bei FORTH wird der Datenstack gemeint, wenn vom Stack

die Rede ist. Den Return-Stack verwaltet das System völlig selbständig, was den Nutzer nur selten interessiert. Wir wollen den Stack jetzt einmal ausprobieren:
4 5 . . (ENTER) **5 4 OK**
Vor dem Betätigen der ENTER-Taste haben wir vier (wieder mit Leerzeichen getrennte) Aufgaben formuliert: das Eintragen der Zahl 4, das Eintragen der Zahl 5 und zweimal die Anzeige eines Wertes vom Stack. Interessant ist die Reihenfolge. Eingetragen haben wir erst die 4 und dann die 5. Unser FORTH-System zeigt jedoch die 5 zuerst an. Hierin äußert sich das LIFO-Prinzip. Der Punkt nimmt immer den obersten, also zuletzt eingetragenen Wert vom Stack. So findet er zunächst die 5 und entfernt sie vom Stapel. Danach ist die 4 an oberster Stelle. Der zweite Punkt bringt sie zur Anzeige und hinterläßt wieder einen leeren Stack. Dieses Prinzip eignet sich gut zur Parameterübergabe von einem FORTH-Wort zum anderen. Es ist wichtig für die hohe Ausführungsgeschwindigkeit, erfordert aber vom Programmierer, stets die Übersicht über den Stackinhalt zu gewährleisten. Wenn man sich daran gewöhnt hat, macht es auch keine größeren Schwierigkeiten als das Anwenden von Variablen in anderen Sprachen. Von besonderer Bedeutung ist offenbar der oberste Stapeleintrag. Er bildet die Spitze (top) und heißt TOS (top of stack). Das ist die Zahl, die als nächste entnommen wird. Ein neuer Eintrag avanciert selbst zum TOS und läßt den bisherigen an die zweitoberste Position rutschen. Das funktioniert alles wie bei einer Ablage, die nur von oben gefüllt und geleert wird.
Für den Anfänger ist es nicht leicht, die Übersicht über den Stackinhalt zu bewahren. Daher empfiehlt es sich, zunächst ein Wort zu definieren, das den Stapelinhalt anzeigt, ohne ihn zu verändern. **Abb. 2** gibt hierfür eine Möglichkeit an, die unter den meisten FORTH-Systemen funktioniert, obwohl nicht alle Wörter zum Standard gehören.
Dr. Helmut Hoyer

Teil 2, JU+TE 2/1990, S. 145-147

(Fortsetzung zu 1. 3. Der Stack)

Nach Eingabe dieser Definition (und nach Betätigen der ENTER-Taste, worauf wieder das OK kommen muß) steht uns das Wort .S zur Verfügung. Nach Rufen mit
.S (ENTER)
erzeugt der Rechner den Stackinhalt auf dem Bildschirm, ohne sonst etwas zu verändern. Daß wir zunächst nicht verstehen, wie das funktioniert, spielt keine Rolle. Ausprobieren läßt es sich trotzdem. Nach
WORDS (ENTER)
erscheint das neu definierte Wort .S an erster Stelle. Daran sehen wir, daß das FORTH-System das Wörterbuch von hinten beginnend durchsucht und die letzten Eintragungen als erste gefunden werden. Das LIFO-Prinzip ist offenbar allgegenwärtig.

2. Wortdefinitionen

2.1. Die Doppelpunktdefinition

Abb. 2 gibt auch ein Beispiel für die wichtigste Form des Erweiterns des Wörterbuchs: die **Doppelpunktdefinition**. Dazu gehören die FORTH-Worte : (colon) als Einleitung und ; (semicolon) als Abschluß. Die Funktion besteht darin, ein neues Wort zu kreieren, bei dessen Aufruf eine Folge bereits im Wörterbuch enthaltener Wörter ausgeführt wird. Nach dem Doppelpunkt ist zuerst der neue Name und dann das zugehörige FORTH-Programm einzugeben, wobei jeweils Leerzeichen die einzelnen Elemente trennen. Das Semikolon als letztes Wort bildet den Abschluß der Doppelpunktdefinition.
Der Doppelpunkt schaltet das FORTH-System vom ausführenden in den compilierenden (übersetzenden) Betrieb um. Das nächste Wort verwendet es als ASCII-Folge für den Namen im Kopf (header) eines neuen Wörterbucheintrags. Das Linkfeld wird eigenständig ergänzt. Aus den weiteren Wörtern der Eingabe erzeugt das FORTH-System das maschinenlesbare Pendant als Rumpf (body), anstatt sie gleich auszuführen. Durch das Semikolon wird der kompilierende Modus wieder verlassen.
Diesen Eigenschaften entsprechend erfolgt die Anwendung der Doppelpunktdefinition praktisch immer zum Erzeugen von ausführbaren Programmen. Es handelt sich dabei grundsätzlich um Unterprogramme. Das ist bei FORTH das normale Prinzip, das im Gegensatz zu Prozeduren in anderen Spra-

chen weder Umstände macht noch Ausführungszeit raubenden Organisationsaufwand erfordert. Die Parameter werden im Stack übergeben. Wir brauchen das beim Formulieren der Wortdefinition jedoch nicht anzuzeigen. Das FORTH-System überläßt es vollständig den einzelnen Wörtern, Daten auf den Stack abzulegen oder von dort zu entnehmen. Die Verantwortung trägt ohnehin der Programmierer. Für dessen Übersicht ist es üblich, zu jedem FORTH-Wort anzugeben, wie es mit dem Stack verfährt.
Runde Klammern schließen diesen Stackkommentar ein. Links stehen die vom Stack entnommenen, rechts die auf dem Stack abgelegten Parameter. Dazwischen befindet sich ein nach rechts gerichteter Pfeil oder einfach zwei Minuszeichen. Werden rechts oder links dieses Trennzeichens mehrere Parameter angegeben, erscheint der TOS am weitesten rechts.
(n adr – –)
bedeutet z. B., daß das so kommentierte Wort zwei Werte vom Stack nimmt, wobei der TOS als Adresse und der darunter liegende Wert als Zahl Verwendung findet. **Abb. 3** gibt eine Übersicht der in Stackkommentaren üblichen Abkürzungen. Man erkennt hier, wie unterschiedlich Stapeleinträge verwendet werden können. Im Grunde gibt es nur zwei Arten: den Standardeintrag (16 bit) und den doppelt genauen (32 bit). Dieser setzt sich aus zwei Standardeinträgen zusammen, von denen der höher im Stapel liegende (dem TOS nähere) die höheren 16 Stellen der 32-bit-Dualzahl enthält.

Die häufigste Deutung (**n**) ist die vorzeichenbehaftete Zahl, intern im Zweierkomplement dargestellt. Zahlen ohne Vorzeichen (**u**) gestatten eine höhere obere Darstellungsgrenze (vgl. Wertebereiche, in Abb. 3 mit eckigen Klammern eingeschlossen). Es gibt FORTH-Worte, die mit beiden Deutungen, also wahlweise n oder u funktionieren (**w**). Nur die obere Hälfte des Bereichs von n können Einträge annehmen, die mit **+n** symbolisiert sind. Allgemeine Deutung ohne Zahlenzuordnung kennzeichnet ein **b**, dem die Anzahl gültiger Bits vorangestellt wird. Auf eine Ja/Nein-Entscheidung reduziert sich die Deutung von Flags (**?**), wobei die Zahl 0 für „falsch" und alle anderen für „wahr" stehen. Mit Flag bezeichnet man in der Computertechnik ein Bedingungs-Flip-flop. Schließlich entspricht **adr** dem u, wird jedoch als Speicheradresse verwendet. Ein **d** kennzeichnet doppelt genaue Einträge.
Die Zahlenbereiche ergeben sich jeweils aus dem dualen Format, das intern immer angewendet wird. Eine Variable mit dem Namen BASE speichert die Zahlenbasis (base) für die Ein- und Ausgabe (Standardwert: 10). Deren Veränderung bewirkt äußerlich den Übergang auf ein anderes Zahlensystem (z. B. dezimal-hexadezimal). Dazu aber später mehr.

2.2. Die Variablendefinition

Zur Übergabe von Zahlen von einem FORTH-Wort zu einem später auszuführenden eignet sich der Stack am besten. Häufig werden Zahlen aber nicht nur einmal, sondern wiederholt gebraucht. Das läßt sich mit dem Stack nur selten gut lösen. Dafür gestattet FORTH, Variablen zu vereinbaren, deren Belegung wie auch bei anderen Programmiersprachen bis auf Widerruf gilt und beliebig oft gelesen werden kann. Einige, wie BASE, verwendet das FORTH-System selbst. Mit dem Wort **VARIABLE** können weitere definiert werden, natürlich ebenfalls als Wörterbucheintragungen.
VARIABLE OTTO (ENTER) erzeugt z. B das Wort OTTO, das zum Speichern einer Zahl taugt. Beim Ausführen von OTTO gelangt die Adresse deren Speicherzelle auf dem TOS. Das scheint zunächst recht umständlich, man hätte lieber sofort den Variablenwert. Der Umweg über die Adresse entspricht jedoch einer wesentlichen Grundidee bei FORTH, keine unnötigen Einschränkungen vorzunehmen, wo allgemeiner verwendbare Wörter möglich sind. Das Lesen des Variableninhalts gelingt nämlich mit @ (adr – - n). Das Wort @ (fetch = holen) nimmt eine Adresse vom Stack, legt statt dessen den Inhalt der so adressierten Speicherzelle auf den TOS. Genau genommen werden zwei Speicherzellen (je ein Byte) angesprochen, da FORTH standardmäßig 16-bit-Zahlen verarbeitet. Das Lesen des Variablenwertes OTTO erfolgt also mit
OTTO @ (ENTER).
Um eine Zahl, die natürlich als TOS erwartet wird, in die Variable OTTO zu laden, brauchen wir das Wort **!** (store = spei-

chern). Es nimmt eine Speicheradresse vom Stack und trägt dort den darunter liegenden Eintrag (als 16-bit-Zahl) ein (n adr --). Das Laden der Variablen OTTO mit der Zahl 13 gelingt mit den Wörtern
13 OTTO ! (ENTER) .
Es ist für den Anfang sehr hilfreich, die Wörter mit den jeweiligen Bedeutungen auszusprechen. Unsere beiden Beispiele lauten dann „OTTO holen" und „Dreizehn (in) OTTO speichern". Das ergibt sehr direkte Assoziationen zur Wirkung und damit Sicherheit in der Anwendung der FORTH-Wörter. @ und ! lassen sich nicht nur auf Variable anwenden, sondern gestatten allgemein den Speicherzugriff. Wie die Adresse auf den TOS gelangt, ist für diese Wörter völlig belanglos. Das läßt den angestrebten großen Anwendungsspielraum. Das in den Stackkommentaren übliche n müßte streng genommen mit w oder 16 b bezeichnet sein, da auch das Darstellungsformat für @ und ! keine Rolle spielt.
In FORTH-83 ist keine Initialbelegung (Anfangswert) für neu vereinbarte Variable festgelegt. Deshalb erhalten wir nach Ausprobieren von OTTO @ und z. B. .S keine sinnvolle Zahl als TOS. Das stört praktisch nicht, da in ordentlichen Programmen immer irgendein Wert in eine Variable geladen wird, bevor das erste Lesen erfolgt. Im alten FIG-Standard erwartet schon die Definition einer Variablen diese Zahl als TOS. Das heißt, daß hier der Stackkommentar (n – –) für VARIABLE gilt. FORTH-79 verlangt wie FORTH-83 keinen Anfangswert:(– –).

f

16-bit-Einträge:

n	Zahl [– 32768, 32767]
u	Zahl [0, 65535]
w	Zahl [–32768, 65535]
+n	Zahl [0, 32767]
16b	Eintrag ohne Zahlendeutung
8b	Eintrag mit nur 8 gültigen Bits
?	Flag
adr	Adresse

32-bit-Einträge

d	Zahl[–2147483648, 2147483647]
ud	Zahl [0, 4294967295]
wd	Zahl [–2147483648, 4294967295]
+d	Zahl [0, 2147483647]
32b	Eintrag ohne Zahlendeutung

2.3. Die CREATE-Definition

Aus der Sicht von FORTH ist die Variablendefinition schon etwas Spezielles. Hier werden nämlich ein Wortkopf mit dem nach VARIABLE angegebenen Namen erzeugt und zwei Byte zum Speichern des Variablenwertes reserviert. Beim Aufruf dieses Namens erhalten wir danach die Adresse der beiden reservierten Speicherzellen.
CREATE (schaffen, hervorbringen) beschränkt sich auf den ersten Teil dieses Mechanismus und überläßt den Rest dem Programmierer. Dieses Wort erzeugt ebenfalls einen Wortkopf mit dem anschließend eingegebenen Namen, reserviert aber keinen Speicherbereich. CREATE verändert den Stack nicht (– –).
Das Erzeugen des Rumpfes ist fast nur an die Bedingung geknüpft, dessen Länge mit entsprechender Einstellung der Systemvariablen DP (dictionary pointer) in der Wörterbuchverwaltung zu berücksichtigen. Wörter wie ALLOT (zumessen) und , (Komma) erledigen das selbständig. Wir merken uns erst einmal nur, daß CREATE einen Wortkopf zur allgemeinen Verwendung definieren läßt. Im Rumpf können z. B. mehrere zusammengehörige Variable (Vektoren) oder Maschinenprogramme untergebracht werden. Auch wenn das im Einzelnen noch nicht zu

„

Definitionswörter

: (– –)
Eröffnung einer Doppelpunktdefinition in der Form
: name wortfolge ;
; (– –)
Abschluß einer Doppelpunktdefinition. Die Wortfolge wird als Rumpf unter dem angegebenen Namen ins Wörterbuch eingetragen und bei Aufruf des Namens ausgeführt
VARIABLE (– –)
Definition einer Variablen in der Form VARIABLE name. Unter dem angegebenen Namen werden zwei Byte reserviert, deren Adresse bei Aufruf des Namens als TOS erscheint.
CREATE (– –)
Definition eines Wortkopfes in der Form CREATE name. Es wird kein Rumpf erzeugt. Bei Aufruf des Namens erscheint die Adresse des ersten Bytes des anderweitig zu erzeugenden Rumpfes als TOS.
CONSTANT (n - –)
Definition der Konstanten n in der Form CONSTANT name. Bei Aufruf des Namens erscheint die Zahl n als TOS.
FORGET (– -)
Streichen von Definitionen in der Form FORGET name. Das Wort name und alle danach definierten verschwinden aus dem Wörterbuch.

verstehen ist, läßt sich CREATE schon mal ausprobieren. Mit
CREATE UDO 2 ALLOT (ENTER)
entsteht eine Variable mit dem Namen UDO, die die gleichen Eigenschaften wie OTTO besitzt. Das läßt sich mit Hilfe von @ und ! gut testen. Wenn wir aber CREATE UDO ohne 2 ALLOT definieren, bringt (Zahl) UDO ! das System zum Absturz. Das Laden einer Variablen, für die kein Platz reserviert wurde, kann eben nicht gut gehen. Wir sehen, daß die Verwendung von CREATE wohl überlegte Zusatzvereinbarungen verlangt. Ein Kopf ohne Rumpf zeigt sich als unvollkommenes Wort.

2.4. Die Konstantendefinition

Der FORTH-83-Standard erlaubt auch, feste Werte mit einem Namen zu versehen und in das Wörterbuch aufzunehmen. Der Wert ist zunächst als TOS einzutragen. Das Wort **CONSTANT** erzeugt mit Hilfe von CREATE den Kopf und trägt den TOS in den Rumpf ein. Wird das so definierte Wort gerufen, erzeugt es diesen Wert als obersten Stapeleintrag. Wir können das z. B mit dem Erdradius ausprobieren:
6366 CONSTANT RADIUS (ENTER)
Mit WORDS läßt sich überprüfen, ob das FORTH-System das Wort RADIUS ins Wörterbuch übernommen hat. Von nun an erhalten wir nach Aufruf dieses Wortes stets die Zahl 6366:
RADIUS . (ENTER)
6366 OK
Der Zahlenwert wird als Dualzahl abgelegt und verändert sich nicht beim Wechseln der Zahlenbasis. Mit
HEX (ENTER)
oder
16 BASE ! (ENTER)
können wir hexadezimale Ein- und Ausgabe vereinbaren. Als Erdradius zeigt das FORTH-System nun
RADIUS . (ENTER)
18DE OK
an. Das ist der hexadezimale Wert, der gleich dem dezimalen 6366 ist. Nach
DECIMAL (ENTER)
oder
0A BASE ! (ENTER)
gilt wieder die Standard-Zahlenbasis.

2.5. Das Löschen von Definitionen

Besonders beim Ausprobieren wird das Wörterbuch nach und nach mit Eintragungen belastet, die dann keine Rolle mehr spielen. Das läßt sich vermeiden, indem wir unnötige Wörter streichen.
Mit **FORGET** (vergiß) entfernt das FORTH-System den anschließend benannten und alle folgenden Einträge aus dem Wörterbuch. Sollen die vereinbarten Variablen und Konstanten wieder verschwinden, muß lediglich
FORGET OTTO (ENTER)
eingegeben werden. Außer OTTO streicht das FORTH-System auch die später definierten Wörter UDO und RADIUS. Mit WORDS können wir uns davon überzeugen.
Es ist nicht möglich, OTTO zu streichen und UDO oder RADIUS stehenzulassen. Das heißt, unter Umständen eine wertvolle, aber erst später als die zu streichenden definierte Eintragung notieren und nach Ausführen des FORGET-Wortes erneut eingeben zu müssen. Deshalb sollten wir uns daran gewöhnen, beim Arbeiten im Dialogbetrieb vor der Eingabe neuer Wörter stets mit WORDS zu prüfen, ob die letzten Eintragungen noch gebraucht werden. Das sichert das rechtzeitige Anwenden von FORGET.
Abb. 4 faßt die Definitionswörter zusammen. Neben den Stackkommentaren ist dort auch kurz deren Anwendung notiert.

Teil 3, JU+TE 3/1990, S. 236-238

3. Rechenoperationen

Das Rechnen gehört zu den wichtigsten Aufgaben eines Computers. Bei FORTH beziehen die betreffenden Wörter die Operanden natürlich aus dem Stack und legen die Ergebnisse auch dort ab. Dementsprechend wird meist mit 16-bit-Zahlen (wie in TINY-BASIC) gerechnet. Einige Wörter gestatten auch das Rechnen mit 32-bit-Zahlen. Das wird doppelte Genauigkeit genannt, obwohl sich (wie in Abb. 3 deutlich) viel größere Wertebereiche ergeben. Das Rechnen mit Gleitkommazahlen (wie in extended BASIC) ist bei FORTH nicht üblich, läßt sich aber mit zusätzlichen Wörtern ergänzen. Eine viel einschneidendere Besonderheit resultiert aus der Datenübergabe im Stack. Ein

Wort, das eine Rechenoperation ausführt, erwartet bereits alle Operanden im Stack. Deshalb ist das Operationswort nicht zwischen, sondern nach den Operanden einzugeben. Diese Schreibweise heißt Postfix-Notation im Gegensatz zu der in den meisten Programmiersprachen üblichen In-fix-Notation. Wegen der Einführung dieser Rechenzeitsparenden Stack-bezogenen Eingabefolge durch einen polnischen Mathematiker heißt sie auch umgekehrt polnische Notation (UPN). Praktisch sieht das so aus: Zum Addieren der Zahlen 13 und 9 muß wie folgt eingegeben werden:
13 9 + (ENTER) **OK**
Das Ergebnis läßt sich mit dem Punkt anzeigen:
. (ENTER) **22 OK**
Die Wortnamen für Rechenoperationen sind zunächst ganz einfach die Operationszeichen. Hinzu kommt die Buchstabenfolge **MOD** (modulo), wenn ein Divisionsrest berechnet wird. Als Vorsätze dienen **D** (für doppelte Genauigkeit), **U** (für vorzeichenloses Zahlenformat) und **M** (für gemischte Formate: z. T. einfache und z. T. doppelt genaue Operanden und Ergebnisse, engl.: mixed). So sagt der Name stets, was hier passiert. Die Reihenfolge der Operanden und Ergebnisse im Stack macht der Stackkommentar deutlich.

3.1. Die Grundrechenarten

In **Abb. 5** sind die vier Grundrechenarten und das Berechnen des Divisionsrestes dargestellt. Es werden stets zwei einfache Operanden im Stack erwartet und durch ein einfaches Ergebnis ersetzt. Die Strichrechenarten funktionieren mit vorzeichenlosem und vorzeichenbehafteten 16-bit-Format, die Punktrechenarten arbeiten vorzeichenbehaftet. Wichtig ist die Operandenfolge: Auf TOS sitzt der zweite Operand. Daraus ergeben sich die Eingabefolgen Subtrahend Minuend Subtraktionsstrich für die Subtraktion und Dividend Divisor Divisionsstrich für die Division. Bei der Berechnung des Divisionsrestes gilt entsprechendes. Anhand einfacher Beispiele läßt sich das gut ausprobieren. Eventuelle Formatüberschreitungen führen nicht zu Fehlermeldungen und werden auch nicht anderweitig angezeigt.

3.2. Die Einoperanden-Operationen

Abb. 6 zeigt die Rechenoperationen, die einen einfachen Operanden verarbeiten. Die ersten fünf funktionieren genau so, als würden die beiden Zeichen des Namens mit Leerzeichen getrennt eingegeben, also Grundrechenarten ausgeführt. Mit den speziellen Operanden (1 und 2) lassen sich diese Operationen mit viel schnelleren Maschinenprogrammen ausführen. Deshalb gibt es diese FORTH-Wörter, die durch besonders kurze Ausführungszeiten gekennzeichnet sind.
Das offene Format w gestattet, die Zähleroperationen 1+, 1-, 2+ und 2- zur Adreßberechnung zu verwenden. Das begünstigt Programme, die die Speicherverwaltung z. B. für die Behandlung von Feldvariablen eigenständig ausführen. Darauf kommen wir aber später noch zu sprechen. ABS und NEGATE bewirken das Berechnen des absoluten Betrages bzw. den Vorzeichenwechsel. Zu beachten ist, daß beide Operationen die Zahl -32768 nicht verändern. Im vorzeichenlosen Format (u) wird dieser Wert als 32768 gedeutet.

3.3. Die Kombinationen

Kombinationen von Rechenoperationen einfacher Operanden sind in **Abb.7** zusammengefaßt. **/MOD** übergibt den Quotienten und darüber (als TOS) den Divisionsrest von n1/n2. Das bedeutet das Zusammenfassen zweier Grundrechenoperationen in der Ausführungszeit, die sonst für eine der beiden gebraucht wird. Darüber hinaus gibt es keine Unterschiede zu den Wörtern / und MOD.
Anders verhält es sich mit der Kombination von Multiplikation und Division mit ***/**. Im Gegensatz zu * wird ein 32-bit-Zwischenergebnis erzeugt, das Bereichsüberläufe an dieser Stelle ausschließt. Deshalb eignet sich das Wort */ gut zum Umrechnen zwischen unterschiedlichen Maßen. Dazu wird mit dem Referenzwert des Zielsystems multipliziert und durch den entsprechenden Wert des Quellsystems dividiert. Als Beispiel soll die Umrechnung von Millimeter in Zoll dienen. 254 mm sind gleich 10" (Zoll). Wieviel Zoll passen nun in sieben Meter? Mit

7000 10 254 */ . (ENTER) **275 OK**
erhalten wir das Ergebnis. Die Eingabe
7000 10 * 254 / . (ENTER)
hat zwar die gleiche Operationsfolge, führt aber zu einem falschen Ergebnis. Das liegt einfach daran, daß das Zwischenergebnis 70000 (nach der Multiplikation) nicht in dem für n gültigen Bereich liegt. Nun kann man aber, um das zu vermeiden, die Division vorziehen. Mit
7000 * 254 / 10 . (ENTER) **270 OK**
fällt das Ergebnis allerdings recht grob aus, da die Nachkommastellen der Division fehlen. Das läßt die Vorteile des Wortes */ deutlich werden. Zum allgemeinen Umrechnen können wir die Wörter ZOLL und MM wie folgt definieren:
: ZOLL 10 254 */ ; (ENTER)
: MM 254 10 */ ; (ENTER)
ZOLL rechnet (wie oben) den TOS von mm in " um, während MM ein Zollmaß in das metrische System wandelt. Nachkommastellen fallen wegen des Festkommaformats unter den Tisch. Statt dessen kann bei Verwendung des Wortes ***/MOD** der Divisionsrest berücksichtigt werden. Es befindet sich dann als TOS über dem Ergebnis n4 (vgl. Abb. 7) und bezieht sich auf zehn Zoll bzw. 254 Millimeter.

3.4. Die Operationen mit doppelt genauen Zahlen

Die Wörter */ und */MOD nutzen gewissermaßen zwischendurch das 32-bit-Format, die Operanden und Ergebnisse besitzen jedoch durchweg einfache Genauigkeit. Deshalb zählen sie zu den 16-bit-Operationen. In **Abb.8** sind die arithmetischen Verknüpfungswörter zusammengestellt, bei denen sich doppelt genaue Zahlen unter den Übergabeparametern befinden.
Der auch hier zugrundeliegende Standard FORTH-83 enthält nur die wichtigsten Wörter, während FIG noch mehr Varianten umfaßt. Zu beachten ist neben Namensunterschieden die bei FIG abweichende Ergebnis-Parameterstruktur von U/MOD (FORTH-83: UM/MOD). Häufig enthalten FORTH-83-Systeme eine Reihe dieser Varianten aus dem FIG-Standard. Die zum Erweiterungswortschatz zur Verarbeitung doppelt genauer Zahlen für den FORTH-83-Standard gehörenden sind in Abb. 8 vermerkt. Obwohl weit verbreitet nutzbar, können sie in FORTH-83-Systemen aber nicht grundsätzlich vorausgesetzt werden. Standardprogramme, die sich auf andere Rechner übertragen lassen, sollten solche Wörter nicht verwenden oder auf FORTH-Niveau neu definieren.
Das umfangreiche Ausprobieren der Operationen mit doppelt genauen Zahlen ist sehr zu empfehlen, da es auch den Umgang mit dem Stack übt. Unter anderem zeigt sich gut die Aufteilung doppelt genauer Zahlen auf zwei Stapeleinträge.

3.5. Die logischen Operationen

FORTH-83 gestattet, die 16 Stellen eines Stapeleintrages auch als Boolsche (binäre) Variable zu behandeln. Diesem Zweck dienen die in **Abb. 9** dargestellten Wörter zur logischen Verknüpfung. Sie sind für das Programmieren von Binärsteuerungen wichtig. Dabei sollte mit
2 BASE ! (ENTER)
die Ein- und Ausgabe im Dualen vereinbart werden. Auch das Anzeigen des Stapelinhalts mit STACK oder S. erfolgt dann binär. Beim Ausprobieren der logischen Operationen stört dann immer noch, daß führende Nullen weggelassen und z. T. negative Zahlen erkannt werden. Die Verwendung von U. statt . beseitigt wenigstens den zweiten Mangel (vorzeichenlose Ausgabe.)

4. Stackmanipulationen

Die Zahleneingabe in den Stack, die Ausgabe aus dem Stack und das Rechnen mit den dort gespeicherten Zahlen haben wir bereits erprobt. Beim Entwurf praktischer Programme liegen die Datenfolgen selten so günstig im Stapel, daß sie sich wie bei den Beispielen ZOLL und MM unverändert verarbeiten lassen. Es kommt auch vor, daß ein Parameter für eine Rechnung vom Stack entnommen und kurz darauf wiederholt benötigt wird. FORTH wäre außerordentlich unvollkommen ohne Wörter, die Stapeleinträge vervielfältigen und umsortieren lassen.

4.1. Die Standardwörter

Das Wort **DEPTH** (Tiefe) fällt etwas aus dem Rahmen. Es verändert den Stack nur insofern, daß die Anzahl der bisherigen Eintragungen als zusätzliche Zahl erscheint. Es ist in FORTH-83 erstmalig aufgenommen und gestattet, jederzeit den Füllgrad des Stapelspeichers zu ermitteln. Besonders beim Testen neuer Wortdefinitionen läßt es den sauberen Umgang mit dem Stack überprüfen. Unnötige Entnahmen und Hinterlassungen stören nämlich gewaltig.
Die einfachste Manipulation besorgt **DUP** (duplizieren): es kopiert den TOS. Das braucht man z. B., wenn eine Meterangabe in Yards und in Zoll umzurechnen ist (1000 Yds = 914 m):

```
: Y". DUP 1000
  914 */ . 10000
  254 */ . ; (ENTER)
```

Im Stack wird nur eine Zahl erwartet: die Meterangabe. DUP kopiert sie, da die erste Umrechnung diesen Wert einmal vom Stack entnimmt. So bleibt diese Angabe ein zweites Mal für die Umrechnung in Zoll. Komplizierter wird das Problem, wenn die Ergebnisse nicht (wie oben) angezeigt, sondern im Stack abgelegt werden sollen. Die folgende Definition erfüllt diesen Zweck:

```
: Y" DUP 1000
  914 */ SWAP
  10000 254 */ ; (ENTER)
```

Abb. 10 gibt als Beispiel die Umrechnung der Strecke von 150 m in 164 Yds und in 5905 " mit einer Tabelle des jeweiligen Stackinhalts (TOS wie immer rechts). Beim Eintritt steht nur die Meterzahl (n1 = 150) im Stack, zuletzt die Yard-Entfernung (n2 = 164) und als TOS das Zollmaß (n3 = 5905). Zwischendurch wird das Wort **SWAP** (tauschen) benutzt, um n1 auf TOS zu bringen, ohne das dort befindliche n2 zu zerstören. SWAP tauscht allgemein die beiden obersten Stapeleinträge.

... Grundrechenarten

+ (w1 w2 -- w3)
w3 ist die Summe von wl plus w2.

- (w1 w2 -- w3)
w3 ist die Differenz w1 minus w2.

***** (n1 n2 -- n3)
n3 ist das Produkt von n1 und n2.

/ (n1 n2 -- n3)
n3 ist der Quotient n1/n2.

MOD (n1 n2 -- n3)
n3 ist der Divisionsrest aus n1/n2.

† Einoperanden-Operationen

1+ (w1 -- w2)
w2 ist die Summe w1 plus 1.

1- (w1 -- w2) w2 ist die Differenz w1 minus 1.

2+ (wl -- w2)
w2 ist die Summe w1 plus 2.

2- (w1 -- w2) w2 ist die Differenz w1 minus 2.

2/ (n1 -- n2)
n2 ist der Quotient n1/2.

NEGATE (n1 -- n2)
n2 ist die Differenz 0 minus n1.
Achtung! FIG: Wortname: MINUS

ABS (n -- u)
u ist der Absolutbetrag von n.

‡ Kombinierte Operationen

/MOD (n1 n2 -- n3 n4)
n3 ist der Quotient n1/n2 und n4 der Rest aus dieser Division.

***/** (n1 n2 n3 -- n4)
n1 und n2 werden zu einem doppelt genauen Zwischenergebnis multipliziert, das nach Division durch n3 dann n4 ergibt.

***/MOD** (n1 n2 n3 -- n4 n5)
Berechnung von n4 wie mit */, jedoch zusätzliche Übergabe des Divisionsrestes n5.

ˆ Operationen mit doppelt genauen Zahlen

D+ (wd1 wd2 -- wd3)
wd3 ist die Summe von wd1 und wd2.

DNEGATE (d1 -- d2)
d2 ist die Differenz 0 minus d1.

UM* (u1 u2 -- ud)
ud ist das doppelt genaue Produkt von u1 und u2, alle Zahlen in vorzeichenloser Deutung.
Achtung! FIG und FORTH-79: Wortname: U*

UM/MOD (ud u1 -- u2 u3)
u2 ist der Rest und u3 der Quotient aus ud/u1, alle Zahlen in vorzeichenloser Deutung.
Achtung! FIG und FORTH-79: Wortname: U/MOD
Im FIG-Standard ist der Quotient doppelt genau: (ud u1 -- u2 ud2)

Weitere Wörter des Erweiterungssatzes zur Verarbeitung doppelt genauer Zahlen:

D- (wd1 wd2 -- wd3)
wd3: Differenz wd1 minus wd2.

D2/ (d1 -- d2)
d2 ist der Quotient d1/2.

DABS (d -- ud)
ud ist der Absolutbetrag von d.

‰ Logische Operationen

AND (16b1 16b2 -- 16b3)

16b3 ist die bitweise logische AND-Verknüpfung von 16b1 und 16b2 (UND).

OR (16b1 16b2 -- 16b3)
16b3 ist die bitweise logische OR-Verknüpfung von 16b1 und 16b2 (ODER).

XOR (16b1 16b2 -- 16b3)
16b3 ist die bitweise logische XOR-Verknüpfung von 16b1 und 16b2 (ANTIVALENZ).

NOT (16b1 -- 16b2)
16b2 ist die bitweise logische Negation von 16b1 (Einserkomplement).
Achtung! Im FIG-Standard nicht enthalten, in FORTH-79: Negation des Wahrheitswertes eines Flags. In FORTH-83 erfüllt NOT diese Funktion auch, wenn wahre Flags durchgängig mit der Zahl -1 dargestellt werden.

Š

FORTH-Wort	Stackinhalt
Y" (n1--n2 n3)	150
DUP	150 150
1000	150 150 1000
914	150 150 1000 914
*/	150 164
SWAP	164 150
10000	164 150 10000
254	164 150 10000 254
*/	164 5905

Teil 4, JU+TE 4/1990, S. 303-305

Abb.11 enthält einen Überblick der Standard-Stackmanipulationswörter. **OVER** (über) und **PICK** (auswählen) lassen sich als Varianten von DUP betrachten. OVER kopiert den zweiten Stapeleintrag und PICK einen beliebigen, dessen Tiefe vorher auf TOS stehen muß. Dabei entspricht 0 PICK dem Wort DUP und 1 PICK in gleicher Weise OVER.

Ebenso lassen sich **ROT** (rotieren) und **ROLL** (rollen) als Ableitungen von SWAP deuten. ROT holt den drittobersten Eintrag an die Spitze (TOS), während die beiden darüber liegenden jeweils eine Etage nach unten rutschen. Mit ROLL lassen sich beliebig lange Tauschringe realisieren. 1 ROLL entspricht SWAP und 2 ROLL entspricht ROT. Die Zahl n+ bei PICK und ROLL gibt an, wieviel Einträge über dem ausgewählten im Stapel liegen. Bei zu geringer Stapeltiefe führen die Tauschwörter SWAP, ROT und ROLL sowie die Vervielfältigungen DUP, OVER und PICK zur Fehlermeldung STACK EMPTY (MSG 1).

DROP (aufgeben) entfernt einfach den obersten Stapeleintrag. Dieses Wort benutzen wir zum Beseitigen überflüssiger Zahlen, die ausgangs einer Ausführung keinen Sinn mehr haben, um den Stack sauber zu halten. **?DUP** schließlich kopiert den obersten Eintrag genau dann, wenn er ungleich Null ist. Für das Ausprobieren der Stackmanipulationswörter empfiehlt es sich, ein paar Zahlen einzugeben und deren Position im Stack vor und nach Ausführen einer solchen Operation mit S. anzuzeigen.

4.2. Der Zugriff zum Return-Stack

Wenn eine größere Anzahl von Werten in recht ungeordneter Folge zu verarbeiten ist, wünscht man sich vielleicht einen zweiten Stapelspeicher.

Wie wir wissen, gibt es ihn ja auch: den Return-Stack. Er ist auch für das Ablegen von Daten geeignet, wenn das Verwalten der Unterprogramm-Rücksprünge nicht gestört wird. Alle Daten, die im Return-Stack abgelegt werden, sind daher vor Ende der betreffenden Struktur (Wortdefinition, Schleife) wieder abzuholen. Außer R@ lassen sich Return-Stack-Zugriffe nicht im Dialog, sondern nur innerhalb von Doppelpunktdefinitionen ausführen. Solange gesichert bleibt, daß alle Rückkehradressen bei Bedarf zur Verfügung stehen, kann der Return-Stack als Zwischenspeicher, nicht jedoch für die Datenübergabe von einem Wort zum anderen verwendet werden.
Günstig wirkt sich dieses Angebot z. B. auf das Berechnen von Polynomen aus. Als Beispiel dient eins dritter Ordnung:
$p = ax^3 + bx^2 + cx + d$
Ein günstiger Algorithmus wird nach Umstellen erkennbar:
p=((ax+b)x+c)x+d
Für den Stackinhalt
(d c b a x -- p) realisiert das Wort p3 die nötige Berechnung:

```
: P3 DUP
  >R * + R@ * + R>
  * + ;
```

Die unabhängige Variable x wird im Return-Stack aufbewahrt, während im Datenstack die Rechnung stattfindet. Vor Definitionsende hebt R> die durch >R hervorgerufene Veränderung des Return-Stack-Inhalts wieder auf. **Abb.12** faßt die Standardwörter des Return-Stack-Zugriffs zusammen.

4.3. Die doppelt genauen Einträge

Das Vervielfältigen und Tauschen doppelt genauer Stapeleinträge gehört nicht zum FORTH-83-Standardwortschatz. Die Standarderweiterung zur Verarbeitung doppelt genauer Zahlen enthält die in **Abb.13** dargestellten Wörter. Zum FIG-Standard gehört davon nur das Wort 2DUP, sonst sind diese Operationen auch bei den älteren Versionen an 32-bit-Erweiterungen gebunden.
Alle in Abb. 13 zusammengefaßten Wörter lassen sich leicht auf Grundlage der in Abb. 14 dargestellten definieren. Wie? Das sollten wir eigenständig herausfinden, um die Sicherheit beim Handhaben des Stacks zu vertiefen!

5. Verzweigungen und Schleifenstrukturen

Unsere Beispiele haben die Aufgaben bisher mit aneinandergereihten FORTH-Wörtern, die zu entsprechenden Ausführungsfolgen führen, gelöst. Solche „Geradeaus"-Programme reichen in den meisten Fällen nicht aus. Häufig müssen verschiedene Gegebenheiten auch mit unterschiedlichen Programmen behandelt und einige Ausführungsfolgen mehrmals wiederholt werden. FORTH kennt kein GOTO, besitzt aber andere, der strukturierten Programmiermethode entsprechende Wörter zur Programmsteuerung.

5.1. Die IF-ELSE-THEN-Verzweigung

Das bedingte Ausführen von Programmteilen wird wie in anderen Sprachen mit dem Wort **IF** (wenn) eingeleitet. Es erwartet ein Flag auf TOS, das mit Null (nein, falsch) oder einem beliebigen anderen Wert (ja, wahr) belegt ist. Diese Zahl läßt sich gut als Prüfergebnis der Verzweigungsbedingung deuten. Es muß vor dem Ausführen von IF berechnet werden. Wie das Flag in den Stack gelangt, ist dabei völlig gleichgültig. Gut eignen sich die in **Abb.14** zusammengefaßten Standard-Vergleichsoperationen und die in **Abb.15** notierten des 32-bit-Erweiterungssatzes. Der weitere Aufbau der Verzweigung beginnt mit dem Ja-Zweig. Das ist die Wortfolge, die bei wahrem Flag ausgeführt werden soll. Danach dürfen sich das Wort **ELSE** (andernfalls) und eine Wortfolge zur Ausführung bei falschem Flag (Nein-Zweig) anschließen. Das Wort **THEN** (dann) schließt in jedem Fall die Verzweigung ab. Was darauf folgt, wird unabhängig von der IF-Funktion (also unbedingt) ausgeführt. **Abb. 16** faßt die drei Verzweigungswörter zusammen. Falls nicht benötigt, können der Ja-Zweig oder ELSE und der Nein-Zweig auch entfallen.
Die IF-ELSE-THEN-Konstruktion darf meist nur innerhalb von Doppelpunktdefinitionen verwendet werden. Zum Ausprobieren der Vergleichsoperationen eignet sich ein wie folgt definiertes Wort:

```
: F? IF ." JA" ELSE
  ." NEIN" THEN ;
```

Im JA- und im NEIN-Zweig steht jeweils das Wort ."., das ausschließlich des folgenden Leerzeichens (Trennzeichen) den anschließenden Text bis zum nächsten Anführungszeichen auf den Bildschirm ausgibt. Wir nutzen das, um den durchlaufenen Zweig zu kennzeichnen. Das sieht z. B. so aus:

```
4 5 > F? (ENTER)
NEIN OK
```

Beim Ausprobieren der Vergleichsoperationen kann mit .S (oder STACK beim JU+TE-FORTH) zwischendurch die Stackbelegung kontrolliert werden. Auch das Wort - (minus) eignet sich zum Vergleichen. Welche Funktion es dabei erfüllt, bekommt ihr sicher selbst heraus.
Verzweigungsstrukturen lassen sich auch verschachteln. Dazu geben wir das Beispiel PLATZ, das die Plazierung nach einem sportlichen Wettkampf auswertet.

```
: MED DUP 1 = IF
        ." GOLD" THEN
      DUP 2 = IF
        ." SILBER" THEN
      DUP 3 = IF
        ." BRONZE" THEN ;

: PLATZ DUP 3 >
      IF . ." PLATZ"
            ELSE MED
            DROP
      THEN CR
            ." GRATULIERE!" ;
```

Das von PLATZ innerhalb einer Verzweigungsstruktur gerufene Wort MED enthält selbst mehrmals IF und THEN. Die Vordefinition MED ist nicht zwingend

und könnte auch in PLATZ direkt aufgenommen werden. Das sprengt bei diesem Beispiel jedoch den Eingabepuffer, der für 80 Zeichen ausgelegt ist. Die Stilvorschriften gestatten ohnehin nur eine Verzweigungs- oder Schleifenstruktur je Wortdefinition. Deshalb müßte eigentlich auch MED noch aufgeteilt werden.
Die Gratulation in unserem Beispiel hat volkssportlichen Charakter. Sollen nur Medaillen honoriert werden, muß das Wort THEN in der PLATZ-Definition ganz an das Ende direkt vor das Semikolon rutschen. **CR** (carriage return) löst eine Zeilenschaltung, also das Fortsetzen der Anzeige in der nächsten Zeile aus. Das Wort DUP kopiert jeweils die Platzziffer, um sie trotz der zerstörenden Entnahme durch den folgenden Vergleich im Stack zu bewahren. Das Säubern des Stacks von diesem Eintrag erledigen der Punkt im Ja-Zweig des Wortes Platz oder DROP im Nein-Zweig. Insgesamt gilt daher der Stackkommentar (n--).

5.2. Die DO-LOOP-Schleife

Die Wörter **DO** (führe aus) und **LOOP** (Schleife) werden verwendet, um dazwischen stehende Wörter mehrmals auszuführen. Der Steuerung dient ein im Return-Stack verwalteter Schleifenindex, dessen Anfangs- und Endwert DO im Datenstack erwartet (den Anfangswert als TOS).

Dr. Helmut Hoyer (wird fortgesetzt)

11 Standard-Stackmanipulationen

DEPTH (-- +n)
+n ist die Anzahl der Stapeleinträge außer +n.
Achtung! In FIG und FORTH-79 nicht enthalten!

DUP (16b --16b 16b)
Duplizieren des TOS

OVER (16b1 16b2 -- 16b1 16b2 16b1)
Duplizieren des zweitobersten Stapeleintrags.

PICK (+n -- 16b)
16b ist die Kopie des (+n + 1)ten Stapeleintrags außer +n.
Achtung! In FIG und FORTH-79 nicht enthalten!

SWAP (16b1 16b2 -- 16b2 16b1)
Vertauschen der beiden obersten Stapeleinträge.

ROT (16b1 16b2 16b3 --16b2 16b3 16b1)
Rotieren der drei obersten Stapeleinträge, so daß der drittoberste an die Spitze gelangt.

ROLL (+n --)
Rotieren der +n + 1 obersten Stapeleinträge außer + n.
Achtung! In FIG und FORTH-79 nicht enthalten!

DROP (16b --)
Entfernen von 16b aus dem Stack.

?DUP (16b -- 16b 16b) oder (0 --0) .
Duplizieren des TOS, wenn er ungleich Null ist.

12 Return-Stack-Zugriffe

>R (16b --)
Transport des TOS zum Return-Stack.

R> (-- 16b)
Transport des obersten Return-Stack-Eintrags zum Datenstack.

R@ (-- 16b)
16b ist die Kopie des obersten Return-Stack-Eintrags. Der Return-Stack wird nicht verändert.
Achtung: FIG: Wortname R, FORTH-79: nicht vorhanden, aber das Wort I hat die gleiche Funktion und kann statt R@ verwendet werden.

13 32-bit-Stackmanipulationen des Erweiterungswortschatzes

2DROP (32b --)
Entfernen von 32b aus dem Stack.

2DUP (32b -- 32b 32b) Duplizieren von 32b auf dem Stack.

2OVER (32b1 32b2 -- 32b1 32b2 32b1)
Duplizieren des zweitobersten doppelt genauen Stapeleintrages.

2SWAP (32b 32b2 -- 32b2 32b1)
Vertauschen der beiden obersten doppelt genauen Stapeleinträge.

2ROT (32b1 32b2 32b3 -- 32b2 32b3 32b1)
Rotieren der drei obersten doppelt genauen Stapeleinträge, so daß der drittoberste an die Spitze gelangt.

14 Vergleichsoperationen

< (n1 n2 -- ?)
Flag ? ist wahr, wenn n1 < n2.

> (n1 n2 -- ?)
Flag ? ist wahr, wenn n1 > n2.

= (wt w2 -- ?)
Flag ? ist wahr, wenn w1 = w2.

0< (n--?)
Flag ? ist wahr, wenn n < 0.

0> (n -- ?)
Flag ? ist wahr, wenn n > 0.

0= (w -- ?)
Flag ? ist wahr, wenn w = 0. Dieses Wort eignet sich zum Negieren eines Flags.

U< (u1 u2 -- ?)

Flag ? ist wahr, wenn u1 < u2.

D< (d1 d2 -- ?)
Flag ? ist wahr, wenn d1 < d2.
Achtung! In FIG und FORTH-79 nicht enthalten!

15 32-bit-Vergleichsoperationen des Erweiterungswortschatzes

D= (wd1 wd2 -- ?)
Flag ? ist wahr, wenn wd1 = wd2.

D0= (wd -- ?)
Flag 7 ist wahr, wenn wd = 0.

DU< (ud1 ud2 -- 7)
Flag ? ist wahr, wenn ud1 < ud2.

16 Verzweigungswörter

IF (? --)
Beginn einer Verzweigung. Ist ? wahr belegt, werden die folgenden Wörter bis ELSE bzw. THEN ausgeführt. Ist ? falsch belegt, kommen die Wörter zwischen ELSE und THEN zur Ausführung.

ELSE (--)
Dieses Wort trennt in einer Verzweigung die davor stehende Wortfolge für den Ja-Zweig (wahr) von der danach stehenden für den Nein-Zweig (falsch).

THEN (--)
Ende einer Verzweigung. Die folgenden Wörter werden unabhängig von der davor stehenden Verzweigung ausgeführt.
Achtung! FIG: Wortname: ENDIF

Verzweigungsstruktur:
IF Ja-Zweig ELSE Nein-Zweig THEN

17 DO-LOOP-Schleife

DO (w1 w2 --)
Eröffnung einer Schleife mit Indexlauf von Anfangswert w2 bis Endwert w1.
Achtung! FIG und FORTH-79 verwenden nur vorzeichenbehaftete Indexlaufgrenzen: (n1 n2 --)

LEAVE (--)
Unbedingtes Verlassen einer DO-LOOP-Schleife mit Beseitigen der Parameter vom Return-Stack.
Achtung! FIG und FORTH-79: Verlassen der Schleife erst nach Ausführung der nächsten LOOP bzw. +LOOP.

LOOP (--)
Inkrementieren des Schleifenindex und Verlassen der Schleife mit Beseitigen der Parameter vom Return-Stack, wenn Endwert erreicht. Sonst Fortsetzung hinter DO.

+LOOP (n --)
Addieren von n zum Schleifenindex und Verlassen der Schleife mit Beseitigen der Parameter vom Return-Stack, wenn Endwert erreicht. Sonst Fortsetzung hinter DO.

I (-- w)
w ist die Kopie des Schleifenindex.

J (--w)
w ist die Kopie des Index der nächstäußeren Schleife.
Achtung! Im FIG nicht enthalten!

Teil 5, JU+TE 5/1990, S. 395-397

(Fortsetzung zu 5.1.: Die IF-ELSE-THEN-Verzweigungen)

Die Erhöhung um einen beliebigen, auf TOS erwarteten Wert mit dem gleichen Test erledigt +LOOP. Dieses Wort ist z. B. beim Anzeigen des Stackinhalts wichtig, wo der Index (Steckadresse) von Durchlauf zu Durchlauf um 2 zu erhöhen ist. FORTH gestattet das Verschachteln von DO-LOOP-Schleifen untereinander und mit IF-ELSE-THEN-Strukturen. Zwischen DO und LOOP (bzw. +LOOP) dürfen auch die Wörter **LEAVE** (Verlassen), **I** und **J** stehen. (vgl. **Abb.17**). LEAVE beendet die Schleife unabhängig vom Index. Es kommt meist innerhalb von IFELSE-THEN-Verzweigungen vor. I kopiert den aktuellen Schleifenindex auf den Stack, das entspricht dem Wort R@. J gestattet in gleicher Weise den Zugriff auf den Index der nächstäußeren Schleife, sofern sie überhaupt existiert. Zum Ausprobieren eignet sich zunächst ein einfaches Beispiel:

```
: TEST 0 DO I .
  LOOP ." FERTIG!" :
```

Es erwartet eine Zahl auf dem Stack, die DO als Endwert verwendet. Innerhalb der Schleife wird nur der Index angezeigt. Seinen Lauf können wir uns mit verschiedenen Endwerten anzeigen lassen. Das sieht z. B. so aus:

```
3 TEST (ENTER)
0 1 2 FERTIG! OK
```

Das nächste Beispiel läßt den Index stets um 3 anwachsen und bricht ab, wenn er zwischendurch gleich 50 ist. Hier wird ein Anfangswert im Steck erwartet und bis Endwert 100 gearbeitet:

```
: TEST2 100 SWAP
  DO I . I 50 = IF
  LEAVE THEN 3
  +LOOP ;
```

Beim Ausprobieren wird deutlich, daß DO-LOOP allgemein mit vorher festgelegtem Indexlauf arbeitet. aber auch vorzeitigen Abbruch gestattet. FORTH bietet darüber hinaus Möglichkeiten, die Schleifenorganisation von vornherein datenabhängig zu gestatten.

5.3. Die BEGIN-UNTIL-Schleife

Schleifen, die mit **BEGIN** (beginne) eröffnet werden, verbinden den Abbruch mit einer Flagauswertung, ohne eine Struktur wie IF LEAVE THEN zu erfordern. Sie brauchen auch keinen Zählindex. **UNTIL** (bis) steht stets am Schleifenende. Dieses Wort entnimmt dem Stack eine Zahl und beendet die Schleife, wenn sie ungleich Null ist. Solange jedoch eine Null gefunden wird, erfolgt die Fortsetzung mit dem ersten Wort nach BEGIN. Auch das läßt sich ausprobieren:

```
: TEST3 BEGIN KEY
  ." TASTE" DUP .
  CR 13 = UNTIL
  ." ENTER" ;
```

Diese Definition verwendet das Wort **KEY** (Taste), das eine Tasteneingabe fordert und den zugehörigen Code (ASCII) auf den Stack legt. TEST3 zeigt ihn als Zahl gedeutet an und beginnt von vorn, wenn es sich dabei nicht um die ENTER-Taste handelt. UNTIL kann die Zahl (Flag) auf beliebige Weise erhalten. Wie bei IF ist eine Vergleichsoperation nicht zwingend erforderlich Mit 0 UNTIL läßt sich z B. eine Endlos-Schleife abschließen.

5.4. Die BEGIN-WHILE-REPEAT-Schleife

WHILE (solange) und **REPEAT** (wiederhole) trennen die in UNTIL zusammengefaßten Elemente Abbruchtest und Schleifenende-Markierung. Damit gestattet FORTH, die Ausführung einer Schleife an beliebiger Stelle abzubrechen. Das Anordnen von WHILE gleich hinter BEGIN macht es möglich, bei nicht erfüllter Bedingung die danach notierte Schleife nicht ein einziges Mal zu durchlaufen. Die Wörter zwischen BEGIN und WHILE werden zunächst unbedingt ausgeführt. WHILE holt eine Zahl (Flag) vom Stack und beendet die Schleife, wenn sie gleich Null ist. In diesem Fall wird mit dem ersten Wort nach REPEAT fortgesetzt. Andernfalls folgen die Wörter zwischen WHILE und REPEAT, anschließend wieder der Schleifenanfang nach BEGIN. WHILE verzweigt also, während REPEAT einen unbedingten Sprung zu BEGIN darstellt. Das gestattet, das letzte Beispiel so zu modifizieren, daß der Code der ENTER-Taste nicht mehr zur Anzeige kommt:

```
: TEST4 BEGIN KEY
  DUP 13 - WHILE .
  CR REPEAT DROP
  ." ENTER" ;
```

Abb.18 faßt die Strukturwörter zur Daten-abhängigen Konstruktion von Programmschleifen zusammen.

5.5. Die Wörter mit impliziten Entscheidungen

Ähnlich den Wörtern UNTIL und WHILE, deren Wirkung von einem Flag abhängt, enthalten auch andere Wörter eine IF-THEN-Struktur im Inneren. Eines haben wir bereits kennengelernt: ?DUP (Abb. 11). Zum FORTH-Standard gehören außerdem zwei Auswahloperationen (**Abb. 19**). **MAX** entfernt den TOS, wenn er kleiner ist als der darunter liegende Wert. Andernfalls bleibt der TOS erhalten, während der darunter liegende Wert verschwindet. Solch eine Operation läßt sich in FORTH definieren, wobei die eingeschlossene Entscheidung deutlich wird:

```
: MAX OVER OVER <
  IF SWAP THEN DROP ;
```

Praktisch sind die Auswahloperationen jedoch meist nicht in FORTH, sondern mit einfachen Maschinenprogrammen realisiert. Das Wort **MIN** funktioniert auf gleiche Weise, natürlich mit entgegengesetzter Vergleichsoperation. FORTH-83-Systeme mit Standarderweiterung zum Verarbeiten doppelt genauer Zahlen enthalten auch die in **Abb. 20** aufgeführten Wörter, die entsprechend zwischen 32-bit-Einträgen auswählen. Ein wichtiges Anwendungsgebiet findet sich im Begrenzen von Zahlenbereichen. Das Wort LIMIT hält als Beispiel den TOS zwischen 0 und 7:

```
: LIMIT 0 MAX 7 MIN ;
```

Liegt der TOS außerhalb des Intervalls, wird er durch die nächstliegende Grenze (0 oder 7) ersetzt.

6. Speicherzugriffe

Da der Stack meist im Speicher angesiedelt ist, führen eigentlich alle FORTH-Wörter irgendwie Speicherzugriffe aus. Sie werden vom System verwaltet. Die Wörter, denen wir uns im folgenden zuwenden, übertragen die Verantwortung über die Speicherverwaltung dem Programmierer. Er kann dazu Reservierungsdienste des FORTH-Systems wie CREATE oder VARIABLE (vgl. Abb.4) in

Anspruch nehmen oder völlig agieren.

6.1. Der Zugriff auf einzelne Speicherzellen

In den Abschnitten 2.2. und 2.3. wurden die beiden wichtigsten Wörter zum Speicherzugriff bereits eingeführt. @ (fetch = holen) und ! (store = speichern) erwarten die 16-bit-Adresse der anzusprechenden Speicherzelle als TOS. @ ersetzt den TOS mit dem Inhalt dieser Zelle, ! dagegen speichert dort den zweitobersten Stapeleintrag (**Abb.21**). Beide Wörter arbeiten mit 16-bit-Zahlen und beziehen sich daher stets auf zwei Speicherzellen (je acht bit). Das Wort **+!** eignet sich zum Addieren von Stapeleinträgen zu Speicherinhalten. Zum Erhöhen des Wertes der Variablen OTTO um drei eignet sich z. B. die folgende Eingabe:
3 OTTO +! (ENTER)
Soll ein Variablenwert zum anderen addiert werden, muß ein Operand zunächst in den Stack kopiert werden. Die BASIC-Anweisung
LET OTTO = OTTO + UDO
heißt in FORTH geschrieben:
UDO @ OTTO +! (ENTER).
Die Wörter @, ! und +! beziehen sich auf das 16-bit-Zahlenformat. Dadurch eignen sie sich gut zum Verwalten normaler Stapeleinträge im Speicher, z. B. mit Hilfe von Variablendefinitionen. Für den byteweisen Zugriff enthält der FORTH-83-Standard die Wörter **C!** und **C@**, die wie ! und @ wirken, aber nur eine 8-bit-Speicherzelle betreffen. Dem entsprechend enthält der zugehörige Stapeleintrag auch nur acht gültige Binärstellen. Das gestattet auf einfache Weise die Verarbeitung von variablen Zeichenketten. Das Beispiel TEXT speichert vier mit der Tastenabfrage KEY eingegebene Zeichen ab der Adresse %FC80. Damit diese Adresse im üblichen hexadezimalen System eingegeben werden kann, stellen wir zunächst die Zahlenbasis auf 16:
16 BASE ! (ENTER).
Nun versteht das FORTH-System die folgende Definition:

```
: TEXT FC80 4 0 DO KEY
  OVER C! 1+
  LOOP DROP ;
```

Wenn wir nun die Zahlenbasis wieder auf den Standardwert stellen, bleibt die Definition TEXT dennoch korrekt gespeichert:
DECIMAL (ENTER)
Beim Ausprobieren werden vier Tasteneingaben erwartet, auf die jedoch kein unmittelbares Echo folgt. Erst nach der vierten Betätigung erscheint das gewohnte **OK**. Ob alles geklappt hat, überprüft das Wort LIES, vor dessen Definition die Basis wieder auf 16 zu stellen ist.

```
: LIES FC80 4 0 DO
  DUP C@ . 1+ LOOP
  DROP ;
```

Steuerzeichen werden von TEXT abgespeichert, aber nicht ausgeführt. LIES bringt den eingegebenen kurzen Text als ASCII-Zahlenfolge auf den Bildschirm. TEXT und LIES verwenden einen vom FORTH-System unabhängigen Speicherbereich und geben damit ein Beispiel für das Erzeugen von Speicheradressen ohne Vermittlung durch Wörter wie VARIABLE.

6.2. Das Verwalten von Speicherfeldern

Solch eine Gruppe zusammenhängender Speicherzellen wie der für TEXT und LIES verwendete Bereich von %FC80 bis %FC83 wird häufig als Feld bezeichnet. FORTH gestattet auch das Verwalten von Speicherfeldern im Wörterbuch. Den Wortkopf erzeugt dabei gewöhnlich die CREATE-Definition (siehe JU+TE 2/90, S. 146f). Zum Erzeugen eines Pufferspeichers mit 80 byte Länge eignen sich die Eingaben:
CREATE PUFFER
(ENTER) **OK**
80 ALLOT (ENTER) **OK**
Das Wort ALLOT (zumessen) reserviert Platz im Wörterbuch. Die Anzahl der benötigten Speicherzellen muß vorher in den Stack eingetragen werden (**Abb. 22**). Unser Puffer kann nun bis zu 80 Zeichen oder 40 Zahlenwerte beherbergen. Beim Speichern von Zeichenketten unbestimmter Länge sollte die erste Speicherzelle des Puffers für das Zählbyte (count) freigehalten werden. Das sichert die Kompatibilität (Paßfähigkeit) zu den Standard-Ein- und -Ausgabewertem (z. B. TYPE = Zeichenketten-Anzeige).

Beim Speichern von Vektoren (Gruppen zusammengehöriger Zahlen) ist zu beachten, daß zwei Byte je Standardeintrag bzw. vier byte je doppelt genauem Ertrag benötigt werden. In unserem Beispiel liefert das Wort PUFFER die Adresse der ersten Zahl.

Das dritte Element eines Vektors von Standardzahlen hat die Adresse PUFFER+4 Es läßt sich mit
PUFFER 4 + @
(ENTER)
lesen. Für das Vereinbaren eines Vektors mit einer bestimmten Anfangsbelegung empfiehlt, sich das Wort (Komma). Es reserviert zwei byte im Wörterbuch und trägt dort den TOS ein Zum Speichern eines Vektors mit der Belegung (3, 4, 50) eignet sich die Eingabefolge
CREATE VEKTOR
(ENTER) **OK**
3 , 4 , 50
(ENTER) **OK**.

Das Wort ALLOT ist in diesem Fall nicht erforderlich Dessen Funktion erledigt das Komma 2-byte-weise mit. Wichtig sind die vor und nach jedem Komma einzugebenden Leerzeichen. Das Anzeigen eines solchen dreidimensionalen Vektors gehört nicht zum Standard, läßt sich aber leicht definieren.

```
: V. DUP 6 +
  SWAP DO I @ . ." , "
  2 +LOOP ;
```

Dieses Wort erwartet die Vektoradresse auf dem Stack und ergibt z.B. die folgende Anzeige:
VEKTOR V. (ENTER)
3 , 4 , 50 , OK
Nicht immer läßt sich das Adressieren mit dem Index einer DO-LOOP-Schleife organisieren. Zum Addieren zweier Vektoren kann die zweite Adresse im Steck verwaltet werden:

```
: V+ SWAP DUP 6
  + SWAP DO DUP
  @ I +! 2+ 2
  +LOOP DROP ;
```

Dieses Wort nimmt zwei Vektoradressen vom Stack, wobei als TOS die Adresse des zweiten (nicht veränderten) Operanden erwartet wird. Bei komplizierten Rechnungen ist das Verwalten der Adressen im Stack umständlicher, hier empfiehlt sich das Einführen von Variablen als Merkhilfe.

18 BEGIN-Schleifen

BEGIN (--)
Eröffnung einer Programmschleife der Struktur BEGIN - UNTIL oder BEGIN - WHILE - REPEAT

UNTIL (? --)
Ist Flag ? wahr, erfolgt die Fortsetzung mit dem folgenden Wort, sonst Rücksprung zu BEGIN. Schleifenabschluß.

WHILE (? --)
Ist Flag ? wahr. erfolgt die Fortsetzung mit dem folgenden Wort, sonst mit dem ersten Wort hinter REPEAT.

REPEAT (--)
Programmfortsetzung mit dem ersten Wort nach BEGIN Schleifenabschluß.

19 Auswahloperationen

MAX (n1 n2 -- n3)
n3 ist die größere der beiden Zahlen n1 und n2.

MIN (n1 n2 -- n3)
n3 ist die kleinere der beiden Zahlen n1 und n2.

20 Auswahloperationen (32-bit-Erweiterung)

DMAX (d1 d2 -- d3)
d3 ist die größere der beiden Zahlen d1 und d2-

DMIN (d1 d2 -- d3)
d3 ist die kleinere der beiden Zahlen d1 und d2

22 Speicherplatz-Reservierung

ALLOT (w --)
Reservieren von w Bytes im Wörterbuch.

, (16b --)
Reservieren von zwei Bytes im Wörterbuch und Speichern von 16b an diesem Ort.

21 Speicherzugriffe

! (16b adr --)
16b wird in den Speicherzellen adr und adr+1 gespeichert

@ (adr -- 16b)
16b ist der Inhalt der Speicherzellen adr und adr+1.

+! (w adr --)
w wird zum Inhalt der Speicherzellen adr und adr+1 addiert.

C! (8b adr --)
8b wird in der Speicherzelle adr gespeichert.

C@ (adr – 8b)
8b ist der Inhalt der Speicherzelle adr.

Wörter des 32-bit-Erweiterungssatzes

2! (32b adr --)
32b wird in den Speicherzellen adr bis adr+3 gespeichert.

2@ (adr -- 32b)
32b ist der Inhalt der Speicherzellen adr bis adr+3

Teil 6, JU+TE 6/1990, S. 80-81

Fortsetzung. zu Folge 5 in JU+TE 5/1990, Abschnitt 6.2.

Matritzen lassen sich wie Vektoren speichern, indem z. B. deren Zeilen einfach aneinan-

dergereiht abgelegt werden. Die Adresse jedes Elements ist dann die Spaltenanzahl mal Spaltenindex plus Zeilenindex mal zwei plus Leitadresse (nach Abruf des Namens als TOS verfügbar):
(m*j+i)*2+adr.
Das Programmieren der ganzen Vektor- und Matritzenalgebra gelingt anhand der mathematischen Vorschriften, stellt aber ein sehr weites Gebiet dar. Wir beschränken uns wieder auf die Standardwörter.

6.3. Das Verschieben von Speicherbereichen

Zum Laden eines Vektors mit dem Inhalt eines andern eignen sich die Wörter **CMOVE** und **CMOVE>** (**Abb. 23**, move: bewegen, C steht für Byteweise). Die drei benötigten Parameter sind die Quelladressen adr1, die Zieladresse adr2 und die Feldgröße (Byteanzahl) u. Der Unterschied zwischen beiden Wörtern ist von Bedeutung, wenn sich Ziel- und Quellbereich überlappen. In diesem Fall sichert CMOVE den Datenerhalt bei Transporten in Richtung niederer und CMOVE> in Richtung höherer Adressen. Das dritte Wort in Abb. 23 eignet sich zum Füllen (**FILL**) aller Speicherzellen eines Bereiches mit der gleichen 8-bit-Kombination. Beim Ausprobieren dieser Wörter müssen wir darauf achten, daß das Wörterbuch nicht überschrieben wird.

23 Speicher-Transport-Operationen

CMOVE (adr1 adr 2 u --)
Transport von u Bytes ab Adresse adrl zur Adresse adr2 beginnend mit dem ersten Byte.

CMOVE> (adr1 adr2 u --) Transport von u Bytes ab Adresse adrl zur Adresse adr2 beginnend mit dem letzten Byte.

FILL (adr u 8b --)
Laden von u Bytes ab Speicheradresse adr mit 8b.

7. Ein- und Ausgabe

7.1. Die Zeichenausgabe

Das Ausgabemedium für FORTH ist der Bildschirm. Hier können die durch die Kodiervorschrift ASCII festgelegten Zeichen ausgegeben werden. Der FORTH-83-Standard enthält für diesen Zweck die in **Abb. 24** dargestellten Wörter. **EMIT** (aussenden) bringt den TOS als ASCII gedeutet auf den Bildschirm, bei jeden Aufruf ein Zeichen. Spezielle Codes werden mit **CR** (Zeilenschaltung) und **SPACE** (Leerzeichen) ausgegeben. Space könnte z. B. wie folgt definiert sein
: SPACE 32 EMIT ;
32 ist der dezimale Wert von %20, dem ASCII für das Leerzeichen. Die Wörter CR und ." haben wir bereits im fünften Kapitel kennengelernt. Hier noch ein Beispiel für die Zeichenkettenausgabe (.") zum Ausprobieren:
: ALF ." MELMAC LAESST GRUESSEN!" ;
ALF (ENTER)

MELMAC LAESST GRUESSEN! OK
Mit der Formulierung
10 0 DO ALF CR LOOP
läßt sich zeigen, daß sich auch die Zeichenkettenausgabe ganz normal in das Wörterbuch einreiht. Das Wort **TYPE** (tippen) gestattet das Anzeigen in einem Puffer zusammengestellter Texte. Mit
16 BASE ! (ENTER)
FC80 4 TYPE (ENTER)
kann die mit TEXT erfaßte Eingabe zeichenorientiert (statt als Codes) angezeigt werden. **-TRAILING** kürzt Zeichenketten um eventuell nachgestellte Leerzeichen mit zu TYPE passender Parameterstruktur (trail: schleppen). Dieses Wort befreit auf diese Weise von unnützem Anhang.

24 Zeichenausgabe

EMIT (16b --)
Ausgabe von 16b als ASCII auf dem Bildschirm. Meist werden nur die sieben niederwertigen Dualstellen verwendet.

CR (--)
Ausgabe des Steuerzeichens Zeilenschaltung (carriage return und line feed).

SPACE (--)
Ausgabe eines Leerzeichens.

." (--)
Ausgabe aller folgenden Zeichen ausschließlich des abschließenden " und des ersten nach ." stehenden Leerzeichens auf dem Bildschirm.

TYPE (adr +n --)
Ab adr werden +n Zeichen aus dem Speicher gelesen und auf dem Bildschirm ausgegeben.

-TRAILING (adr +nr1 -- adr +n2)
Der Zeichenzähler +n1 der ab adr stehenden Zeichenkette wird soweit

verringert, daß nachgestellte Leerzeichen unterdrückt werden.

7.2. Die Zahlenausgabe

Zahlen stellt FORTH intern stets im dualen Format dar. Für die Ein- und Ausgabe wird mit entsprechenden Umrechnungen (Konvertierung) eine vom Programmierer wählbare Zahlenbasis (base) zugrundegelegt. Um diese zu speichern, enthält das FORTH-System die Variable BASE (**Abb. 25**). Sie kann mit **DECIMAL** auf die Standardbelegung, mit ! auf jede andere natürliche Zahl von 2 bis 57 eingestellt werden.

25 Zahlenausgabe

BASE (-- adr)
Adresse der Systemvariablen zum Speichern der Zahlenbasis für die Ein- und Ausgabe.

DECIMAL (--)
Speichern der Zahlenbasis 10 in BASE.

. (n --)
Ausgabe des TOS als Zahl bezüglich der in BASE gespeicherten Basis. Negative Werte kennzeichnet ein vorangestelltes Minuszeichen.

U. (u --)
Zahlenausgabe in vorzeichenloser Deutung.
Doppelt genaue Zahlenausgaben

D. (d --)
Ausgabe von d als vorzeichenbehaftete Zahl.

D.R (d +n --)
Ausgabe von d als vorzeichenbehaftete Zahl rechtsbündig in einem Feld von +n Zeichen.

Die Wörter **.** (dot) und **U.** zeigen den TOS bezüglich der gerade eingestellten Basis vorzeichenbehaftet bzw. vorzeichenlos gedeutet an. Bei 32-bit-Erweiterung wirkt **D.** entsprechend. Mit **D.R** können Ausgabeformat-gebundene Anzeigen mit rechtsbündiger Anordnung in einem Feld wählbarer Breite (+n) realisiert werden. Das gestattet das Darstellen gleichwertiger Stellen untereinander. Führende Nullen werden bei allen vier Zahlenausgabewörtern unterdrückt. Zahlenausgaben mit anderen Eigenschaften lassen sich zusätzlich definieren. Als Grundlage eignen sich die in **Abb. 26** zusammengestellten Wörter. Sie realisieren den mathematischen Konvertierungsalgorithmus, bei dem das fortgesetzte Dividieren der auszugebenden Zahl durch die Basis deren Ziffern, beginnend mit der niederwertigsten, aus dem Divisionsrest ergibt. Dabei entstehen die einzelnen Zeichen in umgekehrter Reihenfolge zur entsprechenden Bildschirm-Ausgabe. Deshalb werden die auszugebenden Zeichen in einen speziellen Stapelspeicher eingetragen. Er ist unabhängig von Daten- und Return-Stack. Die seine Adresse speichernde Variable HLD gehört beim FIG-Standard sogar zum Wortschatz: Das last-in-first-out-Prinzip dieser Form der Speicherverwaltung realisiert die erforderliche Umkehr der Ausgabe-Reihenfolge.

26 Ausgabekonvertierung

<# (--)
Initialisieren der Zahlenausgabe mit Eröffnen eines Stapelspeichers für die auszugebenden Zeichen.

(+d1 -- +d2)
Division von +d1 durch die in BASE gespeicherte Zahlenbasis. +d2 ist der Quotient, der Divisionsrest wird als ASCII in den Ausgabe-Stapelspeicher eingetragen.

#S (+d -- 0 0)
Zyklisches Ausführen von #, bis der doppelt genaue Quotient gleich Null ist.

SIGN (n --)
Eintragen des ASCII für - (minus) in den Ausgabe-Stapelspeicher, wenn n negativ ist.
Achtung! Bei FIG und FORTH-79 wird n aus drittoberster Stackposition entnommen, die beiden darüber liegenden Einträge bleiben erhalten (n d -- d).

HOLD (16b --)
Eintragen von 16b als ASCII in den Ausgabe-Stapelspeicher.

#> (32b -- adr +n)
Abschluß der Zeichenkette im Ausgabe-Stapelspeicher, entfernen von 32b vom Stack und Übergabe von Adresse adr und Länge +n für die Anzeige mit TYPE.

Als Beispiel betrachten wir eine in FORTH übliche Definition zur Anzeige von Geldbeträgen, die intern in Pfennigen als doppelt genaue Einträge gespeichert werden. Die Anzeige erfolgt mit Dezimalpunkt und nachgestelltem Minuszeichen:

```
: .MARK SWAP OVER DABS
  <# ROT SIGN # # 46
  HOLD #S #>TYPE
  ." M" ;
```

Dieses Wort nimmt einen doppelt genauen Eintrag vom Stack (d --). SWAP und OVER vergraben zunächst eine Kopie vom höheren Teil des Ausgabewertes an dritter Stackposition, um das Vorzeichen zu speichern. DABS bildet den Absolutbetrag, <# eröffnet den Zeichenspeicher. ROT und SIGN bringen ein eventuell

nötiges Minuszeichen ein. # # dient dem Eintragen der beiden Pfennig-Stellen, denen der Dezimalpunkt mit 46 HOLD folgt. #S fügt die Mark-Stellen an, bevor #> und TYPE die Anzeige dieser Zeichenkette veranlassen. Den Abschluß bildet die Ausgabe der Maßeinheit. Bei älteren FORTH-Standards wird ROT nicht benötigt. Auch im JU+TE-FORTH (vgl. JU+TE 1/1990) funktioniert SIGN entsprechend der FIG-Definition. Das Anpassen an den FORTH-83-Standard kann wie folgt nachgeholt werden:

```
: SIGN ROT ROT SIGN ;
```

Die danach einzugebende .MARK-Definition funktioniert dann auch auf dem JU+TE-Computer standardgerecht.

7.3. Die Tasteneingaben

Für das Eingeben eines Zeichens per Tastatur enthält FORTH-83 das Wort **KEY** (Taste, **Abb. 27**). Es erwartet eine Neubetätigung (statische Abfrage) und bringt den zugeordneten Zeichencode (ASCII) auf den Stack). Die dynamische Tastenabfrage KEY? wartet nicht auf den Bediener und legt ein Flag auf den Stack, das bei Nichtbetätigung gleich Null (falsch) ist. Dieses Wort ist üblich, gehört aber nicht zum Standard.

27 Eingabeoperationen

KEY (-- 16b)
Die niederen sieben Bit von 16b sind das ASCII der betätigten Taste.

EXPECT (adr +n --)
Eingabe von höchstens +n Zeichen und Abspeichern ab adr. Beim Erkennen des ENTER-Zeichens Rückkehr ohne Speichern des ENTER-Codes. Speichern eines zusätzlichen Leerzeichens. Anzeige aller Zeichen außer ENTER auf dem Bildschirm (Echo).

SPAN (-- adr)
Adresse der Variablen, die die Anzahl der zuletzt mit EXPECT eingegebenen Zeichen enthält.

CONVERT (+d1 adr1 -- +d2 adr2)
Eingabekonvertierung der Zeichenkette ab adr + 1 mit Startwert +d1 in das doppelt genaue Ergebnis +d2, indem bei jedem Zeichen der bisherige Wert mit der Zahlenbasis (BASE) multipliziert und der Wert des neuen Zeichens addiert werden, bis ein nicht konvertierbares Zeichen erreicht ist. Dessen Adresse gibt adr2 an.

Mit KEY können beliebige Eingaben aufgebaut werden. Das Abspeichern eingetasteter Zeichenketten erledigt **EXPECT** (erwarten). Anfangsadresse und maximale Zeichenzahl werden im Stack erwartet. Als einziges Steuerzeichen wirkt ENTER (oder RETURN). Es kennzeichnet den Abschluß. Andere Steuerzeichen werden als ASCII in die Zeichenkette aufgenommen, ohne dort zu wirken. Die Echo-Funktion auf dem Bildschirm kann das allerdings vortäuschen. Das zur erfaßten Zeichenkette gehörige Zählbyte (count) speichert EXPECT in der Variablen **SPAN** (Spannweite). Das gestattet das Erzeugen von FORTH-Zeichenketten (mit vorangestelltem Zählbyte) über Tasteneingabe z. B. in folgender Form:

```
: STRING OVER 1+ SWAP
  EXPECT SPAN @ SWAP ! ;
```

STRING erwartet wie EXPECT Adresse und maximale Zeichenzahl für die anzugebende Zeichenkette im Stack (adr +n --). Abgespeichert wird aber erst ab adr + 1. Auf adr gelangt nach Ausführen von EXPECT der Inhalt der Variablen SPAN, das Zählbyte.
Für Zahleneingaben sieht FORTH zunächst das Erfassen einer Zeichenkette vor, die nachträglich in den entsprechenden Zahlenwert konvertiert wird. Das dazu in den Standard aufgenommene Wort heißt **CONVERT** (umwandeln). Es erwartet an adr1 ein FORTH-Zeichenkette, die ausschließlich des vorangestellten Zählbytes anhand der Zahlenbasis BASE Zeichen für Zeichen in einen Zahlenwert gewandelt wird. Ausgangspunkt ist der doppelt genaue Startwert +d1, Ergebnis entsprechend +d2. Diese Parameterübergabe gestattet das Aneinanderreihen mehrerer CONVERT-Aufrufe. CONVERT bricht ab, sobald ein bei der aktuellen Zahlenbasis nicht als Ziffer deutbares Zeichen erreicht ist. Die Paßfähigkeit zu EXPECT wird bei FORTH-83 dadurch gesichert, daß EXPECT als letztes (statt ENTER) ein Leerzeichen in die Kette aufnimmt, das für CONVERT bei jeder Basis einen Abbruch bedeutet.
Während der Konvertierung kann das Berücksichtigen von Steuerzeichen eingefügt werden. Unser Beispiel enthält solche Funktionen nicht. Es ermittelt eine doppelt genaue positive Zahl (-- +d):

```
: INPUT PAD DUP 64
  STRING 0 0 ROT CONVERT
  DROP ;
```

Im JU+TE-FORTH ist dieses Wort auf der Basis von QUERY und NUMBER, die nicht zum

FORTH-83-Standard gehören, definiert (JU+TE 1/90, S. 70ff). **PAD** (Polster) liefert die Adresse eines bei FORTH-83 für verschiedene Zwecke benutzten Zwischenspeichers (Abb. 28). Auch andere Bereiche eignen sich zum zeitweiligen Aufbewahren ein- und auszugebener Zeichenketten.

28 Zeichenketten-Verarbeitung

HERE (-- adr)
adr ist die Adresse der nächsten verfügbaren Speicherzelle im Wörterbuch.

PAD (-- adr)
adr ist die Adresse. eines Pufferspeichers mit wenigstens 84 byte Länge, dessen Inhalt vom System bei Wörterbuch-Erweiterungen überschrieben wird.

TIB (-- adr)
adr ist die Adresse des Text-Eingabe-Puffers mit wenigstens 80 byte Länge.

#TIB (-- adr)
adr ist die Adresse einer Variablen, die die Anzahl der im Text-Eingabe-Puffer (TIB) gespeicherten Zeichen enthält.

WORD (8b -- adr)
Entnahme eines mit Trennzeichen 8b abgegrenzten Wortes aus dem Text-Eingabe-Puffer (TIB) ab >IN und Abspeichern als FORTH-Zeichenkette mit zusätzlichem Leerzeichen (zählt nicht zur Länge) oberhalb HERE. adr ist die Anfangsadresse dieser Zeichenkette, >IN wird aktualisiert.

>IN (-- adr)
adr ist die Adresse der Variablen, die die Position des aktuellen Zeichens im Text-Eingabe-Puffer (TIB) enthält.

COUNT (adr1 -- adr2 +n)
+n ist das Zählbyte der FORTH-Zeichenkette ab adr1, adr2 die Adresse des ersten Zeichen.

7.4. Das Verwalten von Zeichenketten

FORTH-Systeme führen den Bediener-Dialog auf der Basis von Wortnamen, also Zeichenketten. Einige Elemente des dabei verwendeten Text-Interpreters gehören zum FORTH-83-Standard und lassen sich für die Zeichenketten-Verarbeitung nutzen. Eine wichtige Adresse, nämlich das Ende des Wörterbuchs und damit den Anfang des noch freien Speichers liefert das Wort **HERE** (hier, **Abb. 28**). Ein Stückchen (meist 64 byte) darüber legt das FORTH-System einen Zwischenspeicher namens PAD an, in dem beim Übersetzen Zeichenketten analysiert und manipuliert werden. Während einer Wortausführung bleibt der Übersetzer passiv, steht dieser Bereich dem Nutzer zur Verfügung.
Bei der Tasten-Eingabe wird ein Text-Eingabe-Puffer benutzt, dessen Anfangsadresse vom System mit der Variablen **TIB** festgelegt wird. Nach jeder Eingabe erhält eine zweite Variable, **#TIB** das zugehörige Countbyte, die Anzahl eingegebener Zeichen. Zur Übernahme der einzelnen Wörter aus solch einem Eingabestrom benutzt das FORTH-System **WORD** (Wort). Damit werden anhand des Trennzeichens 8b (meist Leerzeichen) einzelne Eingabewörter separiert und in einem selbständig ermittelten Freibereich oberhalb HERE im FORTH-Format abgelegt. Das Einbeziehen der Variablen **>IN** gestattet das schrittweise Abarbeiten mehrerer aneinandergereihter und mit Trennzeichen eingeschlossener Eingabe-Zeichenketten. WORD sichert, daß >IN nicht über #TIB hinauswächst. Beim Übergang vom ausführenden zum Dialogbetrieb werden >IN und #TIB stets gelöscht. Eine mit WORD im Anwenderprogramm verarbeitete Zeichenkette muß vor dem Start dieses Anwenderprogramm bereits eingetastet werden, da sonst nichts im Text-Eingabe-Puffer auffindbar wäre.
COUNT (Anzahl) entnimmt einer FORTH-Zeichenkette das Zählbyte und legt es auf den Stock. Damit werden die für die Ausgabe von Zeichenketten mit TYPE (Abb. 24) benötigten Parameter bereitgestellt.

Teil 7, JU+TE 7/1990, S. 82

8. Programmabbruch

Das Beenden einer Wortausführung und der Rücksprung in das rufende Wort erfolgt grundsätzlich am Ende. Manchmal ist es aber sinnvoll, den Programmablauf vorzeitig zu beenden. Nach Erkennen eines Fehler wird, zum Beispiel, die weitere Verarbeitung überflüssig. Das vorzeitige Verlassen einer DO-LOOP-Schleife gelingt mit LEAVE. Dieses Wort kennen wir bereits (vgl. Abb. 17). Für einen Abbruch außerhalb von DO-LOOP-Schleifen eignet sich **EXIT** (**Abb. 29**). Es beendet die Wortausführung und führt zum Rücksprung in das rufende Wort, auch wenn

das Ende noch nicht erreicht ist. Es entwspricht dem RETURN-Befehl auf Maschinenniveau bzw. in verschiedenen höheren Programmiersprachen Für einen Fehler, der auch die Fortsetzung hinter dem rufenden Wort unsinnig erscheinen läßt, eignet sich der Abbruch mit **QUIT**. Dabei bleibt der Datenstack erhalten. Die weitere Arbeit steuert die nächste Eingabe z. B. vom Bediener.

29 Abbruch

EXIT (--)
Abbruch der Wortausführung und Rücksprung in die rufende Wortfolge hinter die Aufrufstelle.
Achtung! Im FIG-Standard nicht enthalten!

QUIT (--)
Return-Stack säubern, ausführenden Modus einschalten und Dialog über aktuellen Eingabegerät beginnen.

ABORT (--)
Stack säubern und Ausführung von QUIT.

ABORT" (? --)
Wenn Flag ? wahr ist, werden die folgende Zeichenkette ohne abschließendes " ausgegeben und ABORT ausgeführt.
Achtung! Im FIG-Standard nicht enthalten!

Mit ABORT werden vor dem Erwarten neuer Engaben sowohl Daten. als auch Return-Stack geleert. Für ein Abbruch im Fehlerfall mit einer geeigneten Mitteilung empfiehlt sich **ABORT"**. Es ist im Gegensatz zu den anderen ein immediate-Wort (Unmittelbar-Wort). Es wird normalerweise innerhalb von Doppelpunkdefinitionen nicht übersetzt, sondern ausgeführt. Damit eignet es sich für Meldungen beim Übersetzen und ist beim Ergänzen von Definitionswörtern (die ähnlich CREATE oder : funktionieren) nützlich. Hiermit kommen wir nun bereits dem FORTH-Kern, also dem System schon sehr nahe.

9. Systemnahe Wörter

Natürlich benötigt auch ein FORTH-System Anpassungsprogramme für die konkrete Gerätetechnik des jeweiligen Rechners.
Auch dieser Teil von FORTH-83 ist mit einer Gruppe von Standard-Wörtern organisiert. Das gestattet dem Anwender, auch sehr systemnahe Probleme mit FORTH zu lösen und eine eigene (Fach-) Sprache aufzubauen, der man zum Schluß den FORTH-Kern gar nicht mehr ansieht. Diese Wörter erlauben den Zugang zu Kernbereichen, der sonst nur auf Maschinen-Niveau möglich ist.

9.1. Das Teilen des Wörterbuchs

Beim Verwenden eines bereits vergebenen Namens für eine neue Definition funktionieren zwar alle bereits definierten Wörter mit dem ursprünglichen Namensinhalt weiter, aber für spätere Verwendung wird diese alte Version nicht mehr gefunden. Das kann stören, wenn z.B. für + eine Gleitkomma-Routine vielleicht sogar mit Infix-Notation geschrieben wird, die FORTH-Addition aber weiter zur Verfügung stehen soll. Das läßt sich umgehen, wenn neue Wörter ein neues Vokabular als Organisationsform erhalten. Neben dem FORTH-Vokabular können damit weitere, die einen beliebigen Namen als Oberbegriff erhalten, im Wörterbuch angelegt werden. Trotz gleicher Namen bleiben damit FORTH-Standard-Definitionen auffindbar, indem einfach die Suche in entsprechender Reihenfolge stattfindet.
Das FORTH-System realisiert dieses Teilen des Wörterbuchs einfach mit Hilfe der Link-Felder, die auf die letzte Definition des jeweiligen Vokabulars verweisen. Erst wenn die priorisierte durchsucht wurde, erfolgt die Fortsetzung mit der nächsten in Form eines Vokabulars vereinigten Gruppe von Wörtern.
Der Name jedes Vokabulars, der Oberbegriff, wird stets als erstes definiert. Mit dem Wort **VOCABULARY** (**Abb. 30**) gelangt er in das Wörterbuch. Dos Wort **DEFINITIONS** bewirkt, daß folgende Definitionen unter dem zuletzt gerufenen Oberbegriff eingeordnet werden. Sollen wieder die Standard-Definitionen die Priorität beim Suchen erhalten, läßt sich mit FORTH eine entsprechende Rangfolge einstellen. Erneutes Ausführen von DEFINITIONS würde neue Einträge wieder an das FORTH-Vokabular anhängen. WORDS (soweit überhaupt vorhanden) zeigt übrigens auch entsprechend der jeweils geltenden Suchreihenfolge an

30 Vokabulare

VOCABULARY (--)
Definitionswort :zum Benennen eines neuen Vokabulars in der Form VOCABULARY name. Die Ausführung von name setzt das neue Vokabular an die Spitze der Suchreihenfolge.

DEFINITIONS (--)
Dos Vokabular an der Spitze der Suchreihenfolge wird zum Compilationsvokabular.

FORTH (--)
Dos FORTH-Vokabular gelangt an die Spitze der Suchreihenfolge.

FORTH-83 (--)
Anzeige der Verfügbarkeit eines FORTH-83-Systems

Das Wort FORTH-83 gibt Auskunft über die Vollständigkeit des Standard-Wortschatzes. In älteren Versionen ist es natürlich nicht enthalten und bewirkt bei Aufruf eine entsprechende Fehlermeldung.
Auch die in den folgenden Abbildungen dargestellten Standardwörter sind überwiegend für FORTH-83 neu gestaltet und weichen von FIG sind FORTH-79 ab. Das ist ab Abb. 30 nicht besonders gekennzeichnet.

9.2. Das Speichern von Quelltext

Das Arbeiten im FORTH-Dialog ist sehr effektiv und macht Fehler kurzfristig erkennbar. Zur Korrektur muß aber nicht nur die betreffende, sondern auch jede darauf aufbauende Definition neu eingegeben werden. Um das zu vereinfachen, gestattet FORTH-83, den Quelltext zu speichern. Er läßt sich dann nach Korrekturen insgesamt aufs neue übersetzen.
Ursprünglich wurde (beim FIG) der Inhalt eines Bildschirms (screen) als Quelltext-Einheit gespeichert. Bei FORTH-83 hat ein Quelltextblock stets eine Kapazität von einem Kilobyte, egal, wieviel auf den Bildschirm paßt. Für eine gewisse (vom konkreten Computer abhängige) Anzahl solcher Blöcke existieren Pufferbereiche im RAM. Der gesamte Ouelltext kann aber mehr Blöcke enthalten, als Puffer bereitstehen. Er wird auf Diskette (oder einem anderen Massenspeicher) aufbewahrt und bei Bedarf Blockweise in Puffer geladen. Die dafür nötigen Wörter gehören zum Standard-Wortschatz und können also auch in Anwenderprogrammen verwendet werden
Das Wort **BUFFER** (Puffer, **Abb. 31**) reserviert Platz für einen Block im RAM, dessen Anfangsadresse auf den Stack gelegt wird, **BLOCK** veranlaßt außerdem eventuell nötige Massenspeicher-Zugriffe. Die Systemvariable **BLK** steuert das Übersetzen des Quelltextes. Die Blocknummer 0 ist dabei dem bisher besprochenen Dialog per Tastatur-Eingabe-Puffer (TIB) zugeordnet.

31 Massenspeicher

BUFFER (u – adr)
Zuweisen eines Blockpuffers mit Anfangsadresse adr zum Block u, ohne eventuell nötiges Laden zu veranlassen.

BLOCK (u – adr)
adr ist die Startadresse des Block u zugewiesenen Blockpuffers. Wenn nötig werden dort ein befindlicher Block in den Massenspeicher transportiert und Block u von dort geladen.

BLK (-- adr)
Adresse der Variablen, in der die Nummer des Blocks steht, aus dem eingegeben wird (Nummer 0: TIB).

LOAD (u --)
Sichern der Inhalte von >IN und BLK, Löschen von >IN und Laden von BLK mit u sowie Eingabe von Block u. Noch dessen Erschöpfung Laden von >IN und BLK mit den gesicherten Werten.

UPDATE (--)
Kennzeichnen des aktuellen Blockpuffers als aktualisiert.

SAVE-BUFFERS (--) Transport aller aktualisierten Blöcke zum Massenspeicher.

FLUSH (--)
Ausführung von SAVE-BUFFERS und Freigabe alter Blockpuffer.

Teil 8, JU+TE 8/1990, S. 84-85

Fortsetzung zu 9.2.: Das Speichern von Quelltext

Mit **LOAD** (laden) kann„ das Übersetzen eines Blocks mittem im Programm eingefügt werden. Das ist sehr hilfreich, wenn ein Anwenderprogramm in einem bestimmten Zustand zusätzliche Hilfen wie erweiterte Arithmetik oder Ein-/Ausgabe-Unterstützung benötigt. Dos Wort **UPDATE** kennzeichnet einen im Puffer befindlichen Block als verändert (aktualisiert), so daß er mit SAVE-BUFFERS, FLUSH oder BLOCK in dieser neuen Form auf den Massenspeicher (Diskette) gelangt.
SAFE-BUFFERS (Puffer retten) aktualisiert den Disketteninhalt, **FLUSH** (ausspülen) gibt außerdem alle Puffer für eine Neu-

vergabe mit BUFFER oder BLOCK frei. All diese Wörter können im Dialog, innerhalb des Quelltextes und in Wortdefinitionen verwendet werden. Die lose Zuordnung von Puffern zu Blöcken gestattet auf einfache Weise die virtuelle Speicherverwaltung, d. h. die Arbeit mit mehr Speicherkapazität, als im RAM gleichzeitig unterzubringen ist. Da Quelltext im Gegensatz zu Wörterbucheintragungen viel Kapazität braucht, erweist sich diese Eigenschaft als sehr bedeutend bezüglich weit gesteckter Anwendungsgrenzen.

9.3. Interpreter

Abb. 32 stellt die Standardwörter zusammen, die das Interpretieren von Eingaben betreffen. Das Wort ´ (tick) legt die Compilationsadresse (Aufrufadresse) des mit dem folgenden Namen versehenen Wortes auf den Stack. **>BODY** ermittelt daraus die Parameterfeldadresse, also den Ort für Ergänzungen wie Zeichenketten. Während ´ den Namen des zu suchenden Wortes im Anschluß erwartet, verwendet **FIND** (finden) eine FORTH-Zeichenkette an beliebiger anzugebender Stelle. Mit **EXECUTE** (Ausführen) wird die auf dem Stack liegende Adresse gerufen. Zum Einfügen von Kommentaren eignet sich **(**. Alle folgenden Zeichen einschließlich des abschließenden) übergeht der Interpreter. Mit **.(** und) werden ebenfalls nicht zu berücksichtigende Texte eingeschlossen, die jedoch auf dem Bildschirm erscheinen.

Damit lassen sich solche Meldungen wie .(BLOCK 7 GELADEN) in den Quelltext einfügen.

32 Interpreter

´ (-- adr)
Ermitteln der Compilationsadresse adr von name in der Form ´ name.

>BODY (adr1 -- adr2)
adr2 ist die Parameterfeldadresse des Wortes mit der Compilationsadresse adr1.

FIND (adr1 -- adr2 n)
Suchen des Wortnamens aus FORTH-Zeichenkette ab Adresse adr1 und Angabe dessen Compilationsadresse adr2 und des Parameters n. (n=1: Immediate-Wort, n=-1: kein Immediate-Wort, n=0: nicht fündig).

EXECUTE (adr --)
Ausführung des Wortes mit Compilationsadresse adr.

((--)
Kommentazeichen, das wie die folgenden Zeichen bis einschließlich des abschließenden) beim Übersetzen ignoriert wird.

.((--)
Kommentarzeichen. Die folgenden Zeichen ausschließlich des abschließenden) werden angezeigt, aber nicht übersetzt.

9.4. Der Compiler

Der Compiler ist der eigentliche Übersetzer, der aus Tastatur-Eingaben (über TIB) oder Quelltexten Wörterbucheinträge macht. Auch er läßt sich mit Standard-Wörtern steuern und z.T. in Anwenderprogrammen nutzen. Die Systemvariable **STATE** (Status, **Abb. 33**) läßt zwischen ausführender und übersetzender Betriebsart unterscheiden. Sie wird durch den Doppelpunkt (Beginn einer Definition) mit einer Zahl ungleich 0 geladen und durch das Semikolon wieder gelöscht. Auch mit den eckigen Klammern können wahlweise Compiler oder Executer aktiviert werden. Praktisch dienen beide Wörter dazu, innerhalb eines zu übersetzenden Quelltextes eine Berechnung auszuführen, deren Ergebnis mit **LITERAL** ins Wörterbuch gelangt. Bei der Ausführung der so übersetzten Definition gelangt dieses Ergebnis wieder auf den Stack.

33 Compiler

STATE (-- adr)
Adresse einer Variablen, die den ausführenden Modus mit 0 und den compilierenden mit einem Inhalt ungleich 0 kennzeichnet.

[(--)
Einschalten des ausführenden Modus.

] (--)
Ausschalten des compilierenden Modus.

IMMEDIATE (--)
Kennzeichnen des letzten Wörterbucheintrags als Immediate-Wort, das auch im compilierenden Modus ausgeführt und nicht übersetzt wird.

[COMPILE] (--)
Das folgende Wort wird übersetzt, auch wenn es ein Immediate-Wort ist.

LITERAL (--16b)
Beim Übersetzen wird der TOS als Zahl in die erzeugte Wortfolge eingetragen, der TOS wird gewöhnlich mit Wörtern vor LITERAL innerhalb der eckigen Klammern erzeugt. Bei der Ausführung der Wortfolge wird diese Zahl zum TOS transportiert.

[´] (-- adr)
Beim Übersetzen wird die Compilationsadresse adr von name (Struktur: [´] name) als Zahl gespeichert, die bei der Ausführung der übersetzten Wortfolge auf den TOS gelangt.

COMPILE (--)
Das folgende Wort name (Struktur: COMPILE name) wird nicht ausgeführt, sondern übersetzt. Sinnvoll innerhalb der Definition von Immediate-Wörtern.

DOES> (-- adr)
Festlegen des Ausführungsverhaltens beim Definieren von Definierungswörtern. In der Form: name...create...DOES>...; wird name ein Definitionswort zum Eintragen eines Wortes mit create (CREATE oder CREATE enthaltendes Wort) ins Wörterbuch, bei dessen Aufruf dessen Parameterfeldadresse adr auf dem Stack erscheint und die Sequenz zwischen DOES> und ; ausgeführt wird.

Mit **IMMEDIATE** (unmittelbar) wird der zuletzt definierte Wörterbucheintrag in die Reihe der auch im compilierenden Modus auszuführenden Wörter aufgenommen. Auf diese Weise lassen sich Erweiterungen des Compilers realisieren. Andererseits gestattet **[COMPILE]**, diesen Mechanismus für das folgende Wort außer Kraft zu setzen. Damit können auch Immediate-Wörter in Definitionen aufgenommen werden. Das eröffnet noch mehr Möglichkeiten für Compiler-Erweiterungen.
Die letzten beiden Wörter sind wichtig für das Definieren von Definitionswörtern. Dabei müssen wir gedanklich das Definieren solcher Wörter, das Ausführen dieser Wörter zum Definieren anderer Wörter und die Ausführung dieser Wörter unterscheiden. Das sind drei verschiedene Ausführungszeitpunkte, die schon beim Planen des ersten zu berücksichtigen sind. Mit **COMPILE** kann ein Definitionswort Komponenten des zu definierenden ins Wörterbuch bringen. **DOES>** gestattet sogar, eine Folge von Wörtern (zwischen DOES> und ;) zum ersten Zeitpunkt festzulegen, die aber erst zum dritten zur Ausführung kommen. Natürlich kann ein Anfänger mit solchen Mechanismen nichts anfangen. Alle im Kapitel 9 genannten Standardwörter werden zunehmend erst dann interessant, wenn mittels FORTH Programmiersprachen ganz neu entstehen. Bis zu diesem Zeitpunkt reichen die in Abb. 4 zusammengestellten Definitionswörter völlig aus.
Abb. 34 gibt als Abschluß unseres FORTH-83-ABC einen Überblick des Standard-Wortschatzes. Er enthält die Wörter in alphabetischer Folge sowie Verweise auf die näher beschreibenden Abbildungen. Ein I kennzeichnet Immediate-Wörter.

Dr. Helmut Hoyer

34 FORTH-83-Standard-Übersicht

!	21
#	26
#>	26
#S	26
#TIB	28
'	32
(	32I
*	5
*/	7
*/MOD	7
+	5
+!	21
+LOOP	17I
,	22
-	5
-TRAILING	24
.	25
."	24I
.(	32I
/	5
/MOD	7
0<	14
0=	14
0>	14
1+	6
1-	6
2+	6
2-	6
2/	6
:	4
;	4I
<	14
<#	26
=	14
>	14
>BODY	32
>IN	28
>R	12
?DUP	11
@	21
ABORT	29
ABORT"	29I
ABS	6
ALLOT	22
AND	9
BASE	24
BEGIN	18I
BLK	31
BLOCK	31
BUFFER	31
C!	21
C@	21
CMOVE	23
COMPILE	33
CONSTANT	4
CONVERT	27
COUNT	28
CR	24
CREATE	4
D+	8
D<	14
DECIMAL	24
DEFINITIONS	30
DEPTH	11
DNEGATE	8
DO	17I
DOES>	33I
DROP	11
DUP	11
ELSE	16I
EMIT	24
EXECUTE	32
EXIT	29
EXPECT	27
FILL	23

FIND	32
FLUSH	31
FORGET	4
FORTH	30
FORTH-83	30
HERE	28
HOLD	26
I	17
IF	16I
IMMEDIATE	33
J	17
KEY	27
LEAVE	17I
LITERAL	33I
LOAD	31
LOOP	17I
MAX	19
MIN	19
MOD	5
NEGATE	6
NOT	9
OR	9
OVER	11
PAD	28
PICK	11
QUIT	29
R>	12
R	12
REPEAT	18I
ROLL	11
ROT	11
SAVE-BUFFERS	31
SIGN	26
SPACE	24
SPAN	27
STATE	33
SWAP	11
THEN	16I
TIB	28
TYPE	24
U.	25
U<	14
UM*	8
UM/MOD	8
UNTIL	18I
UPDATE	31
VARIABLE	4
VOCABULARY	30
WHILE	18I
WORD	28
XOR	9
[	33I
[']	33I
[COMPILE]	33I
]	33

32-Bit-Erweiterung

2!	21
2@	21
2DROP	13
2DUP	13
2OVER	13
2ROT	13
2SWAP	13
D+	8
D-	8
D0=	15
D2/	8
D<	14
D=	15
DABS	8
DMAX	20
DMIN	20
DNEGATE	8
DU<	15
D.	25
D.R	25