Basic ist out — Es lebe Forth

Viele sind des Basic überdrüssig geworden. Die Sprache Forth ist eine Alternative. Unsere Einführung in Forth beginnt an einem konkreten Beispiel: der Programmierung des Decompilers.

Vor einiger Zeit erhielt ich ein Programm, das sich nach dem Starten mit »FIG-FORTH« meldete. Forth, soviel wußte ich schon, soll eine neue, ungewöhnliche Programmiersprache sein. Nichts Genaues weiß ich nicht. Also gebe ich ein: LIST und RUN und PRINT 5*3 und NEW und ... nichts passierte, außer, daß mein Computer antwortet: »MESS # 0« oder »MESS # 1« oder manchmal auch »OK«. Also erst einmal zur Seite damit ...

Aber diese Sprache läßt mich nicht los — eben weil ich nicht weiß, was ich damit anstellen kann. Bis ein Freund anruft: »Versuch mal VLIST. Ich habe da was gelesen ...« Der Bann ist gebrochen: Nun flitzen plötzlich Worte über den Bildschirm: AND und BASE und BUFFER und CONSTANT und IF und HEX und noch viel mehr, über 200 Worte. Das ist offensichtlich die Befehlsliste von Forth. Und dann bekomme ich die Fotokopie einer Kopie einer Kopie der Kurz-Befehlsbeschreibung eines FIG-Forth-Systems, kaum lesbar, in englisch, aber immerhin. Ich lese die Beschreibung — nein, ich buchstabiere sie beim ersten Mal, zu neu ist das Konzept dieser Sprache. Beim zweiten Lesen fasziniert mich Forth schon; nach dem dritten Lesen steht für mich fest: Basic ist out, Forth ist die Sprache der Zukunft.

Dies war vor etwa einem Jahr. Seitdem ist mir der Aufbau von Forth um einiges klarer geworden. Forth ist, so merkwürdig das klingt, zum großen Teil in Forth geschrieben. Zum Verständnis ist es deshalb wichtig, die Forth-Befehle zu rekonstruieren. Im Computer-Speicher steht aber nicht der Quelltext, sondern das compilierte, also übersetzte Programm. Forth läßt es jedoch zu, den Quelltext aus dem Speicher fast vollständig zu »decompilieren«. Deshalb wird ein interaktiver und einfacher Decompiler vorgestellt, der alle Forth-Worte als Quelltext listet. Das hilft auch bei der Optimierung und der Fehlersuche in eigenen Programmen. Für die folgenden Abschnitte sollte man jetzt seinen Computer einschalten und die — hoffentlich vorhandene — jeweilige Forth-Version laden. Es wird stillschweigend vorausgesetzt, daß man bereits einige erste Tippversuche in Forth hinter sich hat.

So arbeitet Forth

Forth ist eine Programmiersprache, die sowohl einen Interpreter als auch einen Compiler enthält. Nach dem Start eines Forth-Systems ist der Interpreter eingeschaltet. Er wartet auf »Eingangspost« (Input-Stream), die normalerweise von der Tastatur kommt. Diese Eingangspost liest der Interpreter und isoliert daraus das erste »Wort« (WORD), das heißt, eine Zeichenfolge, die durch Leerzeichen begrenzt ist und versucht, dieses Wort auszuführen. Ein solches Wort kann ein Befehl, eine Zahl oder auch eine sinnlose Zeichenkette sein.

Zunächst durchsucht der Interpreter sein »Wörterbuch« (DICTIONARY) nach dem isolierten Wort. Findet er es, so steht in dem zum Wort gehörigen Absatz des Wörterbuches, was zu tun ist. Diese Arbeitsanweisung wird dann von einem Kollegen des Interpreters ausgeführt (EXECUTE). Anschließend kehrt der Interpreter zurück zum Lesen der Eingangspost, isoliert das nächste Wort und blättert wieder im Wörterbuch.

Kann er ein Wort aber nicht finden, so versucht ein anderer Kollege (NUMBER), die Zeichen des Wortes in eine Zahl zu wandeln. Ist das gelungen, so wird die Zahl oben auf den »Stapel« (STACK) gelegt, wo sie für weitere Operationen zur Verfügung steht.

Kann der Interpreter das Wort weder im Wörterbuch finden, noch eine Zahl wandeln, so beschwert er sich mit einer Fehlermeldung, wirft die Zahlen auf dem Stapel und die noch ungelesene Eingangspost weg und wartet auf neue Post von der Tastatur.

Mit diesem interpretierenden Forth könnten wir Programme schreiben, indem wir eine sinnvolle Folge von Befehlen ausführen ließen. Jedoch, jeder Basic-Programmierer weiß, daß Interpretieren, also Lesen des Programmtextes, Nachsehen in einer Befehlsliste und das Wandeln in Zahlen während der Programmausführung lange dauert. Wollen wir schnelle Ausführung haben, so müssen wir unser Programm compilieren.

Im Forth ist der Compiler schon eingebaut. Es gibt einige Worte, deren Ausführung unter anderem darin besteht, den Interpreter aus- und den Compiler einzuschalten. Eins dieser Worte ist der Doppelpunkt »:« (COLON). Es nimmt das nächste Wort aus der Eingangspost und macht daraus den Kopf eines neuen Eintrags im Wörterbuch. Dann wird der Compiler eingeschaltet. Die folgenden Worte der Eingangspost werden nun nicht mehr ausgeführt, sondern der Compiler trägt »Zeiger« ins Wörterbuch ein. Diese Zeiger sind 16-Bit-Adressen, also 2 Byte lang und heißen Compilationsadressen. Sie weisen auf die entsprechenden Arbeitsanweisungen im Wörterbuch.

Das Eintragen von Zeigern macht der Compiler so lange, bis er wieder ausgeschaltet wird. Auch für diesen Zweck gibt es Worte: »;« schließt Worte ab, die mit »:« angefangen wurden. Es schaltet den Compiler wieder ab und den Interpreter ein. Im Wörterbuch ist jetzt ein neues Wort eingetragen, dessen Arbeitsanweisung aus einer Liste von Zeigern auf andere Worte besteht.

Ein praktisches Beispiel: Wir wollen von zwei Zahlen auf dem Stapel die oberste behalten, die zweite jedoch wegwerfen. Offenbar erfüllt die Befehlsfolge »SWAP DROP« genau diese Forderung, was der Leser bitte selbst prüfen mag. Wir wollen jedoch ein neues Wort NIP einführen, das unsere Forderung erfüllt:

```
: NIP (N1 N2 — N2) SWAP DROP;
```

Wir drücken RETURN, und Forth meldet sich nach dem Compilieren mit »OK«. Nun sehen wir uns an, was der Compiler aus diesem »Kurzbrief« gemacht hat. Dazu dumpen wir den Speicherinhalt mit der Befehlsfolge:

CR HEX ' NIP NFA 0E DUMP
und erhalten das folgende Listing:
83 4E 49 D0 37 58 3F 10 . NIP7X? .
86 F 7D F 33 E 4 44. . . . 3 . . D

Das sieht zunächst etwas wirr aus, doch sind die Bytes bei näherer Betrachtung sinnvoll. Ordnen wir den Speicherdump anders an und ergänzen ihn durch einige Kommentare, dann erhalten wir einen Aufbau wie in Bild 1. Im ersten Byte ist das höchstwertige Bit 7 immer gesetzt, Bit 6 und 5 haben besondere Bedeutung und die Bits 4 bis 0 enthalten die Länge des Namens, hier 3 (Bild 2). Die nächsten drei Byte enthalten dann tatsächlich auch den Namen NIP, nur das Bit 7 des letzten Buchstabens ist wieder gesetzt. Diese 4 Byte bilden das Namensfeld von NIP, gekennzeichnet durch die Namens-Feld-Adresse NFA. Die nächsten beiden Bytes sind das Linkfeld, markiert durch die Link-Feld-Adresse LFA. Im Linkfeld steht ein Zeiger auf das Namensfeld eines vorhergehenden Wortes. Die nun folgenden zwei Bytes bilden das Codefeld, adressiert durch die Code-Feld-Adresse CFA. Im Codefeld befindet sich ein Zeiger auf Maschinencode des jeweiligen Mikroprozessors.

Diese drei Felder bilden den Kopf des Wortes. Das Namensfeld dient dem Interpreter zum Wiedererkennen von »NIP«. Mit Hilfe des Linkfeldes findet der Interpreter zum vorhergehenden Wort, und von dort handelt er sich auf die gleiche Art weiter durch das Wörterbuch. Der

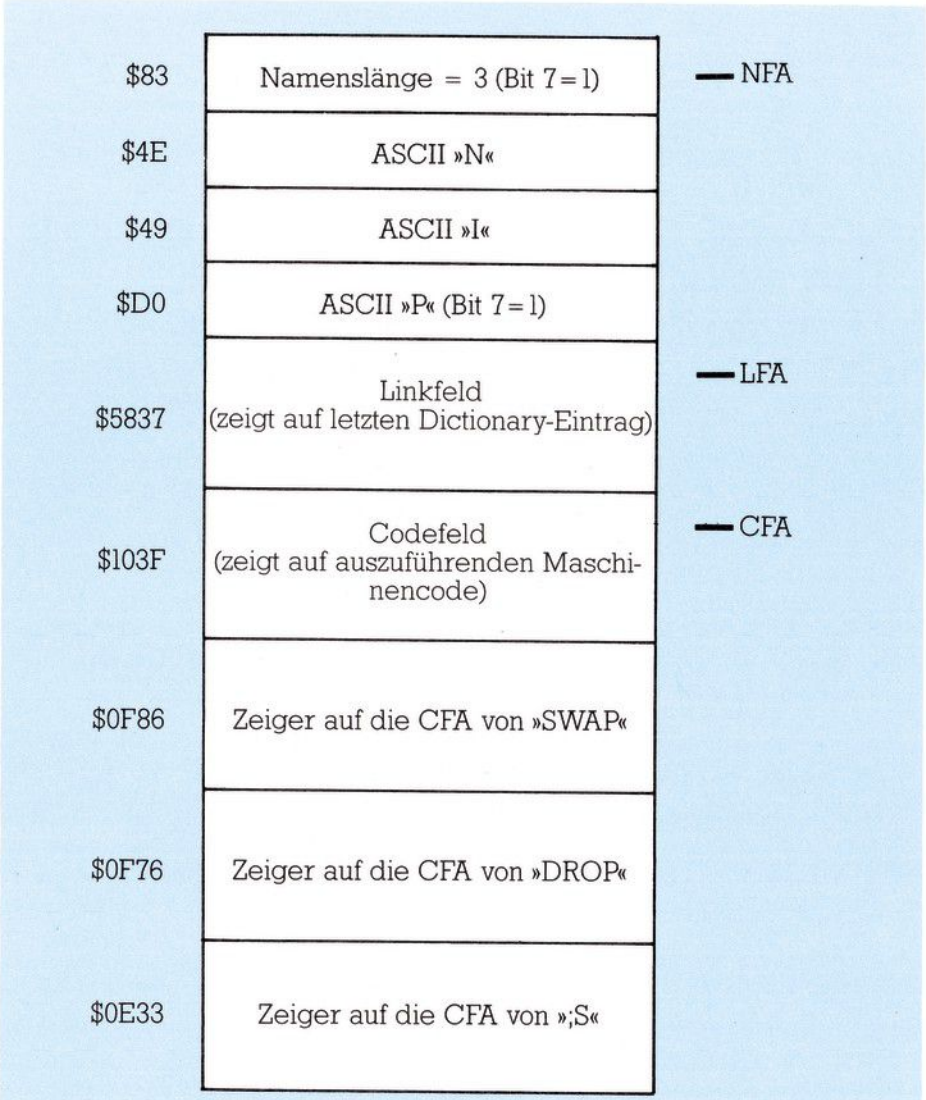

Bild 1. Der Aufbau des neuen Wortes »NIP«. Die angegebenen Adressen sind von der Forth-Version und der aktuellen Belegung des Dictionarys abhängig.

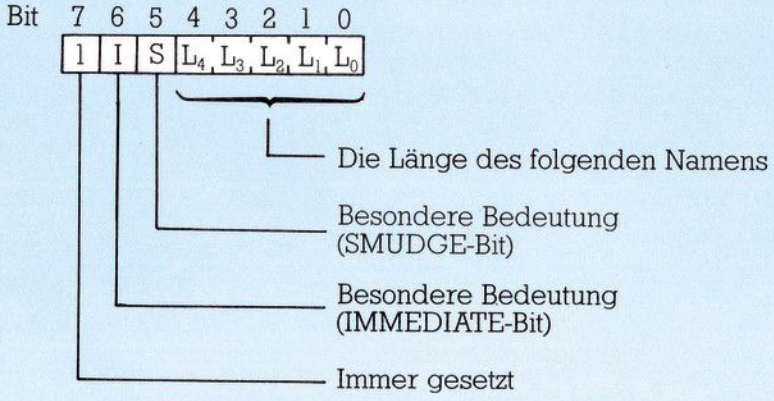

Bild 2. Das erste Byte im Namensfeld unter der Lupe

Inhalt des Codefeldes schließlich bestimmt den Charakter von »NIP«. Es gibt nämlich außer »normalen« Forth-Worten (die durch »:« eingeleitet und mit »;« beendet wurden) noch andere, zum Beispiel »Konstanten« (CONSTANT), »Variable« (VARIABLE) und reine Maschinencodeworte (CODE), wie zum Beispiel »SWAP« und »DROP«. Bei allen »normalen« Worten zeigt der Inhalt des Codefeldes jedoch immer auf denselben kurzen Maschinencode, der bewirkt, daß die im Speicher folgende Liste von Arbeitsanweisungen abgearbeitet wird.

An den Kopf mit seinen drei Feldern schließt sich der Rumpf nahtlos an, beginnend bei der Parameter-Feld-Adresse PFA. Das Parameterfeld ist die Liste von Arbeitsanweisungen und enthält bei unserem Wort NIP drei Zeiger zu 2 Byte; einen auf »SWAP«, einen auf »DROP« und einen Zeiger auf ein Wort »;S«. Bei der Ausführung von NIP beendet das »;S« das Abarbeiten dieser Liste. Wer hat das »;S« ins Wörterbuch geschrieben? Hier hat sich das »;« verewigt, das ja selbst noch einiges mehr tun muß (zum Beispiel den Compiler abschalten). Ganz und gar fehlt die Buchstabenfolge (N1 N2 — N2); dies ist auch gut so, denn mit einer öffnenden Klammer eingeleitete Texte sind Kommentare, der Text wird bis zur schließenden Klammer einfach überlesen.

Die Zeiger auf »SWAP«, »DROP« und »;S« weisen nun nicht auf deren Namensfeld, sondern auf deren Codefeld. Dies beschleunigt die späte-

re Ausführung von »NIP« ganz erheblich, erschwert uns jedoch das Wiedererkennen der Worte beim Speicherdump. Die Zeiger in »NIP« sind nicht nur computerabhängig, sondern auch abhängig von der Forth-Version und von der Stelle, an der »NIP« compiliert wurde. (Deshalb ist der oben gezeigte Speicherdump auch nur ein Beispiel.) Viel komfortabler wäre es, wenn wir statt Zeigeradressen zu dumpen, gleich die Wortnamen listen könnten, aus denen »NIP« sich zusammensetzt. Wir müßten dazu den Zeiger, der zum Beispiel auf die CFA von »SWAP« zeigt, so verbiegen, daß er auf die NFA von »SWAP« weist. Dann könnten wir feststellen, wie lang dieser Name ist und die vier Zeichen SWAP einfach ausdrucken. Diese Aufgabe erfüllt ein Decompiler.

Das Baukasten-Prinzip

Forth ist ein Baukasten, der die wichtigsten Funktionen schon als Bausteine enthält — sie müssen nur noch richtig zusammengesetzt werden. Also basteln wir uns ein Wort, das aus der CFA eines Wortes seinen Namen druckt. Zur Verfügung haben wir ein Wort »ID«, das aus NFA eines Wortes seinen Namen druckt. Das Problem reduziert sich also auf das Verbiegen der CFA auf die NFA, da die Namen verschieden lang sein können, Forth läßt bis 31 Zeichen zu. Auch hierfür gibt es schon ein Bauklötzchen, das verschieden lange Namen berücksichtigt: Habe ich eine PFA, so kann ich sie in die NFA umwandeln mit dem sinnfälligen Wort »NFA«. Unser Problem reduziert sich also weiter auf die Umwandlung einer CFA in die PFA. Das Codefeld enthält immer einen 16-Bit-Zeiger, ist also immer 2 Byte lang. Das Parameterfeld schließt sich immer an das Codefeld an. Also addieren wir zwei zur CFA, auch dafür gibt es ein Klötzchen namens »2+«, und erhalten die PFA, führen dann »NFA« aus und erhalten die NFA; danach drucken wir den Namen mit »ID« und ein trennendes Leerzeichen mit »SPACE«. Diesen neuen Befehl nennen wir »==« (Listing 1).

Nun können wir schon ein bißchen decompilieren: Wir suchen uns die Rumpfadresse von NIP mit:
' NIP
sichern uns die Adresse, holen uns den ersten Zeiger mit:
DUP @
und drucken aus der geholten CFA den zugehörigen Namen mit:
==

Auf dem Bildschirm erscheint »SWAP OK«. Jetzt erhöhen wir die Adresse um 2, damit sie auf den nächsten Zeiger weist, mit:
2+
und decompilieren das nächste Wort mit:
DUP @ ==

So könnten wir uns weiter durch NIP oder andere Worte tasten.

Tasten ist hier im doppelten Sinn treffend: Wir müssen viele Tasten drücken und kommen eben deshalb nur langsam voran. Etwas komfortabler muß es schon gehen.

Zunächst fassen wir die immer wieder benötigte Befehlsfolge »DUP @ == 2+« zusammen zum Wort ».NAME« (Listing 1).

Bei ».NAME« sind immerhin fünf Zeichen zu tippen, das ist jedem Forth-Programmierer zuviel. Außerdem hätten wir noch gerne gewußt, wo wir uns im Speicher gerade aufhalten und das ewige »OK« am Ende wollen wir uns beim Decompilieren auch sparen. Die Folge dieser Gedanken ist das Wort »N« aus Listing 1. Das »DUP U.« druckt uns die Adresse, die wir gerade decompilieren (ohne Vorzeichen), das ».NAME« den Namen und das »QUIT« am Ende unterdrückt die OK-Meldung. Es macht aber noch mehr, es bricht die Ausführung von Worten sofort ab, wirft den Rest der Eingangspost weg und kehrt in den Interpreter zurück. Nur der Zahlenstapel bleibt unangetastet.

Nun können wir NIP schon komfortabel decompilieren:
' NIP findet den Rumpf von NIP,
Ausgabe: OK
N decompiliert Namen,
Ausgabe: adr1 SWAP
N decompiliert Namen,
Ausgabe: adr2 DROP
N decompiliert Namen,
Ausgabe: adr3 ;S

```
Screen Nr. 12

0 ( Decompiler)  == .NAME N .LIT L                          04.08.84re)
1
2 FORTH DEFINITIONS   HEX
3
4 : ==   ( CFA ---)   2+ NFA ID. SPACE   ;
5
6 : .NAME   ( ADR --- ADR+2)   DUP @ == 2+   ;
7
8 : N   ( ADR --- ADR+2)   DUP U. .NAME QUIT   ;
9
A : .LIT   ( ADR --- ADR+2)   DUP @ . 2+   ;
B
C : L   ( ADR --- ADR+2)   DUP U. .LIT QUIT   ;
D
E --
F
```

Listing 1. Der erste Screen des Decompilers

Versuchen wir ein anderes Beispiel: Wir wollen uns ein Wort bilden, das zu einer gegebenen Adresse auf dem Stapel genau 1024 hinzuzählt, so daß wir uns in 1-KByte-Schritten durch den Speicher bewegen.
DECIMAL
: 1024+ (ADR — ADR+1024)
1024 + ;

Decompilieren wir nun unser Wort »1024+«, so erhalten wir als ersten Namen die Buchstabenfolge »LIT«, als zweiten Namen irgendwelchen Unsinn, als dritten Namen ein »+« und als vierten Namen ein »;S«. Das »+« und »;S« bedürfen keiner Erklärung, aber woher kommt das »LIT« und das, was sich da als Name nicht decompilieren läßt? Es ist unsere Zahl 1024, die sich durch das vorangestellte »LIT« als 16-Bit-Zahl zu erkennen gibt. Die Zahl selbst steht in den beiden Bytes, die der CFA von »LIT« folgen. Würde beim Compilieren die Zahl einfach nur ins Wörterbuch geschrieben, könnte bei der späteren Ausführung nicht mehr erkannt werden, daß es sich dabei eben nicht um die CFA eines Wortes handelt, sondern um eine Zahl, die auf den Stapel zu legen ist. Die praktische Folge: Stoßen wir beim Decompilieren auf »LIT«, so dürfen wir die nächsten beiden Bytes nicht als Name auffassen, sondern müssen sie als 16-Bit-Zahl einfach ausdrucken.

Basteln wir uns wieder die dazu nötigen Worte, zunächst ein
.LIT (ADR — ADR+2)
das die Adresse, die wir gerade bearbeiten, rettet und die beiden Bytes holt, die unter dieser Adresse zuhause sind. Diese beiden Bytes müssen dann als Zahl (mit Vorzeichen) ausgedruckt und anschließend die Adresse um 2 erhöht werden. Weiterhin ein kurzes Wort
L (ADR — ADR+2)

das ».LIT« benutzt (siehe wieder Listing 1).

Da sich Zahlen von 0 bis 255 in einem Byte darstellen lassen, compiliert Forth solche Zahlen, um Platz zu sparen, mit dem besonderen Wort »CLIT«. Dessen CFA folgt dann die Zahl in nur einem Byte. Auch dafür brauchen wir passende Worte (siehe Listing 2).

.CLIT (ADR — ADR+1)
und
C (ADR — ADR+1)

Nun gibt es noch eine Reihe von Worten, denen bei der Compilation eine nachfolgende 16-Bit-Zahl mitgegeben wird. Das sind die Verzweigungs-Befehle BRANCH, 0BRANCH, (LOOP) und (+LOOP). Die mitgegebene Zahl ist die Sprungadresse; jedoch ist sie nicht absolut angegeben, sondern als Offset zur Adresse, an der der Befehl compiliert wurde.

Schwierig? Wir müssen uns nur den Offset holen und die gerade laufende Adresse, an der wir decompilieren, addieren. Dies macht das Wort:

.BRANCH (ADR — ADR+2)
als Grundlage für:
B (ADR — ADR+2)
(siehe Listing 2).

Noch ein Wort würde uns beim Decompilieren Schwierigkeiten bereiten, wenn wir uns nicht ein »Gegenwort« basteln würden: das »."«. Ihm folgt ein Forth-String, der zur Ausführungszeit ausgedruckt wird und den wir ebenfalls ausdrucken und die laufende Adresse entsprechend erhöhen müssen. Ein Forth-String ist ähnlich aufgebaut wie das Namensfeld eines Forth-Wortes: Das erste Byte enthält die Länge des Strings, in den folgenden Bytes befinden sich die ASCII-Codes des Strings. Zum Ausdrucken von Strings bietet Forth zwei häufig benutzte Worte an: Haben wir die Adresse, bei der die ASCII-Zeichen des Strings beginnen und wissen wir die Länge, so druckt

TYPE (ADR Länge —)

den String aus. Haben wir jedoch nur die Adresse, unter der wir das Längenbyte eines Forth-Strings finden, so liefert

COUNT (ADR — ADR+1 Länge)

uns die um 1 erhöhte Adresse und das benötigte Längenbyte, so daß ein folgendes »TYPE« richtig vorbereitet ist. Diese beiden Worte benutzen wir in unserem

.STRING (ADR1 — ADR2)

aus Listing 2. Wir dürfen dabei nicht vergessen, die laufende Adresse um die Länge des Strings zu erhöhen, damit wir richtig weiter decompilieren können. Diesem Zweck dient das »2DUP« und das »+« am Ende von ».STRING«. Die Länge des Strings drucken wir uns vor dem String selbst aus. Das bedienerfreundliche Wort S (ADR1 — ADR2) aus Listing 2 benutzt dann wieder ».STRING«.

Werfen wir nochmals einen Blick auf das ».CLIT«. Dort finden wir auch unser »COUNT« wieder. Nun geht es bei ».CLIT« überhaupt nicht um Strings, das zweckentfremdete »COUNT« erfüllt aber genau das Geforderte: Das Byte unter der laufenden Adresse wird geholt, und die Adresse wird um eins erhöht.

Nun haben wir uns einen einfachen Satz von Worten geschaffen, der sämtliche Forth-Worte interaktiv decompiliert. Dies ist ein erheblicher Fortschritt gegenüber einem Speicherdump. Aber besonders komfortabel ist dieses Decompilieren immer noch nicht. Im zweiten Teil dieses Artikels wird deshalb ein automatischer Decompiler erscheinen, der sämtliche Standard-FIG-Forth Worte erkennt und vollständig decompiliert.

Der automatische Decompiler wird die hier vorgestellten grundlegenden Worte .NAME, .LIT, .CLIT, .BRANCH und .STRING benutzen. die Worte N, L, C, B und S sind durch das in ihnen enthaltene »QUIT« dem interaktiven Betrieb vorbehalten. Das interaktive Decompilieren fördert das Verständnis von Forth jedoch sehr und sollte von keinem Leser ausgelassen werden.

(Georg Rehfeld/ev)

```
Screen Nr. 13

0 ( Decompiler, .CLIT C .BRANCH B .STRING S          10.08.84re)
1
2 : .CLIT   ( ADR --- ADR+1)   COUNT .   ;
3
4 : C   ( ADR --- ADR+1)   DUP U. .CLIT QUIT   ;
5
6 : .BRANCH   ( ADR --- ADR+2)   DUP @ OVER + U. 2+   ;
7
8 : B   ( ADR --- ADR+2)   DUP U. .BRANCH QUIT   ;
9
A : .STRING   ( ADR --- ADR+)   COUNT 2DUP DUP . TYPE +   ;
B
C : S   ( ADR --- ADR+)   DUP U. .STRING QUIT   ;
D
E
F
```

Listing 2. Weitere Befehle zum Decompiler

Das Stapeln ist ein Forth-Prinzip: Bemerkungen zum Stapel.

Betrachten wir den Daten-Stapel etwas näher. Forth-Worte, die Eingangsvariable brauchen oder Ausgangsvariable liefern, benutzen in der Regel den Stapel, um diese Zahlen zu übergeben. Der Additions-Befehl »+« zum Beispiel nimmt die beiden obersten Zahlen vom Stapel, addiert sie und legt die Summe wieder oben auf den Stapel. Dies führt zu einer besonderen Schreibweise von Formeln, der Umgekehrten Polnischen Notation (UPN), wie sie auch bei den Taschenrechnern einer bekannten Firma in Gebrauch ist. Die Basic-Schreibweise »PRINT 2+3« liest sich in Forth: »2 3 + .«

Der Einfluß eines Forth-Wortes auf den Daten-Stapel wird im Quelltext bei jedem Wort als Kommentar angegeben. Das Wort »+« hat demzufolge den Stapel-Kommentar:

+ (N1 N2 — N3)

N1 und N2 sind dabei die Summanden, die vom Stapel genommen werden, die drei Bindestriche deuten die Operation an und N3 ist die Summe, die auf dem Stapel hinterlassen wird. Das oberste Element des Stapels (TOS) steht bei dieser Schreibweise rechts, vor der Ausführung von »+« ist also N2 Stapel-Spitze, danach N3.

Wichtige Befehle zur Stack-Kontrolle sind:

DUP (N — NN) dupliziert TOS
DROP (N —) entfernt TOS vom Stapel
SWAP (N1 N2 — N2 N1) vertauscht die beiden obersten Werte des Stapels
OVER (N1 N2 — N1 N2 N1) kopiert zweites Stapelelement als neuen TOS
ROT (N1 N2 N3 — N2 N3 N1) rotiert die obersten drei Stapelelemente